DISVE
Pôle Licence

ANNÉE UNIVERSITAIRE 2012/2013

DS UNIX et Initiation au projet programmation

Parcours/Étape : LIMI201 & LIMI2I1 **Code UE :** J1MI2014 **DS**
Épreuve : UNIX
Date : 22/04/2013 **Heure :** 08h30 **Durée :** 1h30
Documents papier autorisés **12** pages
Responsable du DS1 : O. Delmas

Répondre dans les cadres directement sur le sujet ou cocher lorsque nécessaire la ou les bonnes réponses (ATTENTION : cocher une mauvaise réponse retire des points.)
*Ce sujet comporte **8** pages sans les annexes et **12** pages avec les annexes*

Nom : Prénom :

Questions de cours

Exercice 1 : Cochez les états possibles d'un processus Unix.

- latent ☐
- en exécution ☐
- prioritaire ☐
- stoppé ☐
- en attente ☐
- ininterruptible ☐

Exercice 2 : Sous les systèmes du type Unix, quel est le *login* d'usage du super utilisateur ?

- obiwan kenobi ☐
- root ☐
- superuser ☐
- admin ☐

Exercice 3 : Sous le shell quelle(s) syntaxe(s) permet(tent) de rediriger la sortie d'erreur standard

- `<` ☐
- `>` ☐
- `<<` ☐
- `>>` ☐
- `2>` ☐
- `3>` ☐

Exercice 4 : Quel est le numéro du descripteur de fichier correspondant à l'entrée standard (`stdin`) :

- `-1` ☐
- `0` ☐
- `1` ☐
- `2` ☐
- `3` ☐
- `4` ☐

Exercice 5 : Que contient traditionnellement le répertoire `/usr/sbin` sous Unix ?

- des bibliothèques de logiciels ☐
- des programmes pour les utilisateurs ☐
- des programmes pour le super utilisateur ☐
- des fichiers de données ☐

Exercice 6 : Tous les processus se voient attribuer un PID unique par le noyau, il n'y a aucune exception.

- Vrai ☐
- Faux ☐

Exercice 7 : Généralement lorsque le code de retour d'une commande est 0 cela signifie

Une terminaison normale du processus ..... ☐ Une terminaison anormale du processus .... ☐

Exercice 8 : Quelle variable contient le code de retour de la dernière commande exécutée ?

`!` ..... ☐ `#` ..... ☐
`@` ..... ☐ `&` ..... ☐
`?` ..... ☐ `*` ..... ☐

Exercice 9 : Un processus en avant-plan est prioritaire sur un processus en arrière-plan.

Vrai ..... ☐ Faux ..... ☐

Exercice 10 : Le fonctionnement d'un shell se décompose par les 3 phases successives suivantes (relier par une flèche les cases de gauche en rapport avec celles de droite).

La première phase est la phase ..... ☐ ☐ ..... d'exécution
La deuxième phase est la phase ..... ☐ ☐ ..... de substitution
La troisième phase est la phase ..... ☐ ☐ ..... de saisie

Exercice 11 : La phase de substitution d'un shell se décompose par les 3 remplacements successifs suivants (relier par une flèche les cases de gauche en rapport avec celles de droite).

En premier ont lieu les remplacements ..... ☐ ☐ ..... de chemins
En deuxième ont lieu les remplacements ... ☐ ☐ ..... de variables
En troisième ont lieu les remplacements .... ☐ ☐ ..... de commandes

Exercice 12 : En sachant que le contenu du répertoire courant est (le caractère `$` en début de ligne représente le *prompt* de votre *shell*) :

```
$ ls -a
.  ..  bar.txt  .config  foo.txt  gif?.dat
```

Que produiront les commandes suivantes :

```
$ echo *
```

```
$ echo .*
```

```
$ echo .[^.]*
```

```
$ echo *[0-9]?
```

```
$ echo ?[aeiouy]?\?.*
```

```
$ echo *.[^t]*
```

Exercice 13 : Sous le shell, la phase de *découpage syntaxique* permet de découper une commande en un enchaînement de commandes simples. Par exemple la commande suivante :

```
date > RC.L ; ls -l .bashrc .emacs .bash_profile | grep rc >> RC.L
```

est découpée en une commande simple suivi d'une seconde commande :

`date > RC.L` ; `ls -l .bashrc .emacs .bash_profile | grep rc >> RC.L`

La seconde commande est quant à elle découpée en deux commandes simples :

`ls -l .bashrc .emacs .bash_profile` | `grep rc >> RC.L`

Maintenant, lisez attentivement le comportement suivant du shell :

```
$ cat foo.txt
Hello
$ a=foo.txt
$ echo 42 > ./$a
$ cat ./$a
42
$ COMMAND="echo do not panic"
$ ARGS="> ./$a"
$ echo $COMMAND
echo do not panic
$ echo $ARGS
> ./foo.txt
$ $COMMAND $ARGS
do not panic > ./foo.txt
$ cat ./$a
42
$ eval $COMMAND $ARGS
$ cat ./$a
do not panic
$
```

Que peut-on conclure après avoir examiné le comportement du shell sur l'exemple précédent (cocher les bonnes réponses) :

- La phase de substitution a lieu avant la phase des redirections ☐
- La phase de substitution a lieu après la phase de découpage syntaxique ☐
- La phase de traitement des redirections a lieu avant la phase de découpage syntaxique ☐
- La phase de substitution a lieu avant la phase de découpage syntaxique ☐
- La phase de traitement des redirections a lieu après la phase de découpage syntaxique ☐
- La phase de substitution a lieu après la phase des redirections ☐

Exercice 14 : Examinez la suite de commandes qui suit (le caractère $ en début de ligne représente le *prompt* de votre *shell*) :

```
$ ls
bar.txt  foo.txt
$ ls FOO.TXT 2> /dev/null && echo Hello >> FOO.TXT || (ls | wc -l)
```

1) **Représentez par des « patatoïdes » l'ensemble des processus créés au cours de l'exécution de cette ligne. Vous représenterez également le *shell* qui interprète cette ligne de commande. Vous représenterez, à l'aide de flèches entre les « patatoïdes », la filiation entre processus. Vous n'avez pas besoin de représenter les redirections. Il n'est pas non plus demandé d'expliquer le résultat produit par cette commande.**

2) **Lors de l'exécution des deux lignes de commandes précédentes, combien de processus au maximum coexistent simultanément ? (vous ne comptabiliserez pas le *shell* initial)**

- 1 processus ..... ☐
- 2 processus ..... ☐
- 3 processus ..... ☐
- 4 processus ..... ☐
- 5 processus ..... ☐
- 6 processus ..... ☐

Exercice 15 : Comment peut-on passer un processus en avant-plan dans l'état `STOP` ?

- ctrl-c ..... ☐
- escape ..... ☐
- ctrl-z ..... ☐
- ctrl-d ..... ☐

Exercice 16 : Quel est l'ordre de création des processus lorsque l'on tape une commande du type :

```
$ processus-1 | processus-2 | processus-3
```

- Les trois processus sont créés simultanément ☐
- processus-1 puis processus-2 puis processus-3 ☐
- processus-3 puis processus-2 puis processus-1 ☐
- Cela peut être n'importe quel ordre..... ☐

Exercice 17 : Tous les processus ont un PID et un PPID :

- VRAI, il n'y a aucune exception. ..... ☐
- FAUX, car le processus `init` n'a pas de PPID ..... ☐

Arborescence

Exercice 18 : À votre racine vous avez un répertoire `Téléchargement` et un répertoire `Projet_Programmation_1`. Dans le répertoire `Téléchargement` vous avez téléchargé deux fichiers archives `cours-1.tar.gz` et `cours-2.tar.gz`. L'archive `cours-1.tar.gz` contient les fichiers `statistiques.py`, `math.py`, `test-math.py` et `test-stats.py`. L'archive `cours-2.tar.gz` contient les fichiers `callback.py`, `bouton.py`, `canevas.py` et `test-tkw.py`. On suppose que l'utilisateur est dans le répertoire `Téléchargement`.

Donner une suite de commandes permettant de réorganiser le système de fichiers de la Figure 1(a) vers celui de la Figure 1(b). Cette réorganisation devra respecter les droits d'accès suivants :

```
-rwxrwxr-x math.py, -rwxrwxr-x statistiques.py
-rwxr-xr-x test-math.py, -rwxr-xr-x test-stats.py, -rwxr-xr-x test-tkw.py
-rwxr--r-- bouton.py, -rwxr--r-- callback.py, -rwxr--r-- canevas.py
```

On souhaite utiliser le moins de commandes possible.

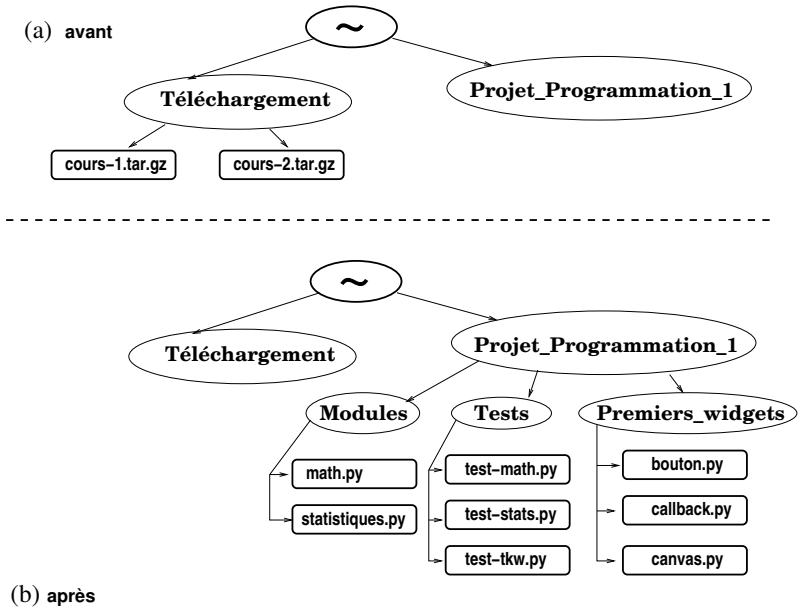

FIGURE 1 – (a) avant (b) après

Exercice 19 : Lecture d'un script-shell

Le script-shell `mystere.sh` prend deux paramètres : un entier suivi d'un nom de fichier.

```
$ cat mystere.sh
#!/bin/bash

USAGE="echo Usage: $(basename $0) num fich; exit 1"
if [ $# -ne 2 ]
then
    eval $USAGE
fi

test $1 -ne 0  2> /dev/null
if [ $? -ge 2 ]
then
    echo "erreur sur $1"
    eval $USAGE
fi

F1=$2-$(date +%m-%d-%Y)
F2=$(find . -mtime +$1 -type f)

if [ -z "$F2" ]
then
  echo "rien à faire"
else
  tar -czf $F1.tgz $F2
  mv $F1.tgz ..
  echo " \"$F1.tgz\" créé dans $(cd ..; pwd)"
fi
```

Que produira la commande `./mystere.sh un svg` ? Expliquez le résultat des différents tests.

Que produira la commande `./mystere.sh 10 svg` lancée à la date d'aujourd'hui dans un répertoire `Documents/projet` ? Expliquez l'affichage obtenu.

Exercice 20 : Écrire un script-shell

On dispose d'un fichier texte comportant une liste de numéros d'étudiants associés à une note sur 20. Ces deux informations sont séparées par un espace. Les notes sont des nombres entiers à 2 chiffres.

```
$ cat num-note.txt
429221 18
445071 14
444220 20
444454 08
444221 11
444140 14
444129 13
....
```

Préliminaire : écrire une ligne de commande permettant de compter le nombre de `11` apparaissant dans la colonne des notes.

Écrire un script-shell affichant pour chaque note entre 00 et 20 le nombre d'étudiants ayant obtenu cette note.

On dispose d'un second fichier comportant une liste de numéros d'étudiants associés à un nom. Ces deux informations sont séparées par un espace.

```
$ cat num-nom.txt
429221 Durand
445071 Dupond
444220 Ellier
444454 Eny
444221 Fall
444140 Fim
444129 Gaud
....
```

Écrire un script `note.sh` qui prend en paramètre un nom d'étudiant et qui affiche sa note.

Ligne de commandes

Exercice 21 :

Écrire une ligne de commande qui permet de recopier tous les fichiers du répertoire courant ayant pour suffixe `.txt` qui n'ont pas été modifiés depuis `24h` dans un sous-répertoire `OLD` du répertoire courant en rajoutant un suffixe `.old`. Par exemple, en supposant que le fichier `./f.txt` n'ait pas été modifié depuis au moins 24h, celui-ci sera recopié dans `./OLD/f.txt.old`. Si le répertoire `OLD` n'existe pas, il devra être créé sur la même ligne de commande.

Fin.

UNIVERSITÉ BORDEAUX 1 Sciences Technologies

Annexe 1 : Extrait de la page de manuel de la commande **seq**

```
NOM

seq - Affiche une séquence de nombres


SYNOPSIS

seq [OPTION]... DERNIER

seq [OPTION]... PREMIER DERNIER

seq [OPTION]... PREMIER INCREMENT DERNIER


DESCRIPTION

Affiche des nombres du PREMIER au DERNIER, par pas d'INCREMENT.

-f, --format=FORMAT
    utilise le style d'affichage FORMAT du même type que la fonction printf(3).
Exemple d'utilisation : seq -f %e 1 2 15

-s, --separator=STRING
    utilise la chaîne de caractères STRING pour séparer chaque nombre (par défaut : \n)

-w, --equal-width
    complète la largeur des nombres par des zéros

--help
    affiche cette aide et termine le programme

--version
    affiche les informations sur la version

Si l'argument PREMIER ou l'argument INCREMENT sont omis, ils sont
initialisés à 1.  PREMIER, INCREMENT, et DERNIER sont des
arguments utilisés en interne comme des valeurs en point flottant.

L'argument INCREMENT devrait être positif si FIRST est plus petit que
DERNIER, et négatif dans les autres cas.

Quand l'option FORMAT est utilisée, elle doit être exactement écrite
dans le style de la commande printf(3) manipulant les valeurs
flottantes. ( %e, %f, %g)
```

Annexe 2 : Extrait de la page de manuel de la commande **find**

```
FIND(1)

NOM
       find - Rechercher des fichiers dans une hiérarchie de répertoires
```

```
SYNOPSIS
       find [-H] [-L] [-P] [-D option-debogage] [-Oniveau] [chemin...] [expression]

[...]

OPTIONS

       Les  options  -H, -L et -P contrôlent le traitement des liens symboliques.

       -P     Ne jamais suivre les liens symboliques, ce qui est le
              comportement par défaut.
[...]

EXPRESSIONS

[...]

   OPTIONS

       -daystart

              Mesurer les temps (avec -amin, -atime, -cmin, -ctime, -mmin, et
              -mtime) depuis le début de la journée plutôt que depuis 24
              heures. Cette option n'affecte que les tests qui sont indiqués
              plus loin sur la ligne de commande.

       -depth Traiter d'abord les sous-répertoires avant le répertoire
              lui-même. L'action delete implique aussi -depth.
[...]
   TESTS

       Les paramètres numériques peuvent être indiqués comme suit :

       +n     supérieur à n,

       -n     inférieur à n,

       n      strictement égal à n.

       -amin n
              Dernier accès au fichier il y a n minutes.

       -anewer fichier
              Dernier accès au fichier plus récent que la dernière
              modification de fichier. Si le fichier est un lien symbolique
              et que les options -H ou -L sont actives, c'est toujours la
              date de dernier accès du fichier pointé qui est utilisée.

       -atime n
              Dernier accès au fichier il y a n*24 heures. Lorsque find
              calcule le nombre de périodes de 24 heures depuis lequel le
              fichier a été accédé, la partie fractionnelle est
              ignorée. Ainsi, pour correspondre à -atime +1, un fichier doit
              avoir été accédé il y a au moins deux jours.
```

```
[...]
       -mmin n
              Fichier dont les données ont été modifiées il y a n minutes.

       -mtime n
              Fichier dont les données ont été modifiées il y a n*24
              heures. Consultez l'explication sur -atime pour comprendre
              comment les arrondis affectent l'interprétation des dates de
              dernière modification des fichiers.

       -newer fichier
              Fichier modifié plus récemment que le fichier indiqué. Si le
              fichier fichier est un lien symbolique et que les options -H ou
              -L sont actives, c'est la date de modification du fichier
              pointé qui sera considérée.
[...]

   ACTIONS
       -delete
              Effacer les fichiers, et renvoyer vrai si l'effacement a
              réussi. Si l'effacement échoue, un message d'erreur est
              envoyé. Si -delete échoue, le statut de sortie de find sera
              différent de zéro (si jamais il s'interrompt). L'utilisation de
              l'action -delete active automatiquement l'option -depth.

       -exec commande ;

              Exécuter la commande ; vrai si le code de retour 0 est
              renvoyé. Tous les paramètres qui suivent find sont considérés
              comme des paramètres pour la ligne de commande, jusqu'à la
              rencontre d'un caractère « ; ». La chaîne « {} » est remplacée
              par le nom du fichier en cours de traitement, ceci dans toutes
              ses occurrences sur la ligne de commande, et pas seulement aux
              endroits où elle est isolée, comme c'est le cas avec d'autres
              versions de find. Ces deux chaînes peuvent avoir besoin d'être
              protégées du développement de la ligne de commande par le
              shell, en utilisant le caractère d'échappement (« \ ») ou une
              protection par des guillemets. Consultez la section EXEMPLES
              pour des exemples d'utilisation de l'option -exec. La commande
              indiquée est exécutée à chaque fois qu'un fichier correspond.
              La commande est exécutée depuis le répertoire de départ. Il
              existe d'inévitables problèmes de sécurité associés à l'usage
              de l'option -exec, c'est pourquoi vous devriez utiliser
              l'option -execdir à la place.

       -fprint fichier
              Vrai ; écrire le nom complet dans le fichier. Si fichier
              n'existe pas au démarrage de find, il est créé. S'il existe, il
              est écrasé.  Les noms de fichier « /dev/stdout » et «
              /dev/stderr » sont traités de manière particulière, ils
              correspondent respectivement à la sortie standard et à la
              sortie d'erreurs. Le fichier de sortie est toujours crée, même
              si le prédicat n'est jamais vérifié.
```

Annexe 3 : Extrait de la page de manuel de la commande **bash**

```
BASH(1)

NOM
       bash - Interpréteur de commandes GNU Bourne-Again SHell

SYNOPSIS
       bash [options] [fichier]

DESCRIPTION
       Bash est un interpréteur de commandes (shell) compatible sh qui
       exécute les commandes lues depuis l'entrée standard ou depuis un
       fichier. Bash inclut aussi des fonctionnalités utiles des
       interpréteurs de commandes Korn et C (ksh et csh).
[...]
CONDITIONS
       Les conditions sont utilisées par la commande composée [[ et les
       commandes internes test et [ pour vérifier l'état d'un fichier et
       effectuer des comparaisons arithmétiques ou sur des chaînes.

       -a fichier
              Vrai si le fichier existe.
       -d fichier
              Vrai si le fichier existe et est un répertoire.
       -e fichier
              Vrai si le fichier existe.
       -f fichier
              Vrai si le fichier existe et est un fichier normal.
       -h fichier
              Vrai si le fichier existe et est un lien symbolique.
       -r fichier
              Vrai si le fichier existe et est accessible en lecture.
       -s fichier
              Vrai si le fichier existe et a une taille strictement positive.
       -w fichier
              Vrai si le fichier existe et est accessible en écriture.
       -x fichier
              Vrai si le fichier existe et est exécutable.
       fichier_1 -nt fichier_2
              Vrai si le fichier_1 est plus récent que le fichier_2 (selon
              les dates de dernière modification) ou si fichier_1 existe et
              non fichier_2.
       fichier_1 -ot fichier_2
              Vrai si le fichier_1 est plus ancien que le fichier_2 ou si
              fichier_2 existe et non fichier_1.
       -o nom_opt
              Vrai si l'option d'interpréteur nom_opt est activée. Consulter
              la liste des options plus bas dans la description de l'option
              -o de la commande interne set.
       -z chaîne
              Vrai si la longueur de la chaîne est nulle.
       -n chaîne
              Vrai si la longueur de la chaîne est non nulle.
```