DISVE
Pôle Licence

ANNÉE UNIVERSITAIRE 2011/2012

DS UNIX et Initiation au projet programmation

Parcours/Étape : LIMI201 & LIMI2I1 **Code UE :** J1MI2014 **DS**
Épreuve : UNIX
Date : 11/04/2012 **Heure :** 14h00 **Durée :** 1h30
Documents papier autorisés **14** 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 **7** pages sans les annexes et **14** pages avec les annexes*

Nom : Prénom :

Questions de cours

Exercice 1 : Quel est le shell généralement disponible sur toutes les machines Unix

- `csh` ☐
- `bash` ☐
- `sh` ☐
- `ksh` ☐
- `tcsh` ☐
- `zsh` ☐

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 standard

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

Exercice 4 : Quel est le numéro du descripteur de fichier correspondant à la sortie d'erreur standard (`stderr`) :

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

Exercice 5 : 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 6 : Généralement lorsque la valeur de retour d'une commande est différente de 0 cela signifie

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

Exercice 7 : Quelle variable contient le PID de la dernière commande lancée en arrière-plan ?

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

Exercice 8 : La valeur de retour d'un enchaînement de commandes simples séparées par des tubes est

celle de la commande la plus à gauche ..... ☐
celle de la commande la plus à droite ..... ☐
la somme de la valeur de la commande la plus à gauche et de la plus à droite..... ☐
la moyenne de la valeur de la commande la plus à gauche et de la plus à droite..... ☐

Exercice 9 : 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 10 : 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 11 : Tous les processus se voient attribuer un PID unique par le noyau, il n'y a aucune exception :

Vrai ..... ☐ Faux ..... ☐

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
. .. b2m baobab baobab.???? baobab.conf base64 .base64.conf? .svn tcscan texi2dvi texutil
vi vlc
```

Que produiront les commandes suivantes :

```
$ echo *
```

```
$ echo .*
```

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

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

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

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

Exercice 13 : Examinez la suite de commandes qui suit (le caractère `$` en début de ligne représente le *prompt* de votre *shell*) :

```
$ pwd
/tmp
$ ls | wc -l
2
$
```

Soit la ligne de commandes :

```
$ NB=10; ls | while read LIGNE
             do
               NB=$(expr $NB + 1)
               echo $NB
             done; echo $NB
```

a) 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.

On rappelle que `while` et `read` sont des commandes internes qui s'exécuteront ensemble au sein d'un même sous-shell.

b) Écrire le résultat produit par l'exécution de cette dernière ligne de commandes :

c) Lors de l'exécution de la ligne de commandes de la question 12-a), combien, au maximum, d'occurrences distinctes de la variable `NB` sont présentes simultanément dans la mémoire de l'ordinateur ?

`1` ..... ☐ `4` ..... ☐
`2` ..... ☐ `5` ..... ☐
`3` ..... ☐ `6 ou plus` ..... ☐

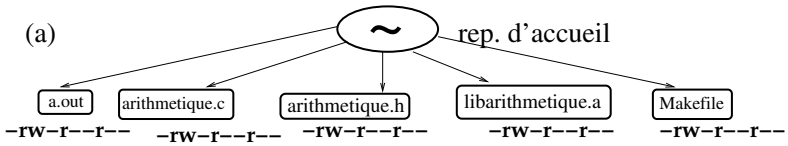
Arborescence

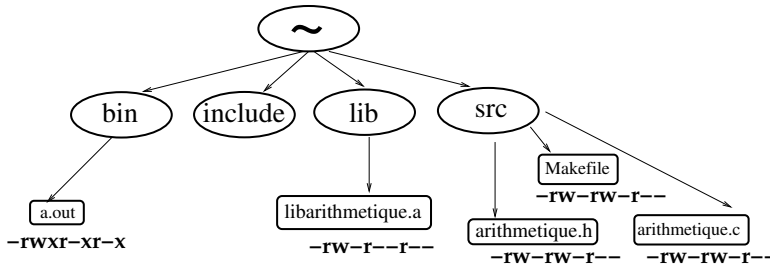
Exercice 14 : Dans le répertoire d'accueil on dispose des fichiers `a.out`, `arithmetique.c`, `arithmetique.h`, `libarithmetique.a` et `Makefile` avec les droits d'accès comme il est illustré sur la figure (a) ci-après.

On suppose que l'utilisateur est dans le répertoire d'accueil.

Donner la suite de commandes nécessaires pour réorganiser le système de fichiers comme indiqué sur la figure (b) ci-après (avec le contenu et droits d'accès correspondants). On souhaite utiliser le moins de commandes possible.

(a)





(b)

Lignes de commandes

Exercice 15 :

Le résultat de la commande `who` est le suivant :

```
$ who
delmas   pts/3        2012-04-03 15:41 (bok.labri.fr)
solange  pts/4        2012-04-03 14:18 (noc-out.u-bordeaux.fr)
durand   pts/5        2012-03-31 13:33 (abi-4-123-63.bdx.modulonet.fr)
durand   pts/6        2012-03-31 13:36 (abi-4-123-63.bdx.modulonet.fr)
martin   pts/7        2012-04-03 15:21 (143.51.112.118)
vialard  pts/9        2012-04-03 15:35 (vialard-laptop.labri.fr)
```

La dernière colonne donne le nom des machines d'où sont connectés les différents utilisateurs. Écrivez une ligne de commande qui affiche la liste de ces machines, sans les parenthèses, dans l'ordre alphabétique et sans doublons.

Exercice 16 :

1) On souhaite supprimer, à partir du répertoire courant et récursivement dans toute la sous-arborescence, tous les fichiers dont le nom se termine par un caractère '~' (même si le nom du fichier commence par un caractère '.') ou dont le nom débute et se termine par un caractère '#' (ces fichiers sont fréquemment des fichiers temporaires). L'option `-r` de la commande `rm` permet d'enlever le contenu des répertoires récursivement. Expliquez de façon précise pourquoi la commande suivante ne donne pas le résultat attendu :

   ```
   $ rm -r *~  .*~  \#*\#
   ```

2) En utilisant la commande find, écrivez une ligne de commande qui effectue la suppression des fichiers comme indiqué dans la 1ère question. La suppression devra fonctionner même à l'intérieur des sous-répertoires dont le nom commence par un caractère '.'.

Script-shell

Exercice 17 : 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

N=$(wc -l < $2)

if [ $N -lt $1 ]
then
   cat $2
else
   cat $2 | head -$(expr $1 - 1)
   cat $2 | tail -$(expr $N - $1)
```

Que contient la variable `N` ?

En supposant que le fichier `poème.txt` contienne :

```
$ cat poème.txt
Sous le vent qui chasse
Sous le vent qui chante
Le vent de la mer
Le coeur lourd dépasse
L'esprit qui le hante
```

Que produira la commande `$ ./mystere.sh 10 poème.txt` ? Expliquez.

Que produira la commande `$ ./mystere.sh 3 poème.txt` ? Expliquez.

Exercice 18 : Écrire un script-shell

La commande Unix `md5sum` permet de calculer une chaîne de 32 caractères à partir du contenu d'un fichier. On appelle *somme de contrôle* cette chaîne. Deux fichiers de même contenu auront la même *somme de contrôle* alors que deux fichiers de contenus différents auront des *sommes de contrôle* différentes.

Le but de cet exercice est d'utiliser cette commande pour détecter si le répertoire courant contient des doublons, c'est-à-dire au moins deux fichiers de noms différents mais de même contenu.

```
$ ls
doublons.sh  mystere.sh  poème.txt
$ ./doublons.sh
Il n'y a pas de doublon
$ cp poème.txt double.txt
$ ./doublons.sh
Il y a des doublons
$ md5sum *
abb0713b3687484f3705a6c6104d99c9  double.txt
9beadd00c1d15fa9f6e43b290c833487  doublons.sh
4d150da502edcabc475382e9fb368421  mystere.sh
abb0713b3687484f3705a6c6104d99c9  poème.txt
```

Écrire le script `doublons.sh`

FIN.

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

```
FIND(1)

NAME
       find - search for files in a directory hierarchy

OPTIONS

       -daystart
       Measure  times  (for  -amin, -atime, -cmin, -ctime, -mmin, and -mtime) from
       the beginning of today rather than from 24 hours ago.  This option only affects
       tests which appear later on the command line.


       -depth
        Process each directory's contents before the directory itself.
        The -delete action also implies -depth.


       -help, --help
       Print a summary of the command-line usage of find and exit.


       -maxdepth levels
       Descend at most levels (a non-negative integer) levels of directories below
       the command line arguments.  -maxdepth 0 means only apply the tests and actions
       to the command line arguments.


       -mindepth levels
       Do not apply any tests or actions at levels less than levels (a non-negative integer).
        -mindepth 1 means process all files except the command line arguments.


       -regextype type
       Changes the regular expression syntax understood by -regex and -iregex tests which
       occur later on the command line.

       -version, --version
       Print the find version number and exit.

       -warn, -nowarn
       Turn warning messages on or off.

       -xdev  Don't descend directories on other filesystems.

   TESTS

       Numeric arguments can be specified as

       +n     for greater than n,

       -n     for less than n,
```

```
n      for exactly n.

-amin n
File was last accessed n minutes ago.

-anewer file
File was last accessed more recently than file was modified.

-atime n
File was last accessed n*24 hours ago.

-cmin n
File's status was last changed n minutes ago.

-cnewer file
File's status was last changed more recently than file was modified.

-ctime n
File's status was last changed n*24 hours ago.

-empty
File is empty and is either a regular file or a directory.

-gid n
File's numeric group ID is n.

-group gname
File belongs to group gname (numeric group ID allowed).

-ilname pattern
Like -lname, but the match is case insensitive.

-iname pattern
Like -name, but the match is case insensitive.

-inum n
File has inode number n.

-ipath pattern
Behaves in the same way as -iwholename.

-iregex pattern
Like -regex, but the match is case insensitive.

-iwholename pattern
Like -wholename, but the match is case insensitive.

-links n
File has n links.

-lname pattern
File  is a symbolic link whose contents match shell pattern pattern.
```

-mmin n
File's data was last modified n minutes ago.

-mtime n
File's data was last modified n*24 hours ago.

-name pattern
Base of file name (the path with the leading directories removed) matches shell pattern pattern. The metacharacters ('*', '?', and '[]') match a '.' at the start of the base name.

-nogroup
No group corresponds to file's numeric group ID.

-nouser
No user corresponds to file's numeric user ID.

-readable
Matches files which are readable.

-regex pattern
File name matches regular expression pattern.

-samefile name
File refers to the same inode as name.

-type c
File is of type c:

b block (buffered) special

c character (unbuffered) special

d directory

p named pipe (FIFO)

f regular file

l symbolic link;

s socket

D door (Solaris)

-uid n
File's numeric user ID is n.

-used n
File was last accessed n days after its status was last changed.

-user uname
File is owned by user uname (numeric user ID allowed).

```
       -wholename pattern
       See -path.    This alternative is less portable than -path.

       -writable
       Matches files which are writable.

       -xtype c
       The same as -type unless the file is a symbolic link.
```

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

```
HEAD(1)                    Commandes                         HEAD(1)

NOM
       head - Afficher le début des fichiers

SYNOPSIS
       head [OPTION]... [FICHIER]...

DESCRIPTION

       Afficher les 10 premières lignes de chaque FICHIER sur la sortie standard.
       Avec plus d'un FICHIER, faire précéder chacun d'un en-tête donnant le nom
       du fichier. L'entrée standard est lue quand FICHIER est omis ou quand
       FICHIER vaut « - ».

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

       -c, --bytes=[-]K
              afficher les K premiers octets de chaque fichier ; avec le préfixe
              « - », afficher tous les octets sauf les K derniers octets de
              chaque fichier

       -n, --lines=[-]K
              afficher les K premières lignes au lieu des 10 premières ; avec le
              préfixe « - », afficher toutes les lignes sauf les K dernières
              lignes de chaque fichier
```

Annexe 4 : Extrait de la page de manuel de la commande **tail**

```
TAIL(1)                    Commandes                         TAIL(1)

NOM
       tail - Afficher la dernière partie de fichiers

SYNOPSIS
       tail [OPTION]... [FICHIER]...

DESCRIPTION
       Afficher les 10 dernières lignes de chaque FICHIER sur la sortie standard.
       Lorsqu'il y a plus d'un FICHIER, faire précéder chaque groupe de lignes
```

```
       d'un en-tête donnant le nom du fichier. L'entrée standard est lue quand
       FICHIER est omis ou quand FICHIER vaut « - ».

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

       -c, --bytes=K
              afficher les K derniers octets ; vous pouvez aussi utiliser +K pour
              afficher les octets de chaque fichier à partir du Kième octet

       -n, --lines=K
              afficher les K dernières lignes, au lieu des 10 dernières ; ou
              utilisez +K pour afficher toutes les lignes à partir de la Kième.
```

Annexe 5 : Extrait de la page de manuel de la commande **sort**

```
SORT(1)                          Commandes                               SORT(1)

NOM
       sort - Trier les lignes de fichiers texte

SYNOPSIS
       sort [OPTION]... [FICHIER]...

DESCRIPTION
       Afficher sur la sortie standard la concaténation triée de tous les FICHIERs.

       Options de tri :

       -n, --numeric-sort
              comparer selon la valeur numérique de la chaîne

       -r, --reverse
              inverser le résultat des comparaisons

       -k, --key=POS1[,POS2]
              utiliser la clé de tri commençant à POS1 (les positions sont comptées
              à partir de 1) et se terminant à POS2 (la fin de la ligne par défaut)

       -o, --output=FICHIER
              écrire le résultat dans le FICHIER à la place de la sortie standard

       -t, --field-separator=SÉPARATEUR
              utiliser le SÉPARATEUR à la place d'une transition d'un caractère non
              blanc vers un caractère blanc

       -u, --unique
              n'afficher que la première entrée identique rencontrée
```

Annexe 6 : Extrait de la page de manuel de la commande **uniq**

```
UNIQ(1)                         Commandes                              UNIQ(1)

NOM
       uniq - Signaler ou éliminer les lignes répétées

SYNOPSIS
       uniq [OPTION]... [ENTRÉE [SORTIE]]

DESCRIPTION
       Filtrer les lignes successives identiques du fichier d'ENTRÉE (ou de l'entrée
       standard), écrire dans le fichier de SORTIE (ou vers la sortie standard).

       Sans options, les lignes identiques sont fusionnées avec la première occurrence.

       -c, --count
              préfixer les lignes par le nombre d'occurrences

       -d, --repeated
              n'afficher que les lignes dupliquées

       -f, --skip-fields=N
              ne pas comparer les N premiers champs

       -s, --skip-chars=N
              ne pas comparer les N premiers caractères

       -u, --unique
              n'afficher que les lignes uniques

       -w, --check-chars=N
              ne pas comparer plus de N caractères par ligne

       Note : « uniq » ne détecte pas les lignes répétées, sauf si elles sont adjacentes.
              Vous voudrez sûrement trier l'entrée d'abord.
```