Utilisation de la ligne de commande avec bash sous *GNU/Linux*

Version 0.3

13 février 2003

La dernière version de ce document peut être trouvée sur `http ://ppercot.free.fr`.

Historique des révisions

Version 0.3 Février 2003 Révision par : Patrick Percot (ppercot@free.fr)
Correction d’erreurs, indication des différences entre Mandrake et Debian
Première diffusion publique

Version 0.2 Janvier 2003 Révision par : Patrick Percot (ppercot@free.fr)
Modification de la mise en page

Version 0.1 Janvier 2003 Version initiale par Patrick Percot (ppercot@free.fr)
Diffusion pour une formation de l’association Tuxbihan

Table des matières

Introduction

Trop souvent, les utilisateurs habitués aux interfaces graphiques sont désemparés lorsqu'il s'agit de réaliser une opération simple en mode console (ou mode texte). Connaître l'utilisation d'un interpréteur de commandes comme `bash(1)`[1] , est pourtant essentiel. En effet, la configuration d'un système ***Unix*** ou ***GNU/Linux*** se fait par le biais de fichiers ne contenant que du texte, et savoir manipuler directement ces fichiers, ou en extraire des informations est indispensable pour la maintenance du système.

Lorsque l'interface graphique est *dépassée*, c'est-à-dire, lorsque ce que vous souhaitez faire n'a pas été programmé au préalable, que la boîte de dialogue qui vous permet d'accéder à l'information dont vous avez besoin n'existe pas encore, ou n'existera jamais, vous n'aurez plus que la ressource de réaliser les traitements en mode console. Vous vous apercevrez alors de toute la puissance qui est entre vos mains, et vous constaterez que cette puissance ne vous est accessible qu'en mode console.

Avertissement

Comme toute documentation technique, ce document peut contenir des erreurs. Bien qu'ayant essayé de tester toutes les commandes qui sont données en exemple, il est toujours possible que ces commandes ne produisent pas exactement le même résultat dans tous les environnements. Si tel était le cas, je vous serais reconnaissant de m'en informer (`ppercot@free.fr`) afin que je corrige ces erreurs dans ce document.

[1] Vous rencontrez souvent cette notation, elle correspond au nom de la commande et au numéro de section de la page de manuel (cf. Section 1.3.1)

Chapitre 1

Les débuts

1.1 Passer en mode console

L'ensemble des opérations qui seront décrites ultérieurement, et l'ensemble des exercices nécessitent de disposer d'une console.

Pour passer en mode console, plusieurs options se présentent à vous :

Terminal X
Cliquez sur une des icônes permettent de lancer un terminal X

Console virtuelle
Utilisez (avec Gnome sous Debian) la séquence Ctl-Alt-Fn, avec $1 <= n <= 6$, [1] pour passer sur une console virtuelle, et authentifiez-vous.

Shell `Emacs(1)`
Sous X ?Emacs[2], lancez la commande "`M-x shell`". A noter que vous bénéficiez des facilités d'édition d'X ?Emacs, et que vous pourrez sauvegarder tous vos exercices dans un fichier, mais que vous ne pourrez pas utiliser certaines commandes comme `man(1)` ou `less(1)` qui nécessitent un terminal.

1.2 Format des commandes

L'ensemble des commandes Unix et ***GNU/Linux*** répondent à quelques règles structurelles simples. Les commandes prennent la forme suivante :
`commande [options] [paramètres]` , où :

`commande`
est le nom de la commande.

`options`
est une liste d'options commençant par `-` ou `--`. Cette liste est optionnelle.

[1] d'autres consoles virtuelles peuvent exister : $n >= 9$, la touche Fn correspond aux touches de fonction en haut du clavier

[2] Ceci est une expression régulière (ou rationnelle), une regex quoi, ou regexp si vous préférez. Elle représente toute chaîne de caractères contenant Emacs, éventuellement précédée d'un X. J'aurais également pu écrire [XJ] ?Emacs pour satisfaire les utilisateurs de JEmacs, cette dernière expression représentant toute chaîne de caractères contenant Emacs, et éventuellement précédée d'un X ou d'un J. Si vous vous intéressez aux expressions régulières, je vous conseille le livre de Jeffrey Friedl "Maîtrise des expression régulières".

`paramètres`
: est une liste de paramètres. Cette liste est optionnelle.

1.3 Obtenir de l'aide

1.3.1 La commande `man(1)`

Une des premières choses à connaître avant de se lancer dans l'exploration du monde des commandes Unix est la commande `man(1)`. Elle permet d'obtenir de l'aide sur la commande passée en paramètre.

Elle utilise par défaut la commande `pager(1)` pour afficher le texte page par page, et offrir certaines fonctionnalités de recherche et de déplacement. Voici les plus importantes :

Quitter
: Pour quitter, tapez "`q`"

Déplacements
: En général les touches *Home*, *End*, et les autres touches de déplacement fonctionnent. Sinon, essayez les séquences : "`j`", "`k`", "`ESPACE`", "`b`", "`ESC <`", "`ESC >`".

Recherche
: "`/texte`" permet de chercher "`texte`" en avant. "`?texte`" permet de chercher "`texte`" en arrière. "`n`" permet de continuer la recherche vers l'avant. "`N`" permet de continuer la recherche vers l'arrière.

Pour se familiariser avec cette commande, pourquoi ne pas lui demander comment elle fonctionne ?

```
$ man man
```

Vous apprendrez notamment que le manuel est divisé en sections.

1 : Programmes exécutables ou commandes shell

2 : Appels systèmes (fonctions fournies par le noyau)

3 : Appels des bibliothèques (Fonctions des bibliothèques système)

4 : Fichiers spéciaux (habituellement situés dans /dev)

5 : Formats de fichiers et conventions par ex. /etc/passwd

6 : Jeux

7 : Paquetages de macros et conventions par ex. man(7), groff(7).

8 : Commandes d'administration système (habituellement réservées à root)

9 : Routines du noyau [Non standard]

Pour les noms possédant des entrées multiples ; essayez par exemple :

```
$ man -f printf
```

`man -f` est équivalent à la commande `whatis(1)`.

Vous pouvez dans ce cas être obligé d'indiquer la section :

```
$ man 3 printf
```

parce qu'il existe une page de manuel de même nom dans une des sections précédentes.

Pour rechercher les pages de manuel qui mentionnent une information, utilisez :

```
$ man -k printf
```

qui est un équivalent de la commande `apropos(1)`.

Pour rechercher tous les fichiers relatifs à une commande, utilisez `whereis(1)` :

```
$ whereis printf
```

1.3.2 La commande `info(1)`

Le système ***GNU/Linux*** dispose en plus d'une commande nommée `info(1)`. Elle permet de parcourir l'aide relative à une commande. Elle utilise un système de commande proche de X ?Emacs, avec notamment le complétement [3] de texte accessible avec la touche "`TAB`". Les pages d'aide sont structurées en une arborescence de nœuds que l'on peut parcourir avec "`m`" pour atteindre un menu, "`n`" pour aller au nœud suivant, et "`p`" pour aller au nœud précédent.

Essayons :

```
$ info info
```

Pour se déplacer, les touches de déplacements devraient fonctionner. Sinon, essayez les séquences "`ESPACE`", "`BACKSPACE`". La touche "`TAB`" permet de passer de menu en menu, la touche "`RET`" permet d'entrer dans le menu. La touche "`l`" permet de revenir au nœud que l'on vient de quitter.

Les commandes "`C-s`" et "`C-r`" [4] permettent respectivement de faire une recherche incrémentielle en avant et en arrière. N'hésitez pas à lire les documentations sur ces commandes, pour ce faire, utilisez "`h`" pour étudier le cours d'instruction[5], et " ?" pour une aide rapide.

NOTE : la commande `info` affiche aussi les pages de manuel, et remplace donc complètement la commande `man` .

1.3.3 La commande `locate(1)`

Cette commande permet de savoir où sont localisés les fichiers correspondant à un modèle[6] passé en paramètre. Ce modèle peut contenir des méta-caractères (voir la section 2.1.4), dans ce cas il faut mettre des apostrophes doubles autour du modèle.

Le résultat est la liste des fichiers qui correspondent.

La base de données (`locatedb(8)`) est mise à jour par la commande `updatedb(1)`, lancée périodiquement par `cron(8)`. Vous pouvez lancer `updatedb(1)` manuellement, mais il faut être `root` pour ce faire (voir la section 1.5).

1.4 Utiliser un éditeur

De nombreux éditeurs sont disponibles sur les systèmes ***GNU/Linux*** et ***Unix*** , mais nous aborderons les deux principaux, à savoir `Emacs(1)` et `vi(1)`.

1.4.1 Emacs

`emacs(1)`Emacs est plus qu'un éditeur, car il est entièrement programmable avec le langage `Emacs Lisp` . Mais nous n'aborderons que l'aspect éditeur. Il est particulièrement puissant, mais aussi très gourmand en mémoire et en CPU.

Pour invoquer Emacs, il suffit de lancer la commande `emacs` suivie éventuellement du nom des fichiers que l'on souhaite éditer.

Les séquences de touches Emacs sont préfixées par des caractères de contrôle ("`C-`" signifie Control, "`M-`" signifie Meta ou Alt, "`S-`" signifie Shift. Il est parfois nécessaire

[3] "Completion" en anglais.

[4] selon la terminologie X ?Emacs, 'S-' signifie Shift, 'C-' signifie Control, 'M-' signifie Meta (ou Alt).

[5] "Tutorial" en anglais.

[6] "Pattern" en anglais.

d'utiliser la touche "ESC" au lieu de la touche "Alt", dans ce cas, la touche "ESC" précède le caractère de la commande. Par exemple, "M-w" peut être obtenu en appuyant simultanément sur "Alt" et "w", mais aussi en appuyant *successivement* "ESC" et "w".

Les tableaux "Les commandes indispensables d'Emacs" (1.1) et "Les commandes d'édition d'Emacs" (1.2) reprennent les séquences les plus importantes.

TAB. 1.1 – Les commandes indispensables d'Emacs

C-x C-c	Quitter Emacs
C-x C-f	Ouvrir un fichier
C-x C-s	Sauvegarder les modifications du buffer courant
C-x s	Sauvegarder tous les buffers modifiés
C-h t	Démarrer la formation assistée
C-h i	Démarrer l'aide (info)
C-h k	Décrire la séquence de touches
C-h f	Obtenir de l'aide sur la fonction
C-h a	Obtenir de l'aide sur les fonctions correspondant à la regexp
C-h v	Obtenir de l'aide sur la variable

TAB. 1.2 – Les commandes d'édition d'Emacs

C-_ ou C-)	Annuler la modification
C-x b	Changer de buffer
C-x C-b	Ouvrir la fenêtre de sélection de buffer
C-x 2	Diviser horizontalement l'écran en deux fenêtres
C-x 3	Diviser verticalement l'écran en deux fenêtres
C-x 0	Garder cette fenêtre (ferme la dernière fenêtre ouverte
C-x o	Changer de fenêtre
C-SPACE	Poser le marqueur
M-w	Copier le texte entre le point et le marqueur
C-w	Couper le texte entre le point et le marqueur
M-d	Couper le mot à droite du point
M-BACKSPACE	Couper le mot à gauche du point
C-k	Couper la ligne à droite du point
C-y	Coller le texte précédemment copié ou coupé
M-y	Coller le texte suivant du kill ring[7]
C-t	Transposer le caractère au point courant et le caractère précédent
M-t	Transposer le mot après le point et celui qui précède
C-x C-t	Transposer la ligne où est le point et la précédente
C-s	Recherche incrémentielle en avant
C-r	Recherche incrémentielle en arrière
M-%	Remplacer du texte

De nombreuses autres fonctionnalités sont disponibles, n'hésitez pas à consulter toutes les aides disponibles 1.1.

1.4.2 vi

`vi(1)` ou `vim` ou `gvim` est un éditeur beaucoup plus léger qu'Emacs. Il n'a cependant pas grand chose à envier à Emacs pour ce qui concerne la puissance[8].

L'utilisation de `vi` est assez inhabituelle, car il utilise un mode commande dans lequel les caractères frappés exécutent des commandes, et un mode saisie où les caractères sont insérés dans le buffer.

`vi` est invoqué en lançant la commande `vi` suivi éventuellement du nom des fichiers à éditer.

`vimtutor` est le cours d'instructions pour vi.

Les fonctionnalités les plus importantes sont regroupées dans ce tableau (1.3).

TAB. 1.3 – Les principales commandes de vi

`:help`	Obtenir de l'aide
`:q`	Quitter vi (le fichier courant)
`:q`	Quitter vi (ferme tous les fichiers)
`:q!`	Quitter vi sans confirmation (abandonne les modifications)
`:qa!`	Quitter vi sans confirmation (abandonne les modifications de tous les fichiers)
`:w`	Enregistrer les modifications
`ESC`	Quitter le mode insertion
`i`	Passer en mode insertion à ce point
`o`	Passer en mode insertion après la ligne courante
`O`	Passer en mode insertion avant la ligne courante
`u`	Annule la modification
`Y`	Copie la ligne courante
`p`	Colle la ligne copiée après la ligne courante
`P`	Colle la ligne copiée avant la ligne courante
`x`	Supprime le caractère à droite
`X`	Supprime le caractère à gauche
`dw`	Détruit le mot à droite
`cw`	Change le mot à droite
`dd`	Supprime la ligne
`/`	Recherche le texte en avant
`?`	Recherche le texte en arrière
`n`	Poursuit la recherche en avant
`N`	Poursuit la recherche en arrière
`:vi`	Ouvrir un fichier à la place du fichier courant
`:next`	Passer au fichier suivant
`:prev`	Passer au fichier précédent

[8] Bien que nous n'abordions pas ici, que ce soit pour Emacs ou vi, les fonctionnalités qui font de ces deux éditeurs des outils hors du commun (notamment, l'utilisation des expressions régulières).

1.5 Devenir super utilisateur

La commande `su(1)` permet de changer d'identité. Sa syntaxe est : `su utilisateur` . Si l'on veut changer son identité pour celle de `root` , le paramètre `utilisateur` est optionnel.

```
$ su
Password:
# exit
$ su tuxbihan
Password:
$ exit
$
```

Pour reprendre son ancienne identité, il suffit d'utiliser la commande `exit(1)` ou te taper la séquence "`Ctrl D`".

Chapitre 2

L'interpréteur de commandes

2.1 La commande `echo(1)`

2.1.1 Affichage de texte brut

La commande `echo(1)` affiche les paramètres qui lui sont passés, sans modification.

```
$ echo Hello World
Hello World
$
```

Nous ne nous attarderons pas sur les options de cette commande, elles sont simples, consultez la page de manuel.

2.1.2 Affichage de variables d'environnement

Essayez ceci :

```
$ echo $HOME
/home/patrick
```

Alors que je viens de vous dire que `echo(1)` affiche les paramètres qui lui sont passés sans modification, cet exemple semble montrer exactement le contraire. Toutefois, je ne vous ai pas menti. Que se passe t'il donc réellement ?

La commande echo reçoit en fait le texte `/home/patrick` , parce que l'interpréteur de commandes `bash(1)` a réalisé une opération nommée "**interpolation de variables**" avant de procéder à l'exécution de la commande spécifiée. Une variable est un nom préfixé par le symbole `$` . Une seconde syntaxe est utilisable :

```
$ echo ${HOME}_SWEET_HOME
/home/patrick_SWEET_HOME
```

Elle permet de spécifier les caractères qui doivent être considérés comme faisant partie du nom de la variable, dans le cas où l'interpréteur de commande ne pourrait pas reconnaître la fin du nom de la variable. Considérez ce qui se passe si l'on n'utilise pas les accolades :

```
$
$ echo $HOME_SWEET_HOME

$
```

La commande `echo(1)` n'affiche rien, car la variable `HOME_SWEET_HOME` n'est pas définie.

Mais si l'on crée cette variable :

```
$ HOME_SWEET_HOME=Breizh
$ echo $HOME_SWEET_HOME
Breizh
$
```

2.1.3 Bloquer l'interpolation

Et comment faire pour afficher le symbole `$` ? Plusieurs méthodes :

```
$ echo \$HOME
$HOME
$ echo '$HOME'
$HOME
$
```

Dans le premier exemple, on utilise le caractère d'échappement[1] pour interdire l'interprétation d'un seul caractère. Dans le second, on masque l'ensemble de la chaîne à l'aide des quotes simples, le texte est donc passé sans modification à la commande `echo(1)`.

L'utilisation des quotes doubles (`""`) n'empêche pas l'interpolation des variables, mais empêche l'interprétation des espaces et tabulations comme séparateurs de paramètres. On peut ainsi passer un paramètre qui sans les doubles quotes serait considéré comme plusieurs paramètres distincts.

```
$ echo abc      def
abc def
$ echo "abc      def"
abc      def
```

2.1.4 Listage des fichiers

Maintenant, essayons ceci :

```
$ echo *
$ first-step.aux first-step.dvi first-step.tex first-step.toc
```

[1] Pas très heureux ce nom, vous direz-vous ! Mais saviez-vous que le mécanisme d'horlogerie qui permet à un ressort de se détendre à vitesse constante est aussi appelé échappement, et qu'il a été inventé durant la dernière partie du Moyen-Âge ? Ce fut une période d'intense activité technologique à la fin de laquelle il restait moins de forêts en France qu'il n'y en a de nos jours. L'Église, plus puissante qu'elle ne l'est aujourd'hui, y a mis un terme pour quelques siècles.

Cette fois, rien à voir avec l'interpolation de variables, c'est à la "**globalisation**", appelée aussi "**expansion**" que nous avons affaire. La globalisation permet de définir un ensemble de noms de fichiers par définition, à charge pour l'interpréteur de commande de nous afficher cet ensemble de fichiers par extension. La globalisation fait appel à deux caractères joker `*` et `?` , ainsi qu'à une classe de caractères, par exemple `[12345]` est équivalent à `[1-5]` .

`*`
: correspond à n'importe quelle suite de caractères, y compris la chaîne vide.

`?`
: correspond à un et un seul caractère, quel qu'il soit.

`[xyz]`
: correspond à l'un des caractères cités à l'intérieur des crochets.

`[x-z]`
: correspond à l'un des caractères compris dans l'intervalle.

`[!xyz]`
: correspond si aucun des caractères cités à l'intérieur des crochets n'est trouvé (le caractère `^` peut être utilisé à la place de `!`).

`[:class:]`
: où `class` est une des classes définies dans le standard POSIX.2 (`alnum alpha ascii blank cntrl digit graph lower print punct space upper xdigit`)

Quelques petits exemples :

```
$ echo *
abc1.c abc1.cpp abc1.h abc1.hxx ABC1.c ABC1.cpp ABC1.h ABC1.hxx
$ echo *h
abc1.h ABC1.h
$ echo *.?
abc1.c abc1.h ABC1.c ABC1.h
$ echo [a-z]*
abc1.c ABC1.c abc1.cpp ABC1.cpp abc1.h ABC1.h abc1.hxx ABC1.hxx
$ echo [A-Z]*
ABC1.c ABC1.cpp ABC1.h ABC1.hxx
$ echo [[:lower:]]*
abc1.c abc1.cpp abc1.h abc1.hxx
$ echo [[:lower:][:upper:]]*
abc1.c ABC1.c abc1.cpp ABC1.cpp abc1.h ABC1.h abc1.hxx ABC1.hxx
$ echo *[px][px]
abc1.cpp abc1.hxx ABC1.cpp ABC1.hxx
```

Vous constaterez une *anomalie* en ligne 4 (`echo [a-z]*`) ne retourne pas le résultat attendu. Si cela ne s'est pas produit sur votre machine, c'est parce qu'une variable de `bash(1)` n'est pas positionnée, il s'agit de `nocaseglob(bash(1))`. Elle indique que l'on ne tient pas compte de la casse. Si l'on veut tenir compte de la casse, et ne reconnaître que des caractères minuscules, il faut utiliser les classes prédéfinies (ne pas oublier les crochets autour du symbole représentant la classe), ou modifier cette variable à l'aide de la commande interne `shopt(bash(1))`.

```
$ shopt -u nocaseglob
$ shopt
[...]
nocaseglob off
[...]
$ shopt -s nocaseglob
$ shopt
[...]
nocaseglob on
[...]
```

Consultez l'aide de `bash(1)` et recherchez `extglob(bash(1))` si vous souhaitez utiliser des formats de description plus complexes. Vous devrez utiliser `shopt(bash(1))` pour positionner `extglob(bash(1))` afin d'utiliser ces formats. Consultez aussi `glob(7)`, prenez garde cette fois à taper `man 7 glob` sinon vous aurez la documentation de la fonction `glob(3)`.

2.2 Les variables d'environnement

Dans la section précédente, nous avons eu un bref aperçu des variables d'environnement. Nous allons approfondir cette notion. Les variables jouent un rôle de conteneur. Elles sont identifiées par une chaîne de caractères, et leur contenu peut être retrouvé (la variable est alors déréférencée) en préfixant son nom (son identifiant) par le symbole $.

2.2.1 Exportation de variables

Les variables sont héritées par les shells fils du shell courant à condition qu'elles soient exportées. Faisons une expérience :

```
$ echo $SHLVL
1
$ bash
$ echo $SHLVL
2
$ exit
$ echo $SHLVL
1
```

La variable SHLVL indique le niveau du shell. Le niveau 1 indique que le shell est le fils direct du processus `init` . A chaque nouveau shell lancé, le niveau est incrémenté de 1.

Poursuivons l'expérience avec les variables :

```
$ HELLO=hello
$ echo $HELLO
hello
$ bash
$ echo $HELLO
```

```
$ exit
$ echo $HELLO
hello
$
```

La variable `HELLO` créée dans le premier shell n'existe pas dans le shell fils, car la variable n'a pas été exportée, ce qui se fait avec la commande interne `export`*(bash(1))*.

Recommençons :

```
$ export HELLO=hello
$ echo $HELLO
hello
$ bash
$ echo $HELLO
hello
$ exit
$ echo $HELLO
hello
$
```

Cette fois, la variable existe dans les deux shells.

2.2.2 Les commandes `set` et `env`

Les commandes `set`*(bash(1))* qui est une commande interne de `bash`*(1)* et `env`*(1)* permettent de connaître l'ensemble des variables qui ont été définies. La commande `set`*(bash(1))* affiche en plus toutes les variables internes de `bash`*(1)*. À noter que les variables internes optionnelles sont affichées par `shopt`*(bash(1))*.

La fonction principale de `env` est de lancer un programme dans un environnement modifié. Elle permet de supprimer certaines variables, et d'en définir d'autres au sein de l'environnement dans lequel sera lancé le programme passé en argument.

Une des utilisations possibles est de lancer un programme sensible à la localisation dans différentes langues sans modifier l'environnement du shell courant :

```
$ env LANG=en gcompris
$ env LANG=fr gcompris
```

Les commandes précédentes lancent d'abord gcompris en anglais, puis en français, ce que vous ne manquerez pas d'entendre si vous avez une carte son.

2.3 Évaluation intermédiaire d'une commande

Les quotes inverses (ou backticks) permettent d'évaluer (au sens fonctionnel du terme, c'est-à-dire exécution et récupération du résultat) une commande et d'utiliser le résultat de l'évaluation.

```
$ echo $SHLVL
1
$ echo /bin/bash > my-preferred-shell
$ ‘cat my-preferred-shell‘
```

```
$ echo $SHLVL
2
$ exit
$ echo $SHLVL
1
$ ls -l ‘cat my-preferred-shell‘
-rwxr-xr-x 1 root root 511400 May 30 2002 /bin/bash
```

Dans le premier cas, `cat my-preferred-shell` est tout d’abord évalué, puis, comme les backticks sont au début de la ligne de commande, son résultat est utilisé comme nom de commande à exécuter.

Dans le second cas, `cat my-preferred-shell` est tout d’abord évalué, puis, comme les backticks ne sont pas au début de la ligne de commande, son résultat est utilisé comme paramètre de la commande `ls` .

Le texte contenu entre les backticks est évalué, puis son résultat est incorporé dans la ligne de commande à l’emplacement même où se trouvaient les backticks. La commande ainsi formée est exécutée.

Chapitre 3

Manipulation de fichiers

Dans ce chapitre, nous allons étudier la structure des répertoires, la création de fichiers, de liens physiques et dynamiques, ainsi que la redirection des entrées et des sorties, les canaux de communications (pipes) et le système des permissions.

3.1 La structure des répertoires

Le système de fichiers des systèmes ***Unix*** et de ***GNU/Linux*** est homogène, bien que les supports physiques puissent ne pas être identiques. En effet, les fichiers stockés sur une disquette, un lecteur de CDROM, un disque SCSI ou un disque IDE vont être accessibles dans la même arborescence de répertoires de façon indépendante du support.

3.1.1 Description du système de fichiers

Sur Debian, vous pouvez trouver une source d'informations sur le FHS dans `/usr/share/doc/debian-policy/fh`
.

La racine de l'arborescence est symbolisée par "`/`". Les répertoires suivants font partie de l'arborescence définie par le "Filesystem Hierarchy Standard" (FHS version 2.1).

`/` Répertoire racine

`/bin` Principales commandes utilisateurs binaires (pour tous les utilisateurs)

`/boot` Fichiers du chargeur de système (boot loader)

`/dev` Fichiers de périphériques

`/etc` Fichiers de configuration système de la machine

`/etc/X11` Configuration du système X Window

`/etc/opt` Fichiers de configuration pour /opt

`/home` Répertoires de base des utilisateurs (home) (optionnel)

`/lib` Bibliothèques partagées essentielles et modules du noyau

`/mnt` Point de montage des systèmes de fichiers montés temporairement

`/opt` Paquetages des logiciels applicatifs ajoutés au système

`/root` Répertoire de base de l'utilisateur root (optionnel)

`/sbin` Binaires systèmes (autrefois dans /etc)
`/tmp` Fichiers temporaires
`/usr/X11R6` Système X Window, Version 11 Release 6
`/usr/bin` Principalement commandes utilisateurs
`/usr/include` Répertoire pour les fichiers “include” standards
`/usr/lib` Bibliothèques pour la programmation et paquetages
`/usr/local` Hiérarchie locale au système
`/usr/sbin` Binaires systèmes standard non essentiels
`/usr/share` Données indépendantes de l’architecture du système
`/usr/share/dict` Listes de mots
`/usr/share/man` Pages de manuel
`/usr/share/misc` Diverses données indépendantes de l’architecture du système
`/usr/src` Code source
`/var/account` Registres (logs) de comptabilité des processus (si supporté)
`/var/cache` Données de cache des applications
`/var/cache/fonts` Polices générées localement
`/var/cache/man` Pages de manuel formatées localement (optionnel)
`/var/crash` Vidages des crash systèmes (si supporté)
`/var/games` Données variables des jeux
`/var/lib` Informations d’état
`/var/lib/<editor>` Fichiers de “backup” et d’état
`/var/lib/misc` Données diverses
`/var/lock` Fichiers de verrous
`/var/log` Fichiers et répertoires registres (logs)
`/var/mail` Fichiers de boîte à lettre des utilisateurs
`/var/opt` Données variables pour /opt
`/var/run` Données variables des applications en cours d’exécution
`/var/spool` Données de spool d’application
`/var/spool/lpd` Files d’attente du daemon d’impression
`/var/spool/rwho` Fichiers rwhod
`/var/tmp` Fichiers temporaires préservés entre les redémarrages
`/var/yp` Fichiers de la base de données NIS (Network Information Service)

Spécificités de Linux :

`/dev` Fichiers de périphériques et fichiers spéciaux
`/proc` Système de fichier virtuel des informations du noyau
`/sbin` Binaires systèmes essentiels
`/usr/include` Fichiers d’entête inclus par les programmes C
`/usr/src` Code source
`/var/spool/cron` Données de spool de `cron`*`(8)`* et `at`*`(1)`*

3.2 Création de répertoires et déplacement dans l'arborescence

Après ce tour d'horizon du système de fichiers, commençons par nous localiser. Normalement, nous devrions nous trouver dans notre répertoire de base. Vérifions-le à l'aide de `pwd(1)` :

```
$ pwd
/home/patrick
$
```

Si nous ne nous trouvons pas dans notre répertoire de base, nous pouvons nous déplacer avec la commande `cd(bash(1))`.

```
$ cd /tmp
$ pwd
/tmp
$ cd
$ pwd
/home/patrick
```

`cd` sans argument nous replace dans notre répertoire de base.

Nous allons maintenant créer un répertoire dans lequel nous pourrons créer des fichiers pour les exercices (`mkdir(1)`).

```
$ mkdir first-step
$ cd first-step
$ pwd
/home/patrick/first-step
$ mkdir second-step
$ cd second-step
$ cd -
$ pwd
/home/patrick/first-step
$ cd second-step
$ pwd
/home/patrick/first-step/second-step
$ cd ~
$ pwd
/home/patrick
$
```

`cd -` nous ramène dans le répertoire que nous venons de quitter.

`cd ~` nous ramène dans notre répertoire de base, comme le fait `cd` et comme le ferait `cd $HOME` . Le caractère "~" peut être suivi d'un nom de répertoire. Il permet alors de spécifier de façon absolue le répertoire dans lequel on désire se placer, par exemple `cd /first-step` , par opposition au déplacement relatif au répertoire courant, comme dans `cd first-step` , qui suppose qu'un répertoire `first-step` existe dans le répertoire courant.

La commande `rmdir(1)` permet de supprimer un répertoire. Si le répertoire n'est pas vide, la commande retourne une erreur, et le répertoire n'est pas détruit.

```
$ mkdir essai
$ cd essai
$ pwd
/home/patrick/first-step/essai
$ cd ..
$ pwd
/home/patrick/first-step
$ rmdir essai
$ cd essai
bash: cd: essai: No such file or directory
$
```

Nous venons de voir une autre nouveauté (`cd ..`). "`..`" représente le répertoire parent du répertoire courant. `cd ..` permet donc de redescendre (ou de remonter si l'on considère que la racine est en haut) d'un niveau dans l'arborescence.

3.3 La redirection des entrées et des sorties

Lorsqu'un programme est démarré, le système ouvre pour lui, automatiquement, trois fichiers. Ces fichiers sont nommés `stdin` , `stdout` et `stderr` , ou `0` , `1` ou `2` , selon que l'on utilise la notation des fonctions de la bibliothèque ' `libc` ' (`fopen`*`(3)`*, `fwrite`*`(3)`*, `fread`*`(3)`* et `fclose`*`(3)`*), ou les fonctions systèmes (`open`*`(2)`*, `write`*`(2)`*, `read`*`(2)`* et `close`*`(2)`*).

Ces fichiers sont normalement le fichier d'entrée (le clavier), le fichier de sortie (l'écran) et le fichier d'erreur (également l'écran). La redirection des entrées et des sorties concerne ces trois fichiers, et consiste à substituer un autre fichier aux fichiers standard.

Nous sommes dans le répertoire de travail. Nous allons maintenant créer des fichiers.

3.3.1 Redirection de la sortie simple

```
$ ls
$ > first-file
$ ls
first-file
$ echo coucou > second-file
$ ls
first-file second-file
$
```

La commande `ls`*`(1)`* liste le contenu du répertoire qui lui est passé en paramètre, ou du répertoire courant si aucun paramètre ne lui a été transmis. Nous voyons que pour l'instant, cette commande a le même effet que `echo *` , mais elle peut faire beaucoup plus.

Nous venons aussi de voir nos premières redirections. En voici la description du fonctionnement. Le caractère "`>`" sert à rediriger la sortie (ce qui est affiché à l'écran) de la commande vers un fichier.

Dans le second cas (`echo coucou > second-file`), la commande `echo` devrait afficher `coucou` à l'écran, mais la redirection provoque l'envoi du texte dans le fichier `second-file` . Le fichier est créé s'il n'existe pas. Il est détruit, puis recréé s'il existait.

Dans le premier cas, l'usage est assez particulier, puisqu'il n'y a pas de commande. De ce fait, rien n'est envoyé dans le fichier, mais il est tout de même créé. Nous aurions pu parvenir au même résultat avec la commande `touch(1)`, dont l'usage est normalement de changer la date de modification, mais qui a comme effet de bord de créer le fichier s'il n'existe pas.

3.3.2 Redirection de la sortie en ajout

```
$ ls -a -1
.
..
first-file
second-file
$ ls -a -1 >> second-file
$ cat second-file
coucou
.
..
first-file
second-file
$
```

La première commande liste les fichiers du répertoire courant, un fichier par ligne. La seconde fait de même, mais la sortie au lieu d'être envoyée sur l'écran est redirigée en ajout "`>>`" dans le fichier `second-file` . La commande `cat(1)` affiche ensuite le contenu du fichier.

3.3.3 Redirection de l'entrée

Pour faire un essai, nous allons utiliser la commande `grep(1)`.

```
$ grep o < second-file
coucou
second-file
$
```

Dans ce contexte d'utilisation, elle recherche dans son entrée standard la chaîne qui lui a été passée en paramètre (dans notre cas "`o`"), et affiche chaque ligne pour laquelle elle a trouvé une correspondance (dans notre cas toutes les lignes contenant au moins un "`o`"). L'interpréteur de commande ouvre le fichier `second-file` et son contenu

Mais nous aurions aussi pu utiliser la commande `grep` sans la redirection. Tout fonctionne correctement, car la commande `grep` traite les paramètres supplémentaires comme des fichiers.

```
$ grep o second-file
coucou
second-file
$
```

3.3.4 Redirection de la sortie d'erreur vers la sortie standard

Pour faire nos essais, nous allons utiliser ce court programme en C (voir figure 3.1), que nous enregistrerons dans un fichier nommé `trace.c` , et que nous allons compiler avec la ligne de commande suivante `cc -o trace trace.c` .

```
#include <stdio.h>
int
main()
{
  fprintf(stdout,"sortie vers stdout\n");
  fprintf(stderr,"erreur vers stderr\n");
}
```

FIG. 3.1 – Code source de trace.c

Nous allons maintenant utiliser le programme `trace` que nous avons généré. Il écrit le texte `sortie vers stdout` dans le fichier `stdout` , et le texte `erreur vers stderr` dans le fichier `stderr` . Ceci va nous permettre de différencier les effets des redirections que nous allons écrire.

```
$ ./trace
sortie vers stdout
erreur vers stderr
$ ./trace > sortie
erreur vers stderr
$ cat sortie
sortie vers stdout
$ ./trace 1> sortie
erreur vers stderr
$ cat sortie
sortie vers stdout
$ ./trace 2> erreur
sortie vers stdout
$ cat erreur
erreur vers stderr
$ ./trace > sortie 2>&1
$ cat sortie
sortie vers stdout
erreur vers stderr
$ ./trace &>sortie
sortie vers stdout
erreur vers stderr
```

Nous constatons deux choses :

1. que nous écrivions "`1>`"[1] ou "`>`", cela ne fait pas de différence. Si le numéro du descripteur de fichier est omis, c'est `stdout` qui est utilisé.
2. les deux syntaxes "`> sortie 2>&1`" et "`&>sortie`" sont équivalentes.

[1] Attention : le "`1`" doit être accolé au "`>`", sinon le shell va interpréter "`1`" comme un paramètre du programme.

Les syntaxes que nous venons de voir sont très utiles dans les cas où il est nécessaire de dissocier les sorties normales d'un programme et les sorties d'erreur, ainsi que dans les cas où il faut récupérer toutes ces sorties dans un même fichier. Nous verrons (Section 3.4) comment conserver l'affichage à l'écran tout en envoyant la sortie vers un fichier.

3.4 Les canaux de communications

Un canal de communication | est un moyen de connecter la sortie d'un programme sur l'entrée d'un autre. Le programme `wc(1)` compte les lignes, les mots et les caractères qu'il lit sur son entrée standard, et affiche ces valeurs sur une ligne lorsqu'il se termine. L'option `-l` lui demande de ne tenir compte que des lignes.

```
$ ./trace | wc -l
erreur vers stderr
      1
$ ./trace 2>&1 | wc -l
      2
```

Dans le premier cas, seule la sortie standard de `trace` est connectée à l'entrée standard de `wc(1)`, alors que dans le second les deux sorties de `trace` sont connectées à l'entrée standard `wc`.

Comme promis, un programme très utile : `tee(1)`. Il recopie son entrée standard dans le fichier passé en paramètre ainsi que sur sa sortie standard.

```
$ ./trace 2>&1 | tee sorties-de-trace
erreur vers stderr
sortie vers stdout
$ cat sorties-de-trace
erreur vers stderr
sortie vers stdout
```

3.5 Création de liens

Deux types de liens existent dans les systèmes de fichiers ***GNU/Linux*** et ***Unix***. Les deux types de liens peuvent être créés à l'aide de la commande `ln(1)`.

Liens physiques Le fichier créé ainsi partage le même espace disque physique que le fichier originel. Les deux fichiers doivent être sur une même partition physique.

Liens symboliques Le fichier créé ainsi indique l'emplacement du fichier désigné. L'emplacement du fichier destination peut être relatif ou absolu, et peut résider sur une autre partition.

```
$ ls -l sortie
-rw-r-----    1 patrick  patrick        38 Dec 27 02:58 sortie
$ ln sortie output
$ ls -l sortie output
-rw-r-----    2 patrick  patrick        38 Dec 27 02:58 output
-rw-r-----    2 patrick  patrick        38 Dec 27 02:58 sortie
$ ln -s sortie sym
```

```
$ ls -l sortie sym
-rw-r-----    2 patrick  patrick        38 Dec 27 02:58 sortie
lrwxrwxrwx    1 patrick  patrick         6 Dec 27 04:11 sym -> sortie
$
```

La valeur dans la seconde colonne donne le nombre de liens physiques sur l'inode du fichier.

La création du premier lien incrémente ce nombre de liens. Une des conséquences est que le fichier ne pourra être détruit physiquement que lorsque le nombre de ces liens deviendra nul. Une autre conséquence est que l'on peut supprimer le fichier originel sans que le nouveau fichier n'en soit altéré.

La création du second type de lien ne modifie pas cette valeur. Le fichier nouvellement créé ne fait qu'indiquer l'emplacement du fichier originel. Si l'on supprime le fichier originel, le lien continue d'exister mais n'est plus fonctionnel.

Pour supprimer un lien, on utilise la commande `rm(1)` qui supprime aussi bien des liens que des fichiers.

```
$ rm sortie
$ ls -l output sym
-rw-r-----    1 patrick  patrick        38 Dec 27 02:58 output
lrwxrwxrwx    1 patrick  patrick         6 Dec 27 04:11 sym -> sortie
$ cat output
erreur vers stderr
sortie vers stdout
$ cat sym
cat: sym: No such file or directory
```

3.6 Suppression de liens symboliques

Lorsque l'on supprime un lien symbolique, cela ne touche pas le fichier ou répertoire désigné par le lien. Ainsi, lors d'une destruction récursive de répertoire (`rm -r repertoire`), les liens symboliques contenus à l'intérieur de ce répertoire ou des répertoires détruis récursivement sont simplement supprimés. Si un lien symbolique désigne un répertoire, le répertoire désigné n'est pas parcouru ni supprimé.

3.7 Système de permissions

Dans la section précédente, nous avons utilisé la commande `ls(1)` avec l'option -l. Nous allons maintenant expliquer ce que signifient les informations que cette commande affiche.

3.7.1 Les permissions

```
$ ls -alh
total 32k
drwxr-s---    2 patrick  patrick      4.0k Dec 27 04:30 .
drwxr-s---    5 patrick  patrick      4.0k Dec 27 04:45 ..
-rw-r-----    1 patrick  patrick        38 Dec 27 02:58 output
lrwxrwxrwx    1 patrick  patrick         6 Dec 27 04:11 sym -> sortie
```

```
[ ... ]
$
```

La valeur affichée sur la première ligne indique la taille occupée par les fichiers du répertoire. A noter qu'un lien symbolique n'occupe pas d'espace disque (la place nécessaire pour enregistrer la destination est prise sur l'inode du répertoire si sa taille est inférieure à 60 octets).

Pour la seconde colonne, voir la section précédente.

La troisième colonne indique quel est le propriétaire du fichier (ici "`patrick`"), et la quatrième colonne indique quel est le groupe du fichier (ici "`patrick`" également).

Les informations suivantes sont la taille (ici affichées dans un format lisible pour un être humain grâce à l'option "`-h`"), la date, l'heure et le nom.

La première colonne se décompose en 10 caractères.

Le premier caractère correspond au type de fichier (voir table 3.1).

TAB. 3.1 – Types de fichiers

-	Fichier régulier
d	Répertoire
l	Lien symbolique
b	Périphérique de type bloc
c	Périphérique de type caractère
s	Socket
p	Tube nommé (Named pipe)

Les neuf caractères suivants se subdivisent en trois groupes :

– Les permissions du propriétaire
– Les permissions du groupe
– Les permissions des autres

Les trois caractères des permissions `rwx` peuvent prendre les valeurs suivantes (voir table 3.2) :

3.7.2 Modification des permissions

Maintenant, voyons comment modifier les bits de permissions avec la commande `chmod(1)`. Nous omettrons quelques syntaxes particulières (voir les systèmes d'aide `info` ou `man`).

Le format symbolique

Syntaxe de la commande `chmod(1)` (mode symbolique) :

```
chmod MODE[,MODE] fichiers
```

Le paramètre `MODE` est formé par la concaténation d'un à trois caractères `ugoa` , d'un caractère `+-` et d'un ou plusieurs caractères `rwxXstugo` , dont la signification est indiquée dans la table 3.7.2.

Exemples :

```
$ chmod ug+rw,o-rwx *
$ chmod u+s prog
```

TAB. 3.2 – Les valeurs des bits de permissions

Bit	Valeurs	Significations
r	r	Permission en lecture
r	-	Interdit en lecture
w	w	Permission en écriture
w	-	Interdit en écriture
x	x	Permission en exécution. Pour un répertoire, signifie que l'on peut rentrer (avec la commande `cd`*(1)*) dans ce répertoire, ou lister son contenu (avec la commande `ls`*(1)*).
x	-	Interdit en exécution
x	s	Donner les droits de l'utilisateur ou groupe lors de l'exécution
x	t	Pour un répertoire, n'autoriser la suppression et le renommage qu'au propriétaire du répertoire et au propriétaire des fichiers contenus dans le répertoire

TAB. 3.3 – Décomposition du MODE symbolique de chmod

Utilisateurs	u	Propriétaire (user)
	g	Groupe
	o	Autres (other)
	a	Tous les utilisateurs (all)
Action	+	Ajouter les permissions
	-	Supprimer les permissions
Bits	r	Lecture
	w	Écriture
	x	Exécution
	X	Exécution si le fichier est un répertoire ou a déjà un bit x positionné
	s	Set (GID ou UID)
	t	Sticky bit
	u	Copier les permissions du propriétaire
	g	Copier les permissions du groupe
	o	Copier les permissions des autres

Le format octal

Syntaxe de la commande `chmod` (mode octal) :

```
chmod OCTAL fichiers
```

Le format octal est constitué de 4 chiffres en octal, le premier étant optionnel (voir leur décomposition dans la table 3.7.2) :

TAB. 3.4 – Décomposition du mode OCTAL de chmod

1^{er} octet	4	Set User ID bit
	2	Groupe
	1	Sticky bit
$2^{ème}$ au $4^{ème}$ octet	4	Read bit
	2	Write bit
	1	Execute bit

La valeur des chiffres est obtenue en composant les valeurs des bits à l'aide de l'addition. Soit pour positionner les bits "`r`" et "`w`", on ajoute $4+2$ ce qui donne 6.

L'équivalent des commandes symboliques est (dans le second cas, on suppose que l'on souhaite accorder les droits en lecture, mais pas en écriture) :

```
$ chmod 660 *
$ chmod 4755 prog
```

Modification récursive

On constate qu'il est plus facile de gérer le format symbolique, car il n'est pas nécessaire de spécifier les valeurs de tous les bits. Ceci est particulièrement pratique lorsque l'on modifie des permissions de façon récursive en utilisant l'option "`-R`".

```
$ chmod -R u+rw,g+r,g-w,o-rwx /home/patrick/*
$
```

Il est aussi possible d'utiliser la commande `find`*`(1)`* pour faire des modifications récursives. Notamment, si l'on ne souhaite effectuer les modifications que sur les fichiers ou les répertoires, ou, si l'on souhaite n'effectuer les modifications qu'à partir d'un répertoire particulier.

La première commande ne modifie que les fichiers, la seconde ne modifie que les répertoires, et la troisième ne modifie que les fichiers et répertoires contenus dans le répertoire nommé `cible` , et la quatrième réalise la même chose que la troisième, mais à l'aide de la commande `xargs`*`(1)`*.

```
$ find . -type f
./1/cible/fic
./2/cible/fic
./3/cible/fic
$ find . -type f -exec chmod u+rw {} \;
.
./1
```

```
./1/cible
./2
./2/cible
./3
./3/cible
$ find . -type d -exec chmod g+rwx {} \;
$ find . -name cible
./1/cible
./2/cible
./3/cible
$ find . -name cible -exec chmod -R og-rwx {} \;
$ find . -name cible | xargs chmod -R og-rwx
```

La commande `find` permet de parcourir une arborescence de répertoires, elle prend pour premier paramètre un répertoire (ici "`.`", c'est-à-dire le répertoire courant). Elle applique ensuite une sélection sur les fichiers et répertoires parcourus (ici, soit une sélection sur le type ("`f`" pour fichier, ou "`d`" pour répertoire).

La fonction exécutée par défaut est l'affichage des noms des fichiers et répertoires parcourus (l'option "`-print`" est l'option par défaut). Elle peut être remplacée par l'action "`-exec`".

La syntaxe de l'action "`-exec`" est singulière. Tout ce qui suit cette option, jusqu'au "`;`" est considéré comme la commande à exécuter et ses paramètres. Le "`;`" doit être échappé, car c'est le séparateur de commandes du shell, et il ne serait pas vu par `find` si le "\". était omis. La séquence "{}" est remplacée par le nom du fichier ou répertoire.

La commande `xargs` reçoit en paramètre le nom d'une commande à exécuter. Elle lit dans son entrée standard une liste de paramètres séparés par des espaces, tabulations ou retour chariot. Elle transmet chacun de ces paramètres à cette commande.

3.7.3 Choisir les permissions par défaut

Lorsqu'un fichier est créé, ses permissions sont attribuées en fonction d'un masque que chaque utilisateur peut modifier selon ses préférences à l'aide de la commande interne `umask`*`(bash(1))`*.

Elle peut être soit utilisée avec les permissions symboliques, dans ce cas, on indique quelles sont les permissions que l'on autorise pour chacun des propriétaire, groupe et autres. Mais aussi avec les permissions numériques en octal, et dans ce cas, la valeur indique les bits que l'on souhaite supprimer.

Ainsi, si je souhaite que mon groupe puisse lire, et exécuter, mais pas écrire, et que les autres n'auront aucun droit, je peux utiliser l'une des deux commandes suivantes :

```
$ umask u=rwx,g=rx,o=
$ umask 027
```

3.7.4 Modifier le propriétaire et/ou le groupe d'un fichier

Un utilisateur peut modifier le groupe d'un fichier qui lui appartient, à condition qu'il fasse lui-même partie du nouveau groupe, mais il ne peut modifier le propriétaire du fichier. Le changement de propriétaire se fait automatiquement si le destinataire copie le fichier.

Le super utilisateur peut, bien évidemment, modifier tant le groupe que le propriétaire, sans aucune restriction.

Pour modifier le groupe d'un fichier, on utilise la commande `chgrp(1)`. La syntaxe est `chgrp groupe fichiers` , où `groupe` est un groupe existant, et fichiers est une liste de fichiers et/ou de répertoires. une option `-R` permet de parcourir les répertoires récursivement.

Pour modifier le propriétaire d'un fichier, on utilise la commande `chown(1)`. La syntaxe est sensiblement la même que pour `chgrp(1)` (`chown proprio fichiers`), mais `proprio` représente le propriétaire, éventuellement suivi d'un nom de groupe séparé par le symbole " `:` " ou le symbole " `.` ".

```
# chgrp patrick *.tex *.dvi
# chown -R tuxbihan:devel *.c* *.h*
```

3.8 Déterminer le type de fichier

Les extensions de fichiers, c'est-à-dire, les caractères situés derrière le dernier point, sont en général une bonne façon de déterminer le type de fichier auquel on a affaire. Par exemple, un fichier terminé par l'extension `.tex` est probablement un fichier LaTeX. Mais il n'existe aucune obligation d'utiliser une telle extension significative, et de nombreux fichiers n'ont pas d'extensions, notamment les commandes ***Unix*** . Pour savoir à quel type de fichier on a affaire, on se base donc sur les permissions, pour savoir si le fichier est exécutable ou non, et on peut utiliser la commande `file(1)` pour s'assurer du contenu exact d'un fichier. Elle analyse le contenu du fichier pour donner un diagnostic.

Si vous souhaitez creuser un peu plus le fonctionnement de cette commande, regardez le fichier `/usr/share/misc/magic` et sa description (`magic(5)`).

3.9 Copie de fichiers

Lorsque l'on copie un fichier, les permissions du fichier, ainsi que le propriétaire sont modifiés. Si je suis l'utilisateur `patrick` et que je copie le fichier `.emacs.el` qui appartient à l'utilisateur `tuxbihan` , ce que je peux faire si j'ai le droit de lire ce fichier, la copie de ce fichier m'appartient, et ses permissions sont réévaluées en fonction de la valeur de `umask(bash(1))`.

Ce comportement est satisfaisant dans la quasi totalité des cas, mais il pose problème lorsque que l'on souhaite faire une copie conforme des fichiers. C'est le cas, par exemple, quand `root` souhaite copier des fichiers qui ne lui appartiennent pas pour les déplacer d'une partition vers une autre, ou d'un répertoire vers un autre. Si cette copie est faite sans précautions, à la fin de la copie, tous les fichiers appartiennent à `root` , et leur ancien propriétaire ne peut plus les utiliser.

Plusieurs options de la commande `cp(1)` sont utiles dans ce cas (`-p` pour préserver les attributs des fichiers, `-d` pour ne pas suivre les liens symboliques, `-R` pour parcourir récursivement les répertoires), l'une d'elles combine les autres (`-a` qui est équivalente à `-dpR`).

De plus, pour copier des partitions entières, il faut utiliser l'option `-x` qui interdit à la commande `cp(1)` de changer de système de fichiers au cours de la copie.

Chapitre 4

Gestion des utilisateurs

Dans le chapitre précédent, nous avons abordé la notion d'utilisateurs et de groupes, mais si l'installation de votre distribution a créé au moins deux utilisateurs automatiquement (`root` et un utilisateur normal), la création de nouveaux utilisateurs est laissée à votre discrétion. La tâche de gestion des utilisateurs et groupes incombe au super utilisateur (`root`).

4.1 Les fichiers d'utilisateurs et de groupes

Les utilisateurs sont stockés dans le fichier `/etc/passwd` , dont le format est décrit dans la page de manuel `passwd(5)`. Les mots de passe, qui peuvent être modifiés à l'aide de la commande `passwd(1)` peuvent être stockés, soit dans le fichier `/etc/passwd` , soit dans le fichier `/etc/shadow` .

Les groupes sont stockés dans le fichier `/etc/group` , dont le format est décrit dans la page de manuel `group(5)`.

Seul `root` est habilité à faire ces manipulations. Il faut donc soit se connecter comme `root` dans une console virtuelle, soit utiliser la commande `su(1)` (voir section 1.5) pour changer d'identité.

4.2 Créer et supprimer des utilisateurs

Pour créer un nouvel utilisateur, plusieurs commandes sont disponibles. `adduser(8)` présente l'avantage de poser des questions pour obtenir les paramètres dont elle a besoin[1].

Pour supprimer un utilisateur, utiliser la commande `deluser(8)`.

```
# adduser newuser
Adding user newuser...
Adding new group newuser (1004).
Adding new user newuser (1004) with group newuser.
Creating home directory /home/newuser.
Copying files from /etc/skel
```

[1] Attention, les utilisateurs de Mandrake seront très déçus de s'apercevoir que cette commande n'est qu'un lien vers la commande `useradd(1)`

```
Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully
Changing the user information for newuser
Enter the new value, or press return for the default
Full Name []:
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [y/n] y
# grep newuser /etc/passwd
newuser:x:1004:1004:,,,:/home/newuser:/bin/bash
# grep newuser /etc/shadow
newuser:$1$9y3d/b5S$e5s2A.$UjydgPiOHklz19w.:12067:0:99999:7:::
# deluser newuser
Removing user newuser...
done.
# grep newuser /etc/passwd
#
```

4.3 Créer et supprimer des groupes

Les commandes `addgroup(8)` et `delgroup(8)` sont les pendants des fonctions de gestion des utilisateurs.

```
# addgroup thegroup
Adding group thegroup (105)...
Done.
# grep thegroup /etc/group
thegroup:x:105:
# delgroup thegroup
Removing group thegroup...
done.
```

4.4 Ajouter un utilisateur dans un groupe

La commande `groups(1)` permet de vérifier à quels groupes un utilisateur appartient. La syntaxe est `groups utilisateur` , où `utilisateur` est une liste d'utilisateurs, qui est optionnelle si l'on souhaite vérifier à quels groupes l'utilisateur courant appartient.

```
# groups patrick tuxbihan
patrick : patrick dialout cdrom floppy audio dip www-data src devel
tuxbihan : tuxbihan audio
```

Les groupes auxquels appartient un utilisateur déterminent ses droits d'utilisation de certains périphériques, ainsi que ses droits en lecture/écriture de certains fichiers.

Au-dessus, on voit que l'utilisateur `tuxbihan` appartient au groupe `audio` , ce qui lui donne le droit d'accéder aux périphériques appartenant au groupe audio.

Les deux commandes ci-dessous permettent de voir quels sont ces périphériques.

```
$ find /dev -group audio
$ ls /dev | grep audio
```

L'utilisateur `patrick` appartient aussi aux groupes `cdrom` , ce qui permet notamment d'utiliser `cdparanioa(1)`[2], au groupe `floppy` , ce qui permet d'utiliser la commande `dd(1)` sur le lecteur de disquettes.

Pour information, `dd(1)` permet de créer une image disque, ou de copier une image disque sur un support, par exemple :

```
$ dd if=/dev/fd0 of=/tmp/img
$ dd of=/dev/fd0 if=/tmp/img
```

Pour visualiser les périphériques concernés, on peut utiliser la commande `find(1)`, avec une recherche sur le groupe et l'opérateur OU - `-o` :

```
$ find /dev -group cdrom -o -group floppy -o group audio -o group dialout
$
```

Pour ajouter un utilisateur à un groupe, on peut utiliser la commande `adduser(8)`. Bien sûr, seul `root` peut faire cela.

```
# adduser tuxbihan floppy
Adding user tuxbihan to group floppy...
Done.
# groups tuxbihan
tuxbihan: tuxbihan floppy audio
#
```

Pour que l'appartenance au nouveau groupe prenne effet, il faut que l'utilisateur `tuxbihan` se reconnecte.

[2]Sur un système utilisant des périphériques SCSI ou IDE-SCSI, il faudra aussi changer le groupe des périphériques `/dev/sg*`

Chapitre 5

Contrôle des processus

Nous allons dans ce chapitre étudier les fonctions qui permettent de manipuler les processus, c'est-à-dire, leur envoyer des signaux, les stopper, les tuer, les mettre en arrière-plan, et les ramener en avant-plan.

5.1 Visualiser les processus en cours

Les processus en cours d'exécution sont listés par la commande `ps(1)`. La première ligne de `ps` indique la signification des différentes colonnes.

```
$ ps
 PID TTY          TIME CMD
1058 pts/4    00:00:00 bash
1225 pts/4    00:00:00 ps
$
```

D'autres options permettent d'afficher plus d'informations. Les options "`a`" et "`x`" permettent de voir l'ensemble des processus (Référez-vous à la documentation de `ps(1)` pour une description complète des très nombreuses options de cette commande).

Voici un petit extrait d'une sortie de générée par `ps ax --forest` .

```
$ ps ax --forest
 PID TTY      STAT   TIME COMMAND
[...]
 995 ?        S      0:12 /usr/bin/emacs -g 110x67
 997 pts/3    S      0:00  \_ /bin/bash
1058 pts/4    S      0:00  \_ /bin/bash -i
[...]
$
```

L'option `--forest` indique les liens de parenté entre les processus. Au-dessus, le processus `emacs` a lancé deux processus fils `bash` , l'un d'eux est un shell emacs et l'autre est utilisé pour compiler les fichiers LaTeX.

5.2 Lancer un processus en arrière-plan

La fonctionnalité la plus fréquemment utilisée, avec l'envoi de signaux, est l'exécution en arrière-plan d'une commande. L'exécution en arrière-plan est utile quand une commande risque de durer longtemps, et que l'on ne souhaite pas attendre la fin de son exécution.

Par exemple, la commande `updatedb` (voir sous-section 1.3.3), qu'il vaut mieux lancer sous le compte de root, car il peut accéder à tous les répertoires, met beaucoup de temps à s'exécuter.

Pour lancer un processus en arrière-plan, il faut terminer la ligne de commande par un "`&`".

```
# updatedb &
[1] 1234
```

Le nombre entre crochets est le numéro de job (voir `jobs`*`(bash(1))`*). Le nombre qui suit est le `PID`, c'est-à-dire l'identifiant du processus.

Ce n'est pas forcément une bonne idée de lancer en arrière-plan un processus qui génère beaucoup de sorties sur l'écran, car ces sorties vont perturber votre travail en s'insérant au milieu des sorties des programmes que vous allez exécuter par la suite. Vous pouvez utiliser les redirections (voir la section 3.3) pour envoyer ces sorties vers un fichier.

5.3 Interrompre un processus

Il est parfois nécessaire de mettre fin au processus actif, (parce que l'on a fait une erreur en le lançant, par exemple).

Il suffit de presser la combinaison de touches "`Ctl C`".

Si le processus n'est pas actif sur le même terminal, on peut l'interrompre en utilisant la commande mal nommée `kill`*`(1)`*. La commande `kill` permet en fait d'envoyer un signal à un processus. Ce signal peut être un signal d'interruption, mais pas nécessairement, bien que le signal envoyé par défaut (`SIGHUP`) mette fin au processus.

Pour visualiser les effets de ces commandes, nous allons utiliser deux programmes C que nous pourrons lancer et interrompre à volonté.

Le premier (voir figure 5.1) affiche un caractère "`X`" sur le terminal toutes les secondes. Ses sorties vont donc se mélanger avec celles des processus courants.

Le second (voir figure 5.2) n'affiche rien, il utilise les ressources processeur autant qu'il le peut, en exécutant du code en permanence.

Nous allons compiler ces fichiers avec les lignes de commande suivantes :

```
$ cc -o longcmd longcmd.c
$ cc -o long-and-silent long-and-silent.c
```

```
$ ./longcmd &
[1] 1594
$ kill 1594
```

Pour détruire le processus, nous aurions pu utiliser également le numéro de job avec la commande `kill %1`, ou encore `kill ‘pidof longcmd‘`. La commande

`pidof(8)`[1] retourne le (ou les) `PID` du (ou des) processus dont le nom est passé en paramètre. Attention à ne pas détruire d'autres processus que celui que vous souhaitez détruire !

5.4 Stopper un processus et le réactiver

Lorsqu'un processus n'a pas été lancé en arrière-plan, et que l'on souhaite reprendre la main, il n'est pas nécessaire d'interrompre le processus. On peut le suspendre à l'aide de la séquence de touches `C-Z(bash(1))`. On retourne au processus stoppé avec la commande interne `fg(bash(1))`.

```
$ ./longcmd
xxxx^Z
[1]+  Stopped                 ./longcmd
$ ps x
 PID TTY      STAT   TIME COMMAND
[...]
1233 pts/4    T      0:00 ./longcmd
1234 pts/4    R      0:00 ps x
$ fg
./longcmd
xxxxx^C
```

[1] Il a été constaté sur des distributions Mandrake que cette commande ne fonctionnait pas.

```
#include <stdio.h>
int
main()
{
  while(1) {
    printf("x");
    fflush(stdout);
    sleep(1);
  }
}
```

FIG. 5.1 – Code source de longcmd.c

```
#include <stdio.h>
int
main()
{
  for(;;);
}
```

FIG. 5.2 – Code source de long-and-silent.c

```
$
```

La lettre “T” dans la colonne `STAT` indique que le processus est stoppé. On peut constater que les caractères “x” que le processus affichait ne sont plus générés.

On peut également, après avoir stoppé le processus, le remettre en fonctionnement, mais en le maintenant en arrière-plan, à l’aide de la commande `bg`*`(bash(1))`*.

```
$ ./longcmd
xxxx^Z
[1]+  Stopped                 ./longcmd
$ bg %1
[1]+ ./longcmd &
$ xxxxps x
 PID TTY      STAT   TIME COMMAND
[...]
1242 pts/4    S      0:00 ./longcmd
1246 pts/4    R      0:00 ps x
$ xxxxkill %1
[1]+  Terminated              ./longcmd
```

On voit qu’après avoir mis le processus en arrière-plan, les caractères “x” qu’il affiche se mélangent à l’affichage. Pour mettre fin au processus, on utilise la commande `kill %1` , qui a pour effet de détruire le job n°1.

5.5 Envoyer un signal à tous les processus de même nom

La commande `killall`*`(1)`* envoie un signal à tous les processus qui portent le même nom. Faire attention aux éventuelles conséquences !

5.6 Envoyer le signal HUP à un daemon

Un moyen courant de forcer un daemon à relire ses fichiers de configuration est de lui envoyer le signal `HUP` . Par exemple, après avoir modifié la configuration du daemon `inetd` :

```
# emacs /etc/inetd.conf
# kill -HUP ‘pidof inetd‘
```

5.7 Lancer une commande avec `nohup`*`(1)`*

Les essais réalisés avec une Debian 3.0 (kernel 2.4.18) n’ont pas permis de mettre en évidence l’intérêt de la commande `nohup` , puisqu’une commande lancée simplement en arrière-plan se comporte comme si elle avait été lancée avec `nohup` . Pour mémoire, cette commande est sensée maintenir active une commande (en redirigeant ses sorties vers le fichier `nohup.out`) après que l’utilisateur qui l’a lancée se soit déconnecté.

5.8 Exécuter une commande avec `nice(1)`

La commande `nice` lance une commande en arrière-plan avec un paramètre qui indique sa priorité d'exécution.

Pour mettre en évidence l'effet de la commande, nous utiliserons la commande `top(1)`. Cette dernière affiche les temps CPU utilisés par chaque processus.

```
$ nice -n 20 ./long-and-silent &
[9] 1033
$ ./long-and-silent &
[10] 1034
$ top
93 processes: 89 sleeping, 4 running, 0 zombie, 0 stopped
CPU states:  71.4% user,  21.4% system,   7.1% nice,   0.0% idle
Mem:    384824K total,   200868K used,   183956K free,     5172K buffers
Swap:   128484K total,        0K used,   128484K free,   100960K cached

 PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
1034 patrick   16   0   252  252   208 R    57.2  0.0   0:32 long-and-silent
1035 patrick   12   0   940  940   716 R    14.3  0.2   0:04 top
 595 root       5 -10 66964  17M  3400 S <   7.1  4.6   1:02 XF86_SVGA
 656 patrick    9   0  4544 4544  3492 R     7.1  1.1   0:03 gnome-terminal
 682 patrick    9   0  4664 4664  3616 S     7.1  1.2   0:05 deskguide_apple
1033 patrick   19  19   252  252   208 R N   7.1  0.0   0:04 long-and-silent
[...]
$
```

On voit bien ici que le processus lancé avec `nice` , dont le PID est 1033 a consommé moins de temps CPU (7,1 %) que le processus lancé sans `nice` dont le PID est 1034 et qui a consommé 57,2 % de CPU.

Chapitre 6

Programmation du shell

Nous allons écrire des programmes pour `bash(1)`. Comme chaque programme devra contenir le chemin de `bash` , il est nécessaire de le localiser. Nous utiliserons pour cela la commande `which(1)`.

```
$ which bash
/bin/bash
$
```

Toutes les commandes que nous avons utilisées jusqu'ici peuvent être enregistrées dans un fichier, dont la première ligne serait :

```
#!/bin/bash
```

Un tel fichier constituerait déjà un programme shell, mais nous allons étudier ici quelques structures qui permettent réellement de contrôler le déroulement du programme, comme le permet tout langage de programmation.

6.1 Les paramètres d'un programme shell

Les paramètres passés à un programme shell sont déréférencés en utilisant la notation "`$x`", où "`x`" est un numéro de "`1`" à "`9`". Ce qui limite le nombre de paramètres utilisables simultanément à 9. Heureusement, la commande interne `shift(bash(1))` permet d'atteindre les paramètres suivants, mais le paramètre "`$1`" devient alors inaccessible.

Essayons notre premier programme (voir figure 6.1) :

FIG. 6.1 – Premier programme shell : prg-hello

```
#!/bin/bash

echo Bonjour $1
```

Nous pouvons l'exécuter comme ceci, en passant ce fichier comme paramètre à la commande `bash(1)` :

```
$ bash prg-hello
Bonjour
$
```

ou comme ceci, en rendant le fichier exécutable :

```
$ chmod +x prg-hello
$ ./prg-hello
Bonjour
$ ./prg-hello Patrick
Bonjour Patrick
$
```

6.2 La commande `read`*(bash(1))*

Modifions-le ainsi en utilisant la commande interne `read`*(bash(1))* (voir figure 6.2) :

FIG. 6.2 – Second programme shell : prg-hello2

```
#!/bin/bash

read name
echo Bonjour $name
```

La commande interne `read`*(bash(1))* lit au clavier un texte terminé par un retour chariot, et le stocke dans la variable passée en paramètre. Comme pour le paramètre "`$1`", le déréférencement se fait en précédant la variable par le symbole "`$`".

6.3 `if`*(bash(1))*

Si l'utilisateur rentre une ligne vide, on indique un message d'erreur (voir figure 6.3) :

Le "\" avant le "`'`", car, rappelez-vous, c'est le caractère qui permet d'éviter l'interpolation de variables.

Le test aurait pu être inversé en utilisant le caractère " `!`" devant les crochets. " `!`" signifie `non` . Faites l'essai.

Toutes les expressions conditionnelles qui peuvent être évaluées sont listées dans la rubrique **CONDITIONAL EXPRESSIONS** du manuel de `bash`*(1)*.

Le test `if`*(bash(1))* peut être plus complexe, avec des tests dans la clause else, ce qui donne : `if ... then ... elif ... then ... else ... fi` .

6.4 La boucle `until`*(bash(1))*

Voici comment parcourir la liste des paramètres de la ligne de commande à l'aide de la boucle `until`*(bash(1))*. Nous aurions pu utiliser la boucle `while`*(bash(1))* en inversant le test avec " `!`".

```
#!/bin/bash

until [ "$1" = "" ]
do
  echo parm: $1
  shift
done
```

6.5 La boucle `for`*(bash(1))*

Avec une boucle `for`*(bash(1))* et la variable "`$*`" qui représente la liste de tous les paramètres, nous aurions pu écrire ce programme comme ceci. A noter que "$#" donne le nombre de paramètres reçus.

```
#!/bin/bash

for i in $*
do
  echo parm: $i
  shift
done
```

6.6 La structure de choix `case`*(bash(1))*

La structure `case`*(bash(1))* permet de sélectionner une option parmi plusieurs. On gagne en lisibilité lorsque le nombre de choix est important par rapport à un test `if ... then ... elif ... then ... else ... fi`.

```
#!/bin/bash
```

FIG. 6.3 – Troisième programme shell : prg-hello3

```
#!/bin/bash

echo -n "Comment vous appelez-vous ? "
read name
if [ "$name" = "" ]
then
  echo Vous n\'avez pas donné votre nom !
else
  echo Bonjour $name
fi
```

```
# Test de $SHELL
case $SHELL in
  "/bin/bash" ) echo Votre shell est un bon shell;;
  "/bin/csh"  ) echo Pas mal non plus;;
             *) echo Vous devriez utiliser le shell bash;;
esac
```

Testons maintenant ce programme :

```
$ ./case-struct
Votre shell est un bon shell
$ env SHELL=autre ./case-struct
Vous devriez utiliser le shell bash
```

6.7 Sortie de boucle

Nous signalerons, sans nous attarder, les instructions `break`*`(bash(1))`* qui permet de quitter une boucle et `continue`*`(bash(1))`* qui permet de quitter le corps de la boucle et de sauter au test suivant.

```
#!/bin/bash

for i in 1 2 3
do
  if [ $i -eq 2 ]
  then
    break
  else
    echo $i
  fi
done
```

```
#!/bin/bash

for i in 1 2 3
do
  if [ $i -eq 2 ]
  then
    continue
  else
    echo $i
  fi
done
```

6.8 Renommer des fichiers

Après avoir utilisé `cdparanoia(1)` pour extraire les pistes audio de votre chanteur préféré, et `oggenc(1)`, vous souhaitez renommer les fichiers `track01.ogg` à `track20.ogg` en `Klaus-Nomi-01.ogg` à `Klaus-Nomi-20.ogg` .

```
for i in track*.ogg
do
  mv $i ${i/track/Klaus-Nomi-}
done
```

6.9 Convertir des noms de fichier des minuscules vers les majuscules

Voici une ligne de commande pour renommer les noms fichiers en minuscules en noms de fichiers en majuscules (pour éviter les accidents malencontreux, cette commande inverse les minuscules et les majuscules, donc, si par accident, vous exécutez cette commande dans un répertoire critique, il suffit d'attendre qu'elle se termine et la relancer à nouveau).

Exécution dans un seul répertoire :

```
$ for i in * ; do mv $i $(echo $i | tr a-zA-Z A-Za-z); done
```

Exécution dans toute une arborescence :

```
$ find . -type f -exec bash -c 'src={};  dst=$( \
echo $src | tr a-zA-Z A-Za-z); \
[ $src != $dst ] && mv $src $dst' \;
$
```

Un petit exercice à réaliser pour ceux qui veulent creuser ; faire la même commande qu'au-dessus mais en modifiant aussi les répertoires ! Le problème est que si l'on se contente d'enlever le '-type f' les répertoires sont renommés avant que find n'ait le temps de les parcourir, et find sort une erreur, car il essaie d'ouvrir un répertoire en minuscules, alors qu'il vient d'être renommé en majuscules.

Une solution a été donnée par Peter Stopschinski (voir figure 6.4 :

FIG. 6.4 – Solution de Peter Stopschinski

```
$ find . -depth -exec bash -c 'src={} ; srcA=${src%/*} ; \
  srcZ=${src##*/} ; dst=$(echo $srcZ | tr a-zA-Z A-Za-z); \
  [ $srcZ != $dst ] &&  mv $src ${srcA}/$dst' \;
```

6.10 Exemples de programmes

6.10.1 Vider le spool de `exim(1)`

Ce programme (voir figure 6.5) permet de supprimer tous les mails en attente dans le spool d'exim. Il recherche tous les fichiers qui se terminent par `-H` , ce sont les fichiers qui contiennent les entêtes. Leur nom est le numéro de mail en attente suivi de `-H` . Il appelle la commande `exim(1)` avec l'option `-Mrm` en supprimant le `-H` du nom de fichier trouvé.

FIG. 6.5 – Vidage du spool exim

```
#!/bin/bash

# $i scans all the files in the exim spool
cd /var/spool/exim/input
for i in *-H
do
  /usr/sbin/exim -Mrm ${i/-H/}
done
cd -
```

6.10.2 Générer de beaux fichiers PDF

Ce programme (voir figure 6.6) pourrait être amélioré en testant le résultat des commandes `pslatex` , `dvips` et `ps2pdf` . L'écriture du test pourrait utiliser "$#".

La manière d'écrire ce test est très *fonctionnelle*, c'est une écriture de type Lisp, souvent adoptée dans les programmes Perl.

6.10.3 Extraire des pistes audio et les convertir en Ogg Vorbis

Ce programme (voir figure 6.7) réalise successivement une extraction de pistes audio avec cdparanoia1, un encodage au format Ogg Vorbis (en faible qualité) avec `oggenc(1)`, puis un renommage des fichiers avec le nom de l'auteur et le titre du disque. Un exemple d'utilisation :

```
$ extract2ogg "Tracy Chapman  Cross Roads"
[...]
$ ogg123 "Tracy Chapman  Cross Roads 01.ogg"
```

FIG. 6.6 – Génération PDF

```
#!/bin/sh

# Un paramètre: $1: Nom du fichier à compiler
# Si plus ou moins de paramètres, on arrête.

usage () {
 echo "$0: $0 myfile.tex"
 echo "Generates a clean myfile.pdf file and a myfile.ps"
}

[ ! "$1" ] || [ "$2" ] && {
 usage
 exit 1
}

pslatex \\nonstopmode\\input $1
dvips ${1/.tex/.dvi}
ps2pdf -dPDFSETTINGS=/printer ${1/.tex/.ps}
```

FIG. 6.7 – Extraction audio

```
#!/bin/bash

# Set defaults values
no_extract=no
no_rm_wav=no
tracks=""

# Check options
for (( ; 1 ; )) ; do
    case "$1" in
 ( '--no-extract' | '-e' ) no_extract=yes;;
 ( '--no-rm_wav' | '-r' ) no_rm_wav=yes;;
 ( '--tracks' | '-t' ) tracks=$2; shift;;
 ( * ) break
    esac
    shift
done

usage () {
    echo
    echo "Usage: extract2ogg OPTIONS \"Nom Chanteur  Titre Disque\""
    echo "extracts a CD"
    echo "Converts files to ogg Vorbis format"
    echo "Renames files to \"Nom Chanteur  Titre Disque\""
    echo "OPTIONS:"
    echo "if --tracks is specified, extracts only the specified tracks (see cdparanoia for
    echo "if --no-extract is specified, doesn\'t performs extraction"
    echo "if --no-rm-wav is specified, Wav files are not removed after processing"
    echo
}

# Test if requested number of parameters is ok
[ ! "$1" ] || [ "$2" ] && {
 usage
 exit 1
}

# Perform extraction of specified tracks
[ "$no_extract" == 'yes' ] || {
    if [ -z "$tracks" ]; then
 cdparanoia -B
    else
 cdparanoia -B "$tracks"
    fi
}

# Rename all the files *.cdda.wav to *.wav
for i in *.wav; do
    mv $i ${i/.cdda/}
done

# Convert wav to ogg Vorbis
oggenc -q1 *.wav

# rename all files to the specified name
for i in *.ogg; do
    dst=${i/track/$1 }
    bash -c "mv $i \"$dst\""
```

Chapitre 7

Commandes utiles

Ce chapitre regroupe (un peu en vrac pour l'instant, en attendant mieux) quelques fonctions, commandes utiles.

7.1 Simplifier la saisie de commandes

La commande interne `alias`*`(bash(1))` permet de substituer une chaîne de caractères à un nom lorsque celui-ci est utilisé comme une commande. Son usage le plus fréquent est de passer automatiquement des paramètres à une commande utilisée souvent, sans avoir à saisir les paramètres à chaque fois.

Sa syntaxe est simple :

```
  alias nom_d_alias="chaine de remplacement"
ou
  alias nom_d_alias='chaine de remplacement'
```

Le choix des quotes utilisées est fonction du fait que l'on souhaite, ou non, que le shell interprète le contenu de la chaîne avant de créer l'alias.

Sans paramètre, la commande `alias` affiche tous les alias existants.

Le plus souvent les commandes `alias`*`(bash(1))` sont regroupées dans le fichier `.bashrc` qui est interprété par le shell bash lorsqu'il démarre. Ces `alias` deviennent donc permanents et peuvent être réutilisés à chaque session. A noter que dans ce fichier, le contenu de la chaîne n'est pas interprété par le shell, même si les quotes doubles sont utilisées.

Un alias peut être supprimé en utilisant la commande `unalias`*`(bash(1))`.

Nous en profitons pour présenter deux commandes utiles `du`*`(1)` et `df`*`(1)` qui permettent respectivement d'indiquer la taille totale occupée par un répertoire (par défaut le répertoire courant), et d'indiquer l'espace disque occupé (et libre) sur chacune des partitions montées. Ces deux commandes ont en commun de pouvoir afficher les valeurs dans un format humainement lisible grâce à l'option `-h` .

```
  $ df
  Filesystem           1k-blocks      Used Available Use% Mounted on
  /dev/hdb5              2071384   1650976    315184  84% /
  /dev/hdb6              2071384   1033728    932432  53% /var
  /dev/hdb9              2071384   1307548    658612  67% /usr/local
```

```
/dev/hdb10              2071384   1731480    234680  89% /home
$ alias df='df -h'
$ df
Filesystem            Size  Used Avail Use% Mounted on
/dev/hdb5             2.0G  1.6G  307M  84% /
/dev/hdb6             2.0G 1010M  910M  53% /var
/dev/hdb9             2.0G  1.3G  643M  67% /usr/local
/dev/hdb10            2.0G  1.7G  229M  89% /home
$ du
8 ./backticks
8 ./find/1/cible
12 ./find/1
8 ./find/2/cible
12 ./find/2
8 ./find/3/cible
12 ./find/3
40 ./find
4 ./listage
40 ./processus
52 ./programmation
20 ./redirection
168 .
$ alias du='du -h'
$ du
8.0k ./backticks
8.0k ./find/1/cible
12k ./find/1
8.0k ./find/2/cible
12k ./find/2
8.0k ./find/3/cible
12k ./find/3
40k ./find
4.0k ./listage
40k ./processus
52k ./programmation
20k ./redirection
168k .
```

7.2 Utiliser la commande `ls`*(1)* en couleur

La commande peut colorier les répertoires, les fichiers, les liens symboliques, ... différemment. Pour cela, il suffit d'ajouter dans le fichier `.bashrc` les lignes suivantes :

```
eval `dircolors`
alias ls="ls --color=auto"
```

Note : La commande `eval`*(bash(1))* interprète dans le shell courant la chaîne de caractères qui lui est transmise en paramètre.

Toutefois, pour les habitués d'X ?Emacs, ceci pose un problème, car l'affichage des couleurs dans le shell Emacs ne donne pas de résultats concluants. La solution consiste en une courte ligne d'alias qui teste la variable d'environnement `TERM` pour savoir si il faut ou non utiliser la coloration. Cette ligne peut être ajoutée au fichier `.bashrc` .

```
alias ls="ls -h $(if [ $TERM = 'dumb' -o $TERM = 'emacs' ]; then echo
--color=no ; else echo --color=auto ; fi)"
alias ll='ls -l'
```

7.3 Archiver des données

`tar`*(1)* est une commande que l'on se doit de connaître, car de nombreux programmes (sous forme de sources ou de binaires) sont distribués dans le format qu'elle génère.

Pour générer une archive : `tar -czvf archive.tgz fichiers` ou `tar -cjvf archive.tar.bz fichiers` .

`fichiers` représente une liste de fichiers et/ou de répertoires. Pour les autres options, voir la table 7.3.

```
$ tar czvf exercices1.tgz exercices
$ tar czf  exercices2.tgz exercices
$ tar czvf exercices3.tgz --exclude '*.o' exercices
$ tar czvvf exercices3.tgz --exclude '*.o' exercices
$ tar cjvf exercices1.tar.bzip exercices
$ ls -l exercices.*
```

Pour lister le contenu d'une archive : `tar -tzvf archive.tgz` ou `tar -tjvf archive.tar.bz` .

```
$ tar tzf exercices1.tgz
$ tar tzvf exercices3.tgz
```

Pour extraire le contenu d'une archive : `tar -xzvf archive.tgz` ou `tar -xjvf archive.tar.bz` .

```
$ mkdir extraction; cd extraction
$ tar xzvf ../exercices1.tgz
```

7.3.1 Comparer des fichiers ou des répertoires

La commande `diff`*(1)* permet de comparer deux fichiers, ou deux répertoires. Elle permet aussi de faire des comparaisons avec l'entrée standard, ce que nous ne verrons pas ici.

Par exemple, pour comparer deux répertoires :

```
$ diff -r exercices/ ../exercices
Only in ../exercices/processus: long-and-silent.o
Only in ../exercices/processus: longcmd.o
Only in ../exercices/redirection: trace.o
$
```

Pour comparer deux fichiers :

```
$ cd exercices/programmation
$ diff prg-hello3 prg-hello2
5,10c5
< if ! [ "$name" = "" ]; then
<  echo Bonjour $name
< else
<  echo Vous n\'avez pas donné votre nom !
< fi
<
---
> echo Bonjour $name
$
```

TAB. 7.1 – Principales options de `tar`

Option	Signification
c	Création d'une archive
t	Listage du contenu d'une archive
x	Extraction du contenu d'une archive
–exclude PATTERN	Exclure les fichiers qui correspondent à PATTERN
z	Compression avec gzip
j	Compression avec bzip2
v	Mode verbeux (plusieurs v possibles)
f FILE	Utiliser le fichier dont le nom est FILE

Chapitre 8

Contributions

La ligne de code qui réalise une conversion majuscules/minuscules des noms de fichiers est de Peter Stopschinski.

Les lecteurs de ce document sont invités à envoyer toute remarque ou toute proposition d'ajout ou de modification à l'auteur[1].

[1] Patrick Percot, ppercot@free.fr

Annexe A

GNU Free Documentation License

This documentation is released under the following license.
GNU Free Documentation License
Version 1.1, March 2000

A.1 PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document ”free” in the sense of freedom : to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation : a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals ; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The ”Document”, below, refers to any such manual or work. Any member of the public is

a licensee, and is addressed as ”you”. A ”Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language. A ”Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ”Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. The ”Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or BackCover Texts, in the notice that says that the Document is released under this License. A ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not ”Transparent” is called ”Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standardconforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine- generated HTML produced by some word processors for output purposes only.

The ”Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ”Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A.3 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4 COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts : Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the

6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

8. Include an unaltered copy of this License.

9. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

11. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

13. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

14. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles. You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard. You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another ; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled ”History” in the various original documents, forming one section entitled ”History”; likewise combine any sections entitled ”Acknowledge- ments”, and any sections entitled ”Dedications”. You must delete all sections entitled ”Endorsements.”

A.7 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an ”aggregate”, and this this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.9 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.10 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http ://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Colophon

Ce document a été écrit en LaTeX, langage de mise en page de documents écrit par Leslie Lamport.

L'éditeur de texte utilisé est Emacs[1], initialement écrit par Richard Stallman.

Le document a été produit sur un système Debian GNU/Linux[2].

L'auteur[3] de ce document l'a réalisé initialement pour l'association Tuxbihan[4].

[1] http ://www.emacs.org

[2] http ://www.debian.org

[3] Patrick Percot, ppercot@free.fr

[4] http ://tuxbihan.apinc.org ou http ://www.tuxbihan.org

Bibliographie

[1] *Documentation Linux (info, man, /usr/share/doc, ...)*

[2] Daniel P. Boyet, Marco Cesati : *Le noyau Linux*, O'Reilly, (Février 2002)

[3] Daniel Robbins, Chris Houser : *LPI certification 101 exam prep, Part 1-3*, IBM developerWorks, (Non daté)

[4] Matt Welsh, Matthias Kalle Dalheimer, Lar KaufMan : *Le système Linux*, O'Reilly, (Mai 2001)

[5] Naba Barkakati : *Red Hat Linux Secrets*, Sybex, (1999)

[6] Franck Gehrke, Peter Klappheck, Forbjörn Glinsky, Günter Gripp : *Linux*, Sybex, (1998)

Table des figures

Liste des tableaux

Index