université de BORDEAUX Collège Sciences & Technologies	**Année universitaire 2013/2014** **DS Unix et Initiation au projet programmation** **Parcours/Étape :** MI201 **Code UE :** J1MI2014 DS **Épreuve :** Unix **Date :** 18 avril 2014 **Heure :** 11h00 **Durée :** 1h30 Documents papier autorisés 10 pages Responsable du DS2 : 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 sur la question.)

Ce sujet comporte 8 pages sans les annexes et 10 pages avec les annexes

Nom : Prénom :

Groupe : A1 ☐ A2 ☐ A3 ☐

Questions de cours

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

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

Exercice 2 : Sous Unix quelle est la signification de ppid.

parent priority idle ☐ parallel processing identifier ☐
parent process identifier ☐ parallel process instrumentation datagram ☐

Exercice 3 : Sous les systèmes du type Unix, quel est le *login* d'usage du super utilisateur ?

obiwan kenobi ☐ superuser ☐
root ☐ admin ☐

Exercice 4 : Sous Unix quel est généralement le nom du premier processus créé ?

kernel ☐ init ☐
root ☐ superuser ☐

Exercice 5 : Sous Unix quel est généralement le pid du premier processus créé ?

-1 ☐ 1 ☐
0 ☐ 10 ☐

Exercice 6 : Sous Unix quel(s) caractère(s) ne peut-on pas utiliser dans les noms de fichiers et de répertoires ?

* ..... ☐	" ..... ☐
/ ..... ☐	? ..... ☐
€ ..... ☐	aucun, tous les caractères sont utilisables ☐

Exercice 7 : Sous le shell quelle(s) syntaxe(s) permet(tent) de rediriger la sortie standard

< ..... ☐	>> ..... ☐
> ..... ☐	2> ..... ☐
<< ..... ☐	3> ..... ☐

Exercice 8 : Quel est le numéro du descripteur de fichier correspondant à la sortie d'erreur standard (`stderr`) :

-1 ..... ☐	2 ..... ☐
0 ..... ☐	3 ..... ☐
1 ..... ☐	4 ..... ☐

Exercice 9 : Quel est le shell commun à toutes les machines Unix ?

bash ..... ☐	sh ..... ☐
zsh ..... ☐	ksh ..... ☐
csh ..... ☐	dash ..... ☐

Exercice 10 : Sous Unix quel est usuellement le code de retour d'une commande qui s'est terminée normalement ?

-1 ..... ☐	1 ..... ☐
0 ..... ☐	10 ..... ☐

Exercice 11 : Sous le shell que signifie la syntaxe de redirection `2>&1`

Le processus est lancé en arrière plan et `stderr` est redirigé dans un fichier nommé `1` ..... ☐
Fusion de la sortie d'erreur sur la sortie standard ..... ☐
Redirection de la sortie standard à la fin d'un fichier s'il existe ..... ☐
Redirection invalide ..... ☐

Exercice 12 : La phase de substitution d'un shell se décompose par les 3 remplacements successifs suivant (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 13 : Un processus dans l'état STOP

est terminé ..... ☐	ne consomme pas de temps processeur ☐
ne consomme pas de mémoire ..... ☐	consomme un pid ..... ☐

Exercice 14 : Que contient traditionnellement le répertoire `/lib` sous Unix ?

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

Exercice 15 : Comment appelle-t-on la partie du système d'exploitation en charge de décider du prochain processus qui sera exécuté par le processeur ?

le processus init ..... ☐
les bibliothèques ..... ☐
l'ordonnanceur ..... ☐
l'horloge interne ..... ☐

Exercice 16 : En sachant que le contenu du répertoire courant est (le caractère `$` en début de ligne représente le *prompt* de votre *shell*) :

```
$ ls -a
. .. b2m baobab baobab.???? baobab.conf base64 .base64.conf? .svn tcscan texi2dvi texutil
```

Que produiront les commandes suivantes :

```
$ echo .*
```

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

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

```
$ echo ?[aeiuoy]\?
```

```
$ echo *."????"
```

Exercice 17 : Un processus en avant-plan est prioritaire sur un processus en arrière-plan.

Vrai ..... ☐
Faux ..... ☐

Exercice 18 : Cochez les commandes permettant de modifier les droits d'accès de

```
-rw-r--r-- 1 moi theo 0 mai 30 15:57 bar
-rw-r--r-- 1 moi theo 0 mai 30 15:57 data
-rw-r--r-- 1 moi theo 0 mai 30 15:57 foo
```

en

```
--w-rw-r-x 1 moi theo 0 mai 30 15:57 bar
-rw-r--r-- 1 moi theo 0 mai 30 15:57 data
--w-rw-r-x 1 moi theo 0 mai 30 15:57 foo
```

chmod u=w,g=rw,o=rx * ..... ☐
chmod u=rwx,g=r,o=wx??? ..... ☐
chmod u+x,g+wr,o-r+w [a-z][a-z][a-z] .. ☐
chmod u=w,g=rw,o=rx??? ..... ☐
chmod o+x,g+wr,u-r+w [a-z][a-z][a-z] .. ☐
chmod u=rw,g=rx,o=r??? ..... ☐

Exercice 19 : Examinez la suite de commandes qui suit (le caractère `$` en début de ligne représente le *prompt* de votre *shell*) :

```
$ ls
42  XIII
$ 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 flêches 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 ..... ☐ 4 processus ..... ☐
2 processus ..... ☐ 5 processus ..... ☐
3 processus ..... ☐ 6 processus ..... ☐

Exercice 20 : En supposant que le chemin suivant est valide `~/../home/../usr/bin`, dites s'il s'agit d'un

chemin absolu ..... ☐ chemin relatif ..... ☐

Exercice 21 : Que produit l'exécution des commandes suivantes :

```
$ pwd
/home/smith
$ REPONSE=42
$ sh
$ echo "$(pwd) $REPONSE"
```

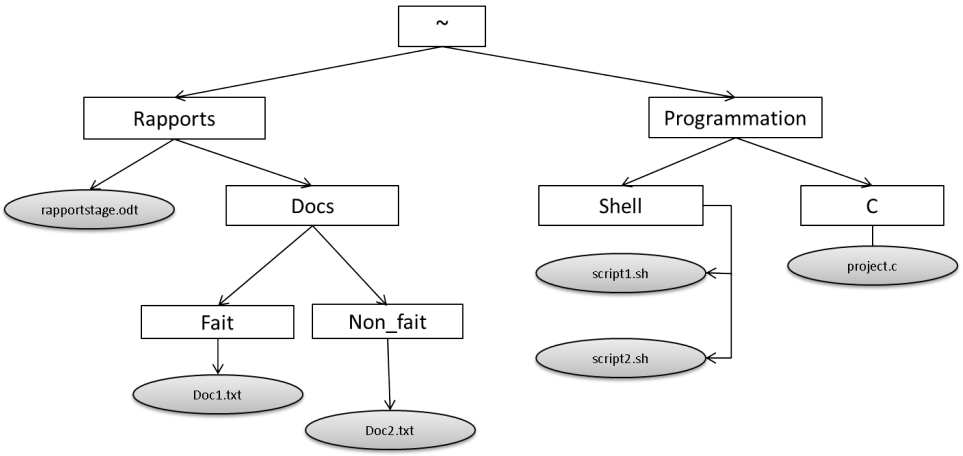
```
$ exit -1
$ echo $? ; (export FOO=bar; (echo $FOO))
```

```
$ echo "*$FOO"
```

Exercice 22 :

Dans votre répertoire d'accueil (HOMEDIR) vous disposez des fichiers : `Doc1.txt`, `Doc2.txt`, `project.c`, `script1.sh`, `script2.sh` et `rapportstage.odt` ainsi que de l'arborescences suivante : `Rapports/Docs/Fait`, `Rapports/Docs/Non_fait` et `Programmation/C` (le répertoire `Shell` n'existe pas). On supposera que les droits d'accès à vos fichiers sont `-rw-r--r--` et les droits d'accès à vos répertoires sont `drwxr-x---`

1) Donnez une suite de commandes permettant de réorganiser votre système de fichiers selon l'arborescence indiquée dans la figure ci-dessous :



2) Répondre aux questions suivantes en utilisant la suite de commandes nécessaires :

   a) Placez vous dans le répertoire `Shell`, vérifiez que vous y êtes puis renommez les fichiers `script1.sh` et `script2.sh` respectivement en `calcul1.sh` et `calcul2.sh`.

   b) Placez vous dans le répertoire `Rapports`. Sans bouger de ce répertoire, déplacez le fichier `Doc2.txt` dans le répertoire `Fait`, puis supprimez le répertoire `Non_fait`.

c) Placez vous dans le répertoire `Fait`. Concaténez les deux fichiers `Doc1.txt` et `Doc2.txt` dans un nouveau fichier `Doc3.txt` de telle sorte que si la concaténation des deux fichiers réussi vous effacerez automatiquement `Doc1.txt` et `Doc2.txt`, sinon ils ne seront pas effacés et le message `ECHEC` sera affiché.

Ligne de commandes

Exercice 23 : Écrire une ligne de commande qui permet d'effacer du répertoire courant et de ses sous-répertoires tous les fichiers dont le nom se termine par le caractère $\sim$ et qui ont une taille supérieure à 50 kilo-octets .

Exercice 24 : Écrire une ligne de commande qui permet d'afficher toutes les lignes du fichier `exemples.py` contenant le mot `if`, se terminant par `:` et dont le test comporte les symboles `==` ou `<` ou `>`. Par exemple :

```
if t[i] < t[j] :
if h == 24:
```

Script-Shell

Dans les exercices de cette section, on supposera que les droit d'accès aux fichiers et répertoires sont compatibles avec le bon fonctionnement des *script-shell*.

Exercice 25 : Lecture de *script-shell*

Soit le script suivant :

```
#!/bin/bash

for DIR in *
do
  if [ -d $DIR ] # teste si $DIR est un répertoire
  then
    cd $DIR
    for FILE in *
      do
        mv $FILE ../$DIR-$FILE
      done
    cd ..
    rmdir $DIR
  fi
done
```

Voici le contenu d'un répertoire `Photos`.

```
Photos/
|-- Plage
|   |__ DSC_0001.jpg
|   |__ DSC_0002.jpg
|__ Ski
    |__ DSC_0001.jpg
    |__ DSC_0002.jpg
```

On lance le script dans ce répertoire `Photos`. Comment est modifié le contenu du répertoire ? Expliquez en précisant le rôle de chacune des deux boucles for.

Exercice 26 : Écriture d'un *script-shell*

On souhaite renommer tous les fichiers contenus dans un répertoire. Le nouveau nom d'un fichier sera son ancien nom préfixé par l'année que l'on trouve dans la date associée au fichier. Dans l'exemple ci-dessous, l'année associée au fichier `DSC_0059.jpg` est `2014`, le nouveau nom du fichier sera donc `2014-DSC_0059.jpg`.

```
$ ls -l --full-time DSC_0059.jpg
-rw-rw-r-- 1 ens martin 947297 2014-04-02 10:48:38.220668966 +0200 DSC_0059.jpg
```

Écrire un script qui parcourt tous les fichiers du répertoire dont le nom est passé en paramètre et les renomme de la façon décrite ci-dessus. On peut supposer que le répertoire traité ne contient ni répertoire, ni fichier dont le nom commence par un point.

FIN.

Annexe 1 : 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.

       -depth
              Traiter d'abord les sous-répertoires avant le répertoire lui-même.

EXPRESSIONS

   OPTIONS

   TESTS

       Les paramètres numériques peuvent être indiqués comme suit :
       +n     supérieur à n,
       -n     inférieur à n,
        n     strictement égal à n.

    -size n[bckw]
          Fichier utilisant n unités d'espace. Les unités sont des blocs de
          512 octets par défaut (ou si un suffixe 'b' suit le nombre n), des
          octets si un suffixe 'c' suit n, des kilo-octets si un suffixe 'k'
          est utilisé, ou des mots de 2 octets si un 'w' suit le nombre n. La
          taille ne prend pas en compte les blocs indirects, mais elle comptabilise
          les blocs des fichiers éparpillés pas encore alloués.

   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.

-ok commande ;
       comme -exec mais interroge d'abord l'utilisateur (en utilisant
       l'entrée standard). Si la réponse ne commence pas par 'y' ou 'Y',
       la commande n'est pas exécutée, et le test devient faux.

-print
       vrai ; affiche le nom complet du fichier sur la sortie standard,
       suivi d'un saut de ligne.
```

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

```
MKDIR(1)

NOM
       mkdir - Créer des répertoires

SYNOPSIS
       mkdir [OPTION]... RÉPERTOIRE...

DESCRIPTION
       Créer les RÉPERTOIREs s'il n'existent pas.

       Les paramètres obligatoires pour les options de forme longue le
       sont aussi pour les options de forme courte.

       -m, --mode=MODE
             utiliser le mode du fichier (comme avec « chmod »), et non au
             format umask (a=rw)

       -p, --parents
             créer des répertoires parent (répertoires intermédiaires) si
             nécessaire, sans générer d'erreur s'ils existent

       -v, --verbose
              afficher un message pour chaque répertoire créé

       -Z, --context=CONTEXTE
              attribuer le CONTEXTE de sécurité SELinux à tous les
              répertoires créés
```