université de BORDEAUX Collège Sciences & Technologies	**Année universitaire 2015/2016** **DS Unix et Initiation au projet programmation** **Parcours/Étape :** MI201 **Code UE :** J1MI2014 DS **Épreuve :** Unix **Date :** 9 mars 2016 **Heure :** 11h00 **Durée :** 1h30 *Documents papier autorisés* 15 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 sur la question.)

Ce sujet comporte 9 pages sans les annexes et 15 pages avec les annexes

Nom : Prénom :

Groupe : A1 ☐ A3 ☐ A7 ☐ A8 ☐

Questions de cours

Exercice 1 : Sur un ordinateur comment appelle-t-on le (ou les) programme(s) qui a(ont) pour rôle de garantir l'intégrité de fonctionnement de l'ordinateur et qui arbitre l'accès concurrent des processus au matériel composant l'ordinateur ?

- le super utilisateur ☐
- le noyau et l'ordonnanceur ☐
- le window manager ☐
- le pare-feu ☐

Exercice 2 : Le fonctionnement d'un shell se décompose en 3 phases successives (relier par une flèche les cases de gauche en rapport avec celles de droite).

Phase 1 ☐	☐ exécution
Phase 2 ☐	☐ saisie
Phase 3 ☐	☐ substitution

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

1) Remplacements de ☐	☐ variables
2) Remplacements de ☐	☐ commandes
3) Remplacements de ☐	☐ chemins

Exercice 4 : Généralement lorsque le code de retour d'une commande est 0 cela signifie

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

Exercice 5 : Quelle variable contient le code de retour de la dernière commande exécutée ?

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

Exercice 6 : En représentant les processus par des « patatoïdes », dessiner les liens de parenté à partir de votre shell *bash* initial des processus engendrés par l'exécution de la ligne de commandes ci-après. Il n'est pas demandé de dessiner les redirections.

```
$ (ps aux | wc -l) ; date
```

Lors de l'exécution de la ligne de commande précédente, 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 7 : Cochez les affirmations qui sont vraies.

- Un utilisateur peut appartenir à plusieurs groupes Unix ☐
- À un instant donné, un utilisateur peut se placer dans un et un seul groupe Unix ☐
- Un fichier à un instant donné appartient à un groupe Unix ☐
- Un fichier à un instant donné peut appartenir à plusieurs groupes Unix ☐
- Les répertoires n'appartiennent à aucun groupe Unix particulier ☐

Exercice 8 : Quelle variable contient le *pid* du dernier processus détaché ?

- `!` ☐
- `@` ☐
- `?` ☐
- `$` ☐
- `&` ☐
- `*` ☐

Exercice 9 : Quelle variable contient le *pid* de la commande courante ?

- `!` ☐
- `&` ☐
- `?` ☐
- `$` ☐
- `@` ☐
- `*` ☐

Exercice 10 : Quel est l'opérateur d'enchaînement séquentiel de commandes ?

- `&` ☐
- `;` ☐
- `&&` ☐
- `||` ☐

Exercice 11 : Reliez par une flèche les commandes de gauche avec leur signification à droite.

$\langle cmd_1 \rangle$; $\langle cmd_2 \rangle$ ☐	☐ $\langle cmd_2 \rangle$ et $\langle cmd_1 \rangle$ sont lancées en parallèle
$\langle cmd_1 \rangle$ & $\langle cmd_2 \rangle$ ☐	☐ $\langle cmd_2 \rangle$ est lancée si $\langle cmd_1 \rangle$ retourne *vrai*
$\langle cmd_1 \rangle$ && $\langle cmd_2 \rangle$ ☐	☐ $\langle cmd_2 \rangle$ est lancée si $\langle cmd_1 \rangle$ retourne *faux*
$\langle cmd_1 \rangle$ || $\langle cmd_2 \rangle$ ☐	☐ $\langle cmd_2 \rangle$ est lancée lorsque $\langle cmd_1 \rangle$ est terminée

Exercice 12 : Comment peut-on passer un processus en avant-plan dans l'état STOP ?

Control-z ☐	Escape ☐
Control-c ☐	Control-d ☐

Exercice 13 : Que contient la variable d'environnement `HOME` ?

Le répertoire courant ☐	Le chemin absolu du répertoire d'accueil ☐
Le chemin relatif du répertoire d'accueil ☐	Le type de terminal vidéo ☐

Exercice 14 : Parmi ces attributs, lesquels font partie de l'environnement d'un processus ?

- le temps qui lui reste avant de passer dans l'état "exécuté" ☐
- son état ☐
- la liste des processus plus prioritaires que lui ☐
- son propriétaire et son groupe ☐
- les *pid* des processus qui communiquent avec lui ☐
- ses entrées/sorties standards ☐
- la partie de la mémoire où il est chargé ☐

Exercice 15 : Quel *shell* est commun à toute les machines Unix ?

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

Exercice 16 : Un *pipeline* est un enchaînement de commandes simples séparées par des `|` (par exemple : `ps aux | grep root | less`). Quel est le code de retour d'un pipeline ?

Celui de la commande la plus à gauche ☐	Celui de la commande la plus à droite ☐
Le maximum des codes de retour des commandes ☐	Le minimum des codes de retour des commandes ☐

Exercice 17 : Cochez la ou les syntaxes utilisées sous `bash` permettant de rediriger la sortie d'erreur standard d'une commande.

> ☐	2> ☐
>2 ☐	2>&1 ☐
>> ☐	stderr > ☐

Exercice 18 : Sous les systèmes du type Unix, quel est le *login* d'usage du super utilisateur ?

obiwan kenobi ☐	superuser ☐
root ☐	admin ☐

Exercice 19 : Que contient traditionnellement le répertoire `/lib` sous Unix ?

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

Soit la suite des commandes :

```
$ FOO=répertoire
$ ls -al
total 8
drwxr-xr-x 5 smith theogra 4096 2015-12-13 15:22 .
drwxr-xr-x 3 smith theogra 4096 2015-12-13 15:11 ..
drwxr-xr-x 2 smith theogra 4096 2015-12-13 15:12 dir0
drwxr-xr-x 2 smith theogra 4096 2015-12-13 15:12 dir3
-rw-r--r-- 1 smith theogra 7619 2015-12-13 15:13 dscn0001.jpg
-rw-r--r-- 1 smith theogra 3875 2015-12-13 15:13 dscn0004.jpg
-rw-r--r-- 1 smith theogra 1722 2015-12-13 15:22 dscn0007.tiff
-rw-r--r-- 1 smith theogra 9915 2015-12-13 15:13 dscn0009.jpg
$
```

Exercice 20 : Écrire les transformations successives du shell avant l'exécution de la commande :
`echo Il y a $(ls *.jpg | wc -l) jpg et les ${FOO}s dir[0-9]`

Exercice 21 : Quel est le résultat produit par l'exécution de la commande précédente ?

Exercice 22 : Écrire une commande permettant de modifier les droits d'accès de
`-rw-r-x-w- 1 bob etu 0 mai 30 15:57 foo` *en* `--w-rw-r-x 1 bob etu 0 mai 30 15:57 foo`

Exercice 23 : En supposant que le chemin suivant est valide `~/tp/../../../bin/../.`**, dites s'il s'agit d'un**
chemin absolu ..... ☐ chemin relatif ..... ☐

Exercice 24 : Pour vous débarrasser proprement d'un processus récalcitrant, indiquer dans la liste ci-après la commande la plus appropriée. Oon 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 25 : La commande `kill SIGKILL` ne peut pas tuer un processus qui est dans l'état "STOP".
VRAI ..... ☐ FAUX ..... ☐

Système de fichier

À partir de son répertoire d'accueil, l'utilisateur `yoda` lance une commande décrivant son arborescence.

```
$ ls -Ral
.:
total 20
drwxrwxr-x 5 yoda jedi 4096 Mar  1 09:13 .
drwxrwxr-x 4 yoda jedi 4096 Mar  1 09:15 ..
drwxrwxrwx 2 yoda jedi 4096 Mar  1 09:14 A
drwxr-x--x 3 yoda jedi 4096 Mar  1 09:11 B
drwxrwxrwx 2 yoda jedi 4096 Mar  1 09:12 C
-rw-rw-rw- 1 yoda jedi  132 Mar  1 09:13 d.txt

./A:
total 8
drwxrwxrwx 2 yoda jedi 4096 Mar  1 09:14 .
drwxrwxr-x 5 yoda jedi 4096 Mar  1 09:13 ..
-rw-rw-rw- 1 yoda jedi   12 Mar  1 09:12 a.txt
-rw-rw-rw- 1 yoda jedi  456 Mar  1 09:14 c.txt

./B:
total 12
drwxr-x--x 3 yoda jedi 4096 Mar  1 09:11 .
drwxrwxr-x 5 yoda jedi 4096 Mar  1 09:13 ..
drwxrwxrwx 2 yoda jedi 4096 Mar  1 09:11 D

./B/D:
total 8
drwxrwxrwx 2 yoda jedi 4096 Mar  1 09:11 .
drwxr-x--x 3 yoda jedi 4096 Mar  1 09:11 ..
-rw-rw-rw- 1 yoda jedi   24 Mar  1 09:11 a.txt

./C:
total 8
drwxrwxrwx 2 yoda jedi 4096 Mar  1 09:12 .
drwxrwxr-x 5 yoda jedi 4096 Mar  1 09:13 ..
-rw-rw-rw- 1 yoda jedi  124 Mar  1 09:11 a1.txt
-rw-rw-rw- 1 yoda jedi  128 Mar  1 09:12 a2.txt
-rw------- 1 yoda jedi  346 Mar  1 09:11 a3.txt
-rw-rw-rw- 1 yoda jedi  234 Mar  1 09:12 e.txt
drwxrwxr-x 2 yoda jedi 4096 Mar  2 15:33 F

./C/F:
total 8
drwxrwxr-x 2 yoda jedi 4096 Mar  2 15:33 .
drwxrwxrwx 3 yoda jedi 4096 Mar  2 15:33 ..
-rw-rw-rw- 1 yoda jedi  234 Mar  1 09:12 b.txt
```

Exercice 26 : Donnez une représentation sous forme d'arbre de cette arborescence de répertoires et de fichiers depuis le répertoire d'accueil de `yoda`.

On souhaite réorganiser cette arborescence selon les règles suivantes :

— On place tout fichier dans le répertoire dont le nom est la première lettre en majuscule du nom du fichier. Par exemple le fichier `c.txt` doit être placé dans le répertoire `C`. Attention, on renommera un fichier s'il doit être déplacé dans un réportoire où figure déjà un fichier de même nom.
— Tous les répertoires sans fichiers seront supprimés et si besoin on ajoutera de nouveaux répertoires pour contenir certains fichiers.
— Tous les répertoires seront contenus dans le répertoire d'accueil de `yoda`, c'est-à-dire que `D` ne sera plus un sous-répertoire de `B`.
— Après toute cette réorganisation, on ajoutera un répertoire nommé `doubles` dans lequel seront copiés tous les fichiers qui auront été renommés.

Les trois exercices suivants discutent des possibilités de la mise en œuvre de cette réorganisation pour l'utilisateur `yoda`, puis un membre de son groupe, puis un utilisateur quelconque. Les trois exercices sont indépendnant, et supposent que vous travailliez à partir du système de fichiers initial (tel que décrit dans l'exercice 26).

Exercice 27 : Donnez une suite la plus courte possible de commandes permettant à l'utilisateur `yoda` de réorganiser l'arborescence. Expliquez également pourquoi et comment vous avez dû renommer un fichier.

Exercice 28 : Cochez la(les) opération(s) interdites à un membre du groupe `jedi`.

Placer correctement le fichier `b.txt` ..... ☐
Placer correctement le fichier `a2.txt` ..... ☐
Placer correctement le fichier `a3.txt` ..... ☐
Placer correctement le fichier `e.txt` ..... ☐

Exercice 29 : Dans le cas où l'utilisateur n'est pas membre du groupe `jedi`. Peut-il exécuter les commandes suivantes sans erreurs ? On suppose que son répertoire courant est le répertoire d'accueil de `yoda`.

`cp d.txt B/D/.`	☐ Oui	☐ Non
`mv d.txt B/D/.`	☐ Oui	☐ Non
`ls B/*`	☐ Oui	☐ Non

Ligne de commandes

Exercice 30 : Écrire une ligne de commande qui permet d'afficher toutes les lignes du fichier `fichier1.txt` ne finissant pas par un signe de ponctuation (point, virgule, point-virgule, deux-points, point d'interrogation, point d'exclamation).

Exercice 31 : Écrire une ligne de commande qui permet d'effacer tous les fichiers ayant comme extension "`.mp3`" présents sur le système et de taille supérieure à 300 kOctets.

Exercice 32 : Lecture de *script-shell*

Soit le script `mon-xterm.sh` suivant :

```
$ cat mon-xterm.sh
#!/bin/bash

if [ -r couleurs.txt -a -f couleurs.txt -a -s couleurs.txt ]
then
    NB_COUL=$(wc -l < couleurs.txt)
    NO_COUL=$(expr $RANDOM % $NB_COUL)
    NO_COUL=$(expr $NO_COUL + 1)
    COUL=$(cat couleurs.txt | head -$NO_COUL | tail -1)
    BG_COUL="-bg $COUL"
fi
xterm $BG_COUL &
```

Notons que `$RANDOM` est une fonction interne de Bash qui renvoie un entier pseudo-aléatoire dans

l'intervalle $[0, 32767]$. De plus notons que l'option `-bg couleur` de la commande `xterm` permet de choisir la couleur de fond de la fenêtre créée par l'exécution de cette commande.

1) Se peut-il que la variable `NB_COUL` soit égale à 0 lors d'une exécution de ce script ? Justifier.

2) Supposons que l'on exécute le script dans un répertoire contenant le fichier `couleurs.txt` suivant et que l'appel à `$RANDOM` retourne 16.

```
red
green
blue
white
yellow
orange
grey
lightblue
pink
```

Quelle sera la valeur de chacune des variables à la fin de l'exécution du script ?

3) Décrire le comportement de l'exécution du script lorsque le fichier `couleurs.txt` n'existe pas.

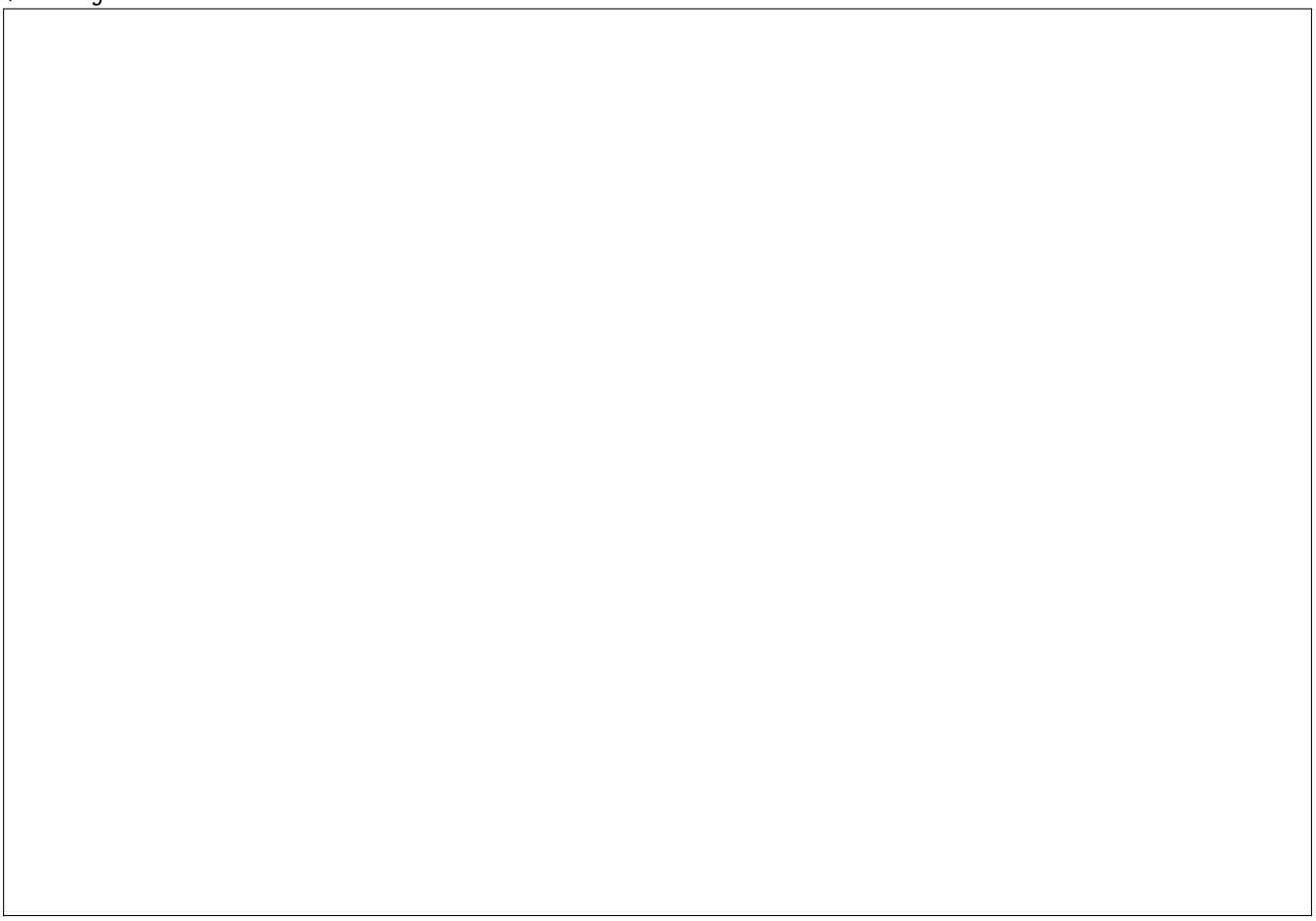
Exercice 33 : Écriture d'un *script-shell*

On peut écouter un fichier `mp3` situé dans le répertoire courant avec la commande

```
mplayer fichier.mp3
```

Écrire un script `jouer-morceau.sh` qui permet d'écouter un morceau choisi aléatoirement parmi la liste des fichiers `mp3` contenus dans un répertoire passé en paramètre au script. Ce script renverra le code d'erreur 3 si le nom du répertoire n'est pas valide ou si le répertoire est inaccessible et le code d'erreur 4 si le répertoire ne contient aucun fichier. Pour simplifier on suppose que le répertoire ne peut contenir que des fichiers `mp3`.

```
$ cat jouer-morceau.sh
```



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 **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

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

-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.

-size n[cwbkMG]
Fichier utilisant n unités d'espace. On pourra utiliser les suffixes suivants :
« b » Pour des blocs de 512 octets (comportement par défaut si rien n'est indiqué).
« c » Pour indiquer des octets.
« w » Pour des mots de deux octets.
« k » Pour des kilo-octets (unités de 1 024 octets).
« M » Pour des méga-octets (unités de 1 048 576 octets).
« G » Pour des giga-octets (unités de 1 073 741 824 octets).

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. 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 3 : 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
...

       -f FICHIER
              FICHIER existe et est de type ordinaire.

       -r FICHIER
              FICHIER existe et est accessible en lecture.

       -s FICHIER
              FICHIER existe et a une taille non nulle.

       -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 4 : 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.)

       -v, --invert-match
              Inverser la mise en correspondance, pour sélectionner les lignes
              ne correspondant pas au motif. (-v est une spécification POSIX.)
```

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

```
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 ...

        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

       -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 6 : Extrait de la page de manuel de la commande **tail**

```
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 -c
            +K pour afficher les octets de chaque fichier à partir du Kième
            octet

       -f, --follow[={name|descriptor}]
            afficher les dernières données ajoutées au fur et à mesure que
            le fichier s'accroît ; -f, --follow, et --follow=descriptor
            sont équivalents

       -F     identique à --follow=name --retry

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

       --max-unchanged-stats=N
              avec l'option --follow=name, réouvrir le FICHIER dont la taille
              n'a pas changé après N itérations (par défaut 5), afin de
              vérifier s'il a été détruit  ou  renommé (c'est habituellement
              le cas des fichiers journaux dont on effectue la rotation).
              Avec inotify, cette option est rarement utile.

       --pid=PID
              avec -f, terminer après l'arrêt du processus PID
...
```