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.

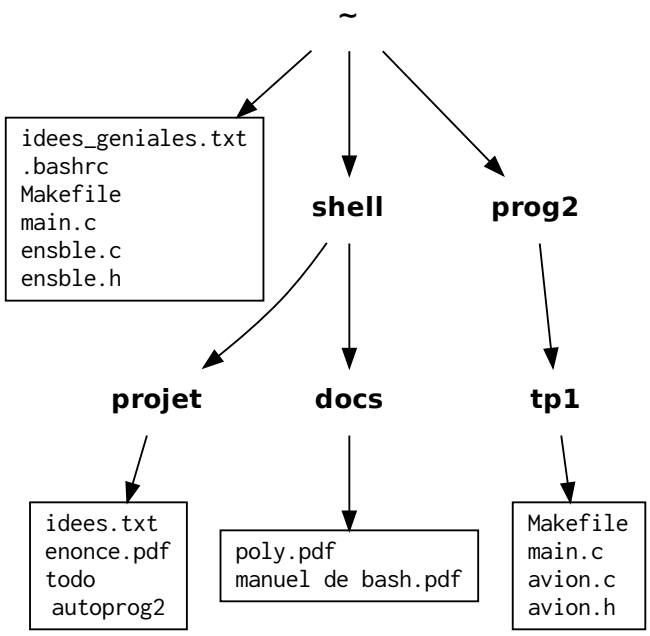
— Il est toujours possible de sauter des questions.

— 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`.
Correction :

```
mkdir prog2/tp2
cd prog2/tp2
touch main.c scooter.c 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** ?
Correction : Le chemin relatif est `../tp1/Makefile`. Le chemin absolu est `~/prog2/tp1/Makefile`

c) Copier ce fichier dans le répertoire **tp2**.
Correction : On peut utiliser le chemin relatif ou absolu, donc

```
cp ../tp1/Makefile .
# ou cp ~/prog2/tp1/Makefile .
```

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

```
sed 's/avion/scooter/g' Makefile > tmp
mv tmp Makefile
```

On peut utiliser aussi l'option `-i` de GNU pour remplacer directement dans le fichier, ...

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
```

Correction : Plein de possibilités, plusieurs `echo`, `printf`, `cat` (avec ou sans here-document).

```
echo '#ifndef SCOOTER_H' >> scooter.h
echo '#define SCOOTER_H' >> scooter.h
echo '#endif' >> scooter.h
```

```
cat >> scooter.h <<FIN
#ifndef SCOOTER_H
#include SCOOTER_H
#endif
FIN
```

```
printf '#ifndef SCOOTER_H\n#include SCOOTER_H\n#endif\n' >>
    scooter.h
```

à la place de `printf` (mais moins portable), il y a `echo -e` de GNU qui permet d'interpréter `\n` et sans doute encore 1000 possibilités...

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

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

Correction : `cd ../tp1`

b) Lister les fichiers de ce répertoire.

Correction : `ls # ou ls -l, ls -la, ou même echo *`

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

Correction : `grep 'avion_creer'*`

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

Correction : `grep '^#'avion.h`

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

Correction : `grep '.'avion.h` ou `grep -v '^$'avion.h` ou ...

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

Correction : `wc -l avion.h`

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

Correction : `grep '.'avion.h | wc -l`

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 ~.

Correction :

```
mkdir ~/prog2/tp_ensemble
mv ~/Makefile ~/main.c ~/ensemble.c ~/ensemble.h ~/prog2/↩
   tp_ensemble
```

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

Correction : `cd prog2 ; mv tp2 tp_scooter`

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

Correction : `cp -R tp1 tp_avion`

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

Correction : `rm -r tp1`

4. Permissions **/6**

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

Correction : `chmod u+x ~/shell/projet/todo ~/shell/projet/autoprog2`

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

Correction : `chmod og= ~/idees_geniales.txt` et 1000 autres possibilités

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

Correction : `chmod a-w ~/shell/projet/autoprog2`

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

Correction : Il faut enlever à tous la permission `w` au *répertoire* qui le contient. `chmod a-w ~/shell/projet/`

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 ?

Correction : Ce masque enlève les droits d'écriture au groupe et aux autres. Pour le fichier normal, créé au départ sans la permission `x`, ça donne les permissions `rw-r--r--` et pour le répertoire, créé avec la permission `x` avant d'appliquer le masque, ça donne `rwxr-xr-x`.

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`.

Correction : `evince ~/shell/docs/poly.pdf &`

b) Afficher le PID de ce processus.

Correction : `echo $!`

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

Correction : `kill $!`

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

Correction : `evince ~/shell/docs/"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).

Correction : `Ctrl-z` puis `bg`, ou à la place de `bg`, aller chercher le PID du processus, envoyer `SIGTSTP` avec `kill`, etc

6. Pot-pourri **/3**

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

Correction : C'est un fichier caché car son nom commence par un point. Pour le lister, on peut utiliser `ls -a`

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

Correction : `find ~ -name "*.pdf"`

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

Correction : `find ~ -type d`

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 ?

Correction : fichier normal, répertoire et lien symbolique.

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

Correction : Ce sont respectivement le propriétaire et le groupe propriétaire des fichiers.

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

Correction : Tous peuvent exécuter ce fichier. Rob peut en plus le lire et le modifier. En octal, ça fait `711`

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

Correction : C'est la taille du fichier en nombre d'octets. La date qui suit est la date de dernière modification.

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

Correction : C'est le nombre de liens physiques vers ce fichier.

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

Correction : Le fichier `urgent` est un lien symbolique qui pointe vers le fichier `attention` situé dans son répertoire parent.

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

a) `ls *.???`

Correction : Lister, dans le répertoire courant, les fichiers (non cachés) dont le nom se termine par un point, puis 3 caractères.

b) `kill -SIGCONT 38470`

Correction : Demander au processus de PID 38470 de reprendre son exécution.

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

Correction : Compter les fichiers (non cachés) du répertoire `/etc`

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

Correction : Mettre le contenu du fichier `c.txt` (situé dans le sous-répertoire `tata` du répertoire personnel) à la fin du fichier `a.txt` (situé dans le sous-répertoire `toto` du répertoire personnel).

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

Correction : Copier tous les fichiers du répertoire `/var/log` dont le nom :

- commence par une lettre minuscule entre `b` et `m` ;
- est suivi d'un caractère quelconque ;
- se termine par un caractère qui n'est pas un chiffre entre 3 et 9.

f) `ps -ef > psliste.txt`

Correction : Lister tous (option `-e`) les processus en cours, en format long (option `-f`) et mettre le résultat dans le fichier `psliste.txt`

g) `ps -ef | grep ^root`

Correction : Ne lister que les processus lancés par `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`

Correction : `ls *liste*.[ch]`

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.

Correction : Elle sert à dire au système quel interpréteur doit lire ce script.

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

Correction : La ligne 2 sert à mettre le contenu du premier argument dans une variable appelée `repertoire`. La ligne 3 met la chaîne `100` dans une variable appelée `limite`

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

Correction : La ligne 5 sert à mettre le résultat du calcul de 1024 fois la valeur de la variable limite dans la variable `limite_octet`. La syntaxe `$(( ))` sert à dire au shell d'en interpréter le contenu comme une expression arithmétique, de faire ce calcul et de remplacer toute la construction par le résultat du calcul (expansion arithmétique).

4. À quoi sert la ligne 6 ?

Correction : Elle sert à boucler sur tous les fichiers (non cachés) du répertoire dont le nom est `$repertoire`. À chaque itération, la variable `fichier` prend le nom d'un nouveau fichier du répertoire.

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 `$(  )` ?

Correction : Elle sera remplacée par la sortie standard de la commande qui est à l'intérieur (substitution de commande).

b) À quoi sert le symbole `|` ?

Correction : C'est un pipe qui sert à faire de la sortie standard de la première comande l'entrée standard de la commande suivante.

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

Correction : Elle sert à supprimer tout ce qui suit le nombre dans la sortie de la commande `wc -l`.

6. Expliquer en détail la ligne 8.

Correction : Il y a une construction `if` dont le corps ne sera exécuté que si la commande entre `if` et `;` sort avec un code de succès (`0`).

La commande en question est une commande `test` qui retourne `0` si le contenu de `taille` est supérieur au contenu de `limite_octet`.

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

Correction : Alors les `$` sont protégés et l'expansion de variables n'a pas lieu. La chaîne est copiée littéralement dans le terminal.

8. Que fait ce script ?

Correction : Il sert à afficher le nom des fichiers du répertoire donné en paramètre dont la taille dépasse 100Ko.

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

Correction : Correction globale de ces questions à la fin de l'exercice.

— 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.*

Correction :

```
#! /bin/bash

if [ "$#" -ne 1 ] ; then
    echo "usage : $0 REPERTOIRE" >&2
    exit 1
fi

repertoire=$1

if ! [ -d "$repertoire" ] ; then
    echo "L'argument $1 n'est pas un répertoire" >&2
    exit 2
fi

somme_tous=0
somme_gros=0
```

```
limite=100 #taille limite en Ko (kilo-octet)
limite_octets=$(( 1024 * $limite ))
for fichier in "$repertoire"/* ; do
    taille=$(wc -c "$fichier" | sed 's/ .*//')
    somme_tous=$(( $somme_tous + $taille ))
    if [ $taille -gt $limite_octets ] ; then
        echo "Le fichier $fichier fait plus de $limite Ko !"
        somme_gros=$(( $somme_gros + $taille ))
    fi
done

echo "Somme des tailles des fichiers de $repertoire : $somme_tous"
echo "Somme des tailles des fichiers de $repertoire dont la taille dépasse"\
     "$limite Ko : $somme_gros"
```

Exercice 4 : Copier en toute sécurité — 14 points

Correction : La correction est globale, à la fin de l'exercice.

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.

Correction :

```
#! /bin/bash
if [ "$#" -ne 2 ] ; then
    echo "usage : $0 SOURCE DESTINATION" >&2
    exit 1
fi
```

```
src=$1
dest=$2

if ! [ -e "$src" ] || ! [ -r "$src" ] ; then
    echo "le fichier $src n'existe pas ou n'est pas copiable" >&2
    exit 2
fi

if ! [ -f "$src" ] ; then
    echo "le fichier $src n'est pas un fichier ordinaire" >&2
    exit 3
fi

if [ -e "$dest" ] ; then
    reponse=""
    while [ "$reponse" != "o" ] && [ "$reponse" != "n" ] ; do
        echo "Le fichier $dest existe, l'écraser (o/n) ?"
        read reponse
    done
    if [ $reponse == "o" ] ; then
        cp "$src" "$dest"
    fi
else
        cp "$src" "$dest"
fi
```

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**.

Correction : Correction à la fin de l'exercice.

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/>
```

Correction :

```bash
#! /bin/bash

rep_img="$HOME/images"
rep_vid="$HOME/videos"
rep_doc="$HOME/documents"

for rep in "$rep_img" "$rep_vid" "$rep_doc" ; do
    if ! [ -e "$rep" ] ; then
        mkdir "$rep"
    elif ! [ -d "$rep" ] ; then
        echo "Le fichier $rep existe mais n'est pas un répertoire !" ↩
           >&2
        exit 2
    elif ! [ -x "$rep" ] || ! [ -w "$rep" ] ; then
        echo "Le répertoire $rep existe mais"\
             " il n'a pas les permissions suffisantes" >&2
        exit 3
```

```
    fi
done

for fichier in "$@" ; do # ou juste "for fichier ; do"
    case $fichier in
        *.png|*.gif) mv "$fichier" "$rep_img" ;;
        *.avi|*.mov) mv "$fichier" "$rep_vid" ;;
        *.pdf) mv "$fichier" "$rep_doc" ;;
        *.jpeg) mv "$fichier" "$rep_img" ;;
        *.jpg) nouveau_nom="$(basename -s .jpg "$fichier").jpeg"
            mv "$fichier" "$rep_img/$nouveau_nom" ;;
        *.JPG) nouveau_nom="$(basename -s .JPG "$fichier").jpeg"
            mv "$fichier" "$rep_img/$nouveau_nom" ;;
        *.JPEG) nouveau_nom="$(basename -s .JPEG "$fichier").jpeg"
            mv "$fichier" "$rep_img/$nouveau_nom" ;;
    esac
done
```