université de BORDEAUX Collège Sciences & Technologies	**Année universitaire 2014/2015** **DS Unix et Initiation au projet programmation** **Parcours/Étape :** MI201 **Code UE :** J1MI2014 DS **Épreuve :** Unix **Date :** 21 avril 2015 **Heure :** 08h30 **Durée :** 1h30 *Documents papier autorisés* 12 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 12 pages avec les annexes

Nom : Prénom :

Groupe : A1 ☐ A2 ☐ A3 ☐ A7 ☐

Questions de cours

Exercice 1 : Quel est le numéro du descripteur de fichier correspondant à l'entrée standard (`stdin`) :

`-1` ☐ `2` ☐
`0` ☐ `3` ☐
`1` ☐ `4` ☐

Exercice 2 : Sous le shell quelle(s) syntaxe(s) permet(tent) de rediriger la sortie d'erreur standard

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

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

obiwan kenobi ☐ superuser ☐
root ☐ admin ☐

Exercice 4 : Que contient traditionnellement le répertoire `/usr/sbin` sous Unix ?

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

Exercice 5 : Généralement lorsque le code de retour d'une commande est 0 cela signifie

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

Exercice 6 : Un processus en avant-plan est prioritaire sur un processus en arrière-plan

Vrai ☐ Faux ☐

Exercice 7 : Quelle variable contient le code de retour de la dernière commande exécutée ?

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

Exercice 8 : 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 9 : 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 10 : 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 (ce verbatim du shell se lit colonne par colonne, d'abord celle de gauche, puis celle de droite). Le caractère `$` en début de ligne représente le *prompt* de votre *shell*.

```
$ cat foo.txt
Hello world ;-)
$ FILE=foo.txt
$ echo 42 > $FILE
$ cat foo.txt
42
$ COMMAND="echo do not panic"
$ ARGS="> $FILE"
$ echo $COMMAND
echo do not panic
```

```
$ echo $ARGS
> foo.txt
$ $COMMAND $ARGS
do not panic > foo.txt
$ cat $FILE
42
$ eval $COMMAND $ARGS
$ cat $FILE
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 de traitement des redirections ..... ☐

Exercice 11 : Examinez la suite de commandes qui suit (le caractère `$` en début de ligne représente le *prompt* de votre *shell*)

```
$ ls
bar.txt  terre.txt
$ ls t*.txt 2> /dev/null && (ls | wc -l) || echo "Globalement inoffensif" >> terre.txt
```

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 12 : Pour vous débarrassez proprement d'un processus récalcitrant, indiquer dans la liste ci-après la commande la plus appropriée, (on supposera que le PID du processus que vous voulez supprimer a été placé dans la variable `PID`.

`kill -s SIGTERM $PID && kill -s SIGKILL $PID` ..... ☐
`kill -s SIGKILL $PID && kill -s SIGTERM $PID` ..... ☐
`kill -s SIGTERM $PID || kill -s SIGKILL $PID` ..... ☐
`kill -s SIGKILL $PID || kill -s SIGTERM $PID` ..... ☐

Exercice 13 : Indiquer dans les cadres les résultats produits par les commandes qui les précèdent (le caractère `$` en début de ligne représente le *prompt* de votre *shell*)

```
$ FOO=XIII; echo $FOO
```

```
$ M=\$FOO; echo $M
```

```
$ N=$M; echo $N
```

```
$ eval echo $N
```

Exercice 14 : 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.conf  .config.dat  foo.txt  gif?.dat
```

Que produiront les commandes suivantes :

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

```
$ echo .*
```

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

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

```
$ expr $(ls | wc -l) % 2
```

Exercice 15 : Indiquer dans les cadres les résultats produits par les commandes qui les précèdent (le caractère `$` en début de ligne représente le *prompt* de votre *shell*)

```
$ FOO=XIII && export BAR=42 ; echo $FOO $BAR
```

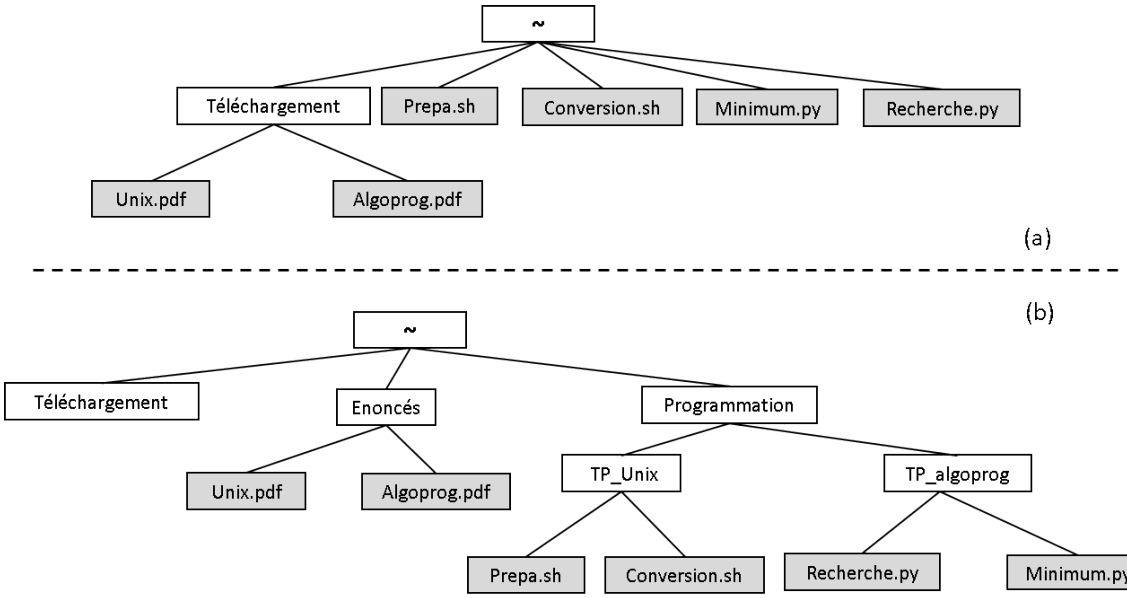
```
$ sh
$ echo $FOO $BAR; export FOO=X ; exit
```

```
$ echo $FOO $BAR
```

Exercice 16 :

On suppose que votre répertoire d'accueil (HOMEDIR noté ∼) contient le système de fichiers de la Figure 1 (a) (les répertoires sont en blanc, tandis que les fichiers sont en grisé). Les droits d'accès à vos fichiers sont `-rw-------` et ceux de vos répertoires sont (par défaut) `drwxr-xr-x`

1) Vous êtes dans votre HOMEDIR. Donner une suite de commandes permettant de réorganiser le système de fichiers de la Figure (a) vers celui de la Figure (b).



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

   a) Vous êtes dans le répertoire `TP_Unix`. Comment rendre les scripts `Prepa.sh` et `Conversion.sh` exécutables seulement par vous et accessible uniquement en lecture pour les membres de votre groupe Unix ?

   b) Vous êtes dans le répertoire `Enoncés`, donnez un chemin relatif valide et le chemin absolu valide du répertoire `TP_Unix`.

   c) Vous êtes dans le répertoire `Programmation`. Dans ce répertoire, créer un lien symbolique vers le répertoire `Téléchargement`.

Ligne de commandes

Exercice 17 : Écrire une ligne de commande qui permet d'afficher la liste des fichiers exécutables contenus dans une sous-arborescence du répertoire courant et dont le nom se termine par `.sh` et qui ont été modifiés plus récemment que le fichier `./monscript.sh`.

Exercice 18 : Le fichier `/etc/passwd` contient des informations relatives aux utilisateurs. Chaque ligne contient les 7 champs suivants : login, mot de passe chiffré, uid, gid, nom réel ou commentaire, répertoire d'accueil, shell. Les champs sont séparés par le caractère `':'` . Par exemple :

```
[...]
root:fi3sED95ibqR6:0:1:System Operator:/:/bin/bash
rachel:eH5/.mj7NB3dx:181:100:Rachel Zimmermann:/home/rachel:/bin/ksh
arlin:f8fk3j1OIf34.:182:100:Arlin Steinberg:/home/arlin:/bin/tcsh
[...]
```

Écrire une ligne de commande qui affiche uniquement les champs login et gid, séparés par un espace et triés dans l'ordre alphabétique selon les login.

Exercice 19 : On souhaite rechercher des lignes dans un programme `python`. On supposera que les noms de variables commencent toujours par une lettre, suivie éventuellement d'autres caractères choisis parmi les lettres, les chiffres ou le caractère souligné.

Écrire une ligne de commande qui permet d'afficher toutes les lignes d'un fichier `listes.py` contenant un nom de variable, suivi d'un point, suivi des mots `insert` ou `index`, suivis d'une suite non vide de chiffres encadrée par une paire de parenthèses. Par exemple :

```
liste1.insert(2)
une_liste.index(33)
```

Exercice 20 : Lecture de *script-shell*

Dans ce script, on suppose que les répertoires d'accueil des utilisateurs se trouvent dans `/home` :

```
 1   #!/bin/bash
 2
 3   cd /home
 4   for DIR in $(ls)
 5   do
 6      if [ -x $DIR -a -w $DIR/.bashrc ]
 7      then
 8         echo "echo don\'t panic !" >> $DIR/.bashrc
 9      fi
10   done
```

1) Expliquer en français quelles seront les valeurs successives de la variable `DIR` ?

2) À quoi servent les tests de la ligne `6` ?

3) Que fait la commande de la ligne `8` ?

4) Au final, que fait ce script ? Est-ce que certains utilisateurs verront que ce script a été exécuté ? Expliquez.

Problème

Le fichier pi.txt contient les cent mille premières décimales du nombre π [1]. Un extrait du contenu de ce fichier est présenté ci-dessous.

```
$ head -4 pi.txt; echo "[...]"
3.
1415 92653     589793gdfgf2384626   43383      27950
288419   716939937510(582!!09749    4459230
78 1640   6286   208,99862   803482534211706 fdf
[...]
```

Ce fichier débute par une ligne contenant uniquement la partie entière du nombre π c'est-à-dire `3.`, puis les lignes suivantes contiennent les décimales (et des caractères parasites).

Exercice 21 : Écrivez les commandes permettant de créer un fichier `decimales-pi.txt` contenant toutes les lignes du fichier précédent sauf la première. Ainsi, on élimine du fichier la partie entière du nombre π.

Ce fichier n'est malheureusement pas exploitable tel quel par les mathématiciens, car il contient parfois des caractères alphabétiques ou non-alphanumériques, parfois un ou plusieurs espaces entre deux décimales, parfois des tabulations et il contient un retour à la ligne à la fin de chaque ligne. Les mathématiciens souhaiteraient pouvoir travailler sur un fichier normalisé appelé decimales-pi.dat qui ne contienne que les

1. π est le rapport constant de la circonférence d'un cercle à son diamètre dans un plan euclidien

décimales du nombre π sans autres caractères que les chiffres. Voici un extrait du fichier decimales-pi.dat que l'on souhaite obtenir :

```
14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211
70679821480865132823066470938446095505822317253594081284811174502841027019385211055596446229489
8954930381964428810 9 [...]
```

Notez bien que dans ce fichier l'ensemble des décimales de π sont sur une seule et même ligne, il n'y a aucun retour à la ligne dans ce fichier.

Exercice 22 : Écrire la suite de commandes permettant d'obtenir le fichier normalisé `decimales-pi.dat` à partir du fichier initial `decimales-pi.txt` :

Les mathématiciens souhaitent faire différentes statistiques à partir de ce fichier normalisé.

Exercice 23 : Écrire la commande permettant de calculer le nombre de décimales présentes dans ce fichier :

Exercice 24 : Écrire un script `nb-occurrences-chiffre.sh`. Ce script prend deux paramètres, un chiffre et un fichier contenant des décimales. Il renvoie le nombre d'occurrences du chiffre (passé en paramètre) présent dans le fichier. Par exemple,

```
$ ./nb-occurrences-chiffre.sh 8 decimales-pi.dat
9978
```

Exercice 25 : Écrire un script-shell `statistiques-decimales.sh` qui prend en paramètre un fichier contenant des décimales. Ce script affiche sur chaque ligne de la sortie standard les chiffres (de 0 à 9) suivis de leur nombre d'apparitions dans le fichier passé en paramètre.

université de BORDEAUX

FIN.

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

```
EXPR(1)

NOM
       expr - Évaluer des expressions

SYNOPSIS
       expr EXPRESSION

DESCRIPTION
[...]
      PARAM1 + PARAM2
             somme arithmétique de PARAM1 et PARAM2
      PARAM1 - PARAM2
             différence arithmétique de PARAM1 et PARAM2

      PARAM1 * PARAM2
             produit arithmétique de PARAM1 et PARAM2

      PARAM1 / PARAM2
             division arithmétique de PARAM1 par PARAM2

      PARAM1 % PARAM2
             reste arithmétique de la division de PARAM1 par PARAM2
[...]
```

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

```
eval [arguments ...]

 (fr) Les arguments sont lus et regroupés en une seule commande
      simple. Cette commande est alors lue et exécutée par l'interpréteur et
      son état final est renvoyé comme valeur de la commande eval. S'il n'y
      a pas d'arguments ou uniquement des arguments vides, eval renvoie 0.

 (en) The  args  are  read and concatenated together into a single command.
      This command is then read and executed by the shell, and its exit status
      is returned as the value of eval. If there are no args, or only null
      arguments, eval returns 0.


Exemple: les caractères $> en début de ligne représentent le prompt du shell
  $> foo=10; x=foo
  $> y=\$$x
  $> echo $y
  $foo
  $> eval echo $y
  10
  $> _
```

Annexe 3 : 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.

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

EXPRESSIONS

   OPTIONS

   TESTS
       -executable
              Correspond  aux fichiers qui sont exécutables et aux répertoires
              qui sont accessibles (en ce qui concerne la résolution d'un  nom
              de  fichier).  Les  listes  de contrôles d'accès (ACL) et autres
              artefacts de permissions sont pris en  compte,  à  l'inverse  du
              test  -perm  qui lui les ignore.

       -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.

       -perm /mode
              Fichier ayant certains des bits d'autorisations indiqués dans le
              mode.  La  notation  symbolique  est  acceptée dans ce cas. Vous
              devez indiquer « u », « g » ou « o » si vous  utilisez  un  mode
              symbolique.

   ACTIONS

      -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. 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.
```

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

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

```
NOM
       test - Vérifier le type d'un fichier, et comparer des valeurs

SYNOPSIS
       test EXPRESSION

       [ EXPRESSION ]


DESCRIPTION
       Quitter avec un code de retour déterminé par EXPRESSION
       ...
       EXPRESSION doit être d'une des formes suivantes :

...
      EXPRESSION1 -a EXPRESSION2
            EXPRESSION1 et EXPRESSION2 sont vraies
...
      -w FICHIER
            FICHIER existe et est accessible en écriture

       -x FICHIER
            FICHIER existe et est exécutable (ou peut être parcouru dans le
            cas d'un répertoire)
```

Annexe 5 : 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.

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

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

```
GREP(1)

NOM
       grep - Afficher les lignes correspondant à un motif donné

SYNOPSIS
       grep [OPTIONS] MOTIF [FICHIER...]
       grep [OPTIONS] [-e MOTIF | -f FICHIER] [FICHIER...]

DESCRIPTION
       grep  recherche  dans les FICHIERs indiqués les lignes correspondant
       à un certain MOTIF. Par défaut, grep affiche les lignes qui contiennent
       une correspondance au motif. L'entrée standard est lue si FICHIER est
       omis ou si FICHIER vaut « - ».

OPTIONS
...
   Sélection des correspondances
       -E, --extended-regexp
              Interpréter le MOTIF comme une expression rationnelle étendue (ERE,
              voir ci-dessous). (-E est une spécification POSIX.)
       -P, --perl-regexp
              Interpréter le MOTIF comme une expression  rationnelle  Perl.

   Contrôle de correspondance
       -e MOTIF, --regexp=MOTIF
              Utiliser  le  MOTIF comme motif. Ceci peut être utilisé pour spécifier des
              motifs de recherche multiples ou protéger les motifs commençant par un
              tiret « - ». (-e est une spécification POSIX.)

       -i, --ignore-case
              Ignorer la casse aussi bien dans le MOTIF que dans les fichiers. (-i est
              une spécification POSIX.)
```

Annexe 7 : Extrait de la page de manuel de la commande **echo**

```
ECHO(1)

NOM
       echo - Afficher une ligne de texte

SYNOPSIS
       echo [OPTION-COURTE]... [CHAÎNE]...
       echo OPTION-LONGUE

DESCRIPTION
       Afficher la(les) CHAÎNE(s) en écho sur la sortie standard.

       -n     ne pas effectuer le saut de ligne final
       -e     interpréter les caractères déspécifiés par une contre-oblique
...
```