MiniMax

Des livres qu'on lit vraiment

Collection dirigée par Céline Chevalier

La ligne de commande
par l'exemple

Vincent Fourmond

Développeur Debian
Ancien élève de l'École Normale Supérieure
Chargé de recherche au C.N.R.S.

En un clin d'œil

Avant-propos

La ligne de commande : sous une apparence austère se cache un outil d'une puissance et d'une efficacité remarquables, accessible à tous ceux qui connaissent son language.

Muni de ce livre, vous apprendrez à gérer efficacement vos fichiers, manipuler vos documents, rechercher et mettre en forme des informations, effectuer des sauvegardes, planifier des tâches, travailler à distance sur d'autres ordinateurs et bien d'autres choses. Mieux encore, vous apprendrez qu'en ligne de commande, dès qu'on sait réaliser une opération une fois, la lancer des milliers de fois ne prend que quelques instants : fini, les tâches répétitives et rébarbatives ! Avec la ligne de commande, ce qui était une corvée devient un mini-défi faisant appel à votre créativité : cent fois plus rapide, et autrement plus intéressant !

Ce livre est organisé en chapitres de difficulté croissante, depuis les éléments de base jusqu'à la combinaison de commandes et l'écriture de programmes. Il contient des exercices pour mettre directement en pratique vos acquis, et un lexique, un index et un tableau récapitulatif des commandes pour faciliter vos expérimentations.

La ligne de commande décrite dans ce livre est celle des systèmes Unix (Linux, Mac OS X). Cependant, moyennant l'installation de la suite logicielle (gratuite) Cygwin, vous pouvez retrouver un environnement et une efficacité similaire sous Windows.

Pour vos débuts dans la ligne de commande, profitez de l'expérience de quelqu'un qui l'utilise au quotidien depuis une dizaine d'années ; détendez-vous, et suivez le guide !

J'espère que ce livre vous permettra de tirer le meilleur de la ligne de commande et je vous souhaite autant de plaisir à le lire que j'en ai eu à l'écrire.

Bonne lecture !

Vincent Fourmond

La collection MiniMax

Objectifs

- ▹ Aller à l'essentiel
- ▹ Comprendre comment ça marche
- ▹ Réaliser votre projet

Vous vous lancez dans l'apprentissage d'un nouvel outil. Parmi les centaines de possibilités qu'il recèle, vous devez découvrir et maîtriser rapidement celles qui vous seront utiles.

Vous avez besoin que des spécialistes du sujet sélectionnent pour vous tout ce qui sert vraiment en pratique et vous montrent, exemples à l'appui, comment ça marche.

Suivez le guide

Ce livre est conçu comme un guide de voyage, qui présente un aperçu du pays, des informations ciblées et des phrases de conversation. C'est un guide pratique qui privilégie les besoins les plus fréquents.

Il ne vous conviendra pas si vous pensez que la bonne méthode pour découvrir la Chine, c'est de lire un dictionnaire de chinois.

Note du marketing : il convient à tout le monde !

Mais vous connaissez l'adage : « Choisir, c'est renoncer. »

– Beaucoup d'ouvrages informatiques ne renoncent à rien et deviennent des pavés indigestes.
– D'autres renoncent à tout et se réduisent à des aide-mémoire qui n'expliquent rien.

Dans les MiniMax, vous trouverez enfin les compromis équilibrés et bien construits dont nous avons tous besoin pour démarrer sur un nouveau sujet.

Résumé

- ▸ D'abord le plus utile, ensuite l'accessoire, et rien de superflu.
- ▸ La pratique, pas la théorie.
- ▸ 1 idée = 1 exemple.

Table des matières

Tableau récapitulatif des commandes

Pourquoi utiliser la ligne de commande ?

Chapitre 1

La ligne de commande effraie et fascine en même temps : la suite de symboles semblant attendre que l'on tape quelque chose réveille l'angoisse de la page blanche. Quels mystères sont enfouis derrière cette interface d'apparence aride ? Quelle puissance obscure recèle-t-elle ?

S'il faut un effort certain pour apprivoiser la ligne de commande et tirer parti de sa force, elle vous le rendra au centuple : nombre d'opérations que vous considériez fastidieuses, voire impossibles, se révéleront faciles, rapides, et même amusantes !

1.1 D'où vient la ligne de commande ?

Les premiers ordinateurs n'avaient pas d'interface à proprement parler : on entrait directement l'intégralité du programme que l'on souhaitait exécuter via des cartes perforées ou des rubans, et l'ordinateur en donnait le résultat. Ce n'est qu'avec l'avènement des systèmes de stockage et des fichiers qu'est apparu le besoin de choisir les programmes à exécuter et de gérer les fichiers qu'ils produisaient. Pour ce faire, les informaticiens ont conçu des programmes spécifiques, les interpréteurs de commandes, qui reçoivent des commandes tapées au clavier et les exécutent. Ces commandes sont des petites phrases écrites dans un langage minimaliste compris par l'interpréteur de commandes. Il suffit d'apprendre ce langage pour dialoguer avec la machine. En voici quelques exemples :

```
convert image.jpeg image.png
```

Cette « phrase » demande à l'interpréteur de lancer le programme `convert`, qui se charge de convertir l'image `image.jpeg`, au format JPEG, en une image au format PNG nommée `image.png`.

```
rename -v 's/image(\d+)/$1image/' *
```

permute `image` avec le nombre qui le suit dans le nom de tous les fichiers, en affichant comment chaque fichier est affecté par cette transformation.

```
find . -mmin -10 | xargs cp -t Recents
```

envoie dans le répertoire « `Recents` » une copie de tous les fichiers modifiés au cours des dix dernières minutes.

Les ordinateurs ont ensuite longtemps été cantonnés à des interfaces en mode texte uniquement. Puis, la popularisation par Apple des premiers systèmes d'exploitation « graphiques », suivie par les débuts de Windows, ont amorcé la transition d'une ère du « tout clavier » à celle actuelle du « tout souris », où la plupart des opérations simples sont réalisables en quelques clics bien placés.

« Tout souris » ? Pas complètement : les descendants des premiers interpréteurs de commande sont toujours développés activement à l'heure actuelle, et ils sont encore utilisés par de nombreux adeptes. Par exemple, les hackers des films sont toujours en train de taper furieusement des commandes cryptiques sur des terminaux en mode texte[1]. Sans donner dans le cliché, la ligne de commande reste l'outil de choix pour les administrateurs système, bon nombre de programmeurs, ainsi qu'une proportion importante de ces utilisateurs qui, sans être des professionnels, cherchent à exploiter au maximum les possibilités de leur ordinateur.

1.2 Pourquoi utiliser la ligne de commande ?

Pourquoi ces irréductibles persistent-ils à utiliser cet outil d'apparence si archaïque à l'ère du multimédia ? Ce n'est certainement pas par nostalgie : les interpréteurs de commandes ont beaucoup évolué depuis l'ancien temps. Cherchons donc ailleurs. Qu'est-ce qui peut pousser quelqu'un à préférer taper des commandes au lieu de cliquer sur des icônes ?

Ce qui fait la force des interfaces graphiques actuelles, c'est qu'avec la souris, il est facile de choisir un ou quelques éléments parmi un petit nombre : sélectionner un fichier parmi une trentaine dans un explorateur ou cliquer sur l'un des quelques liens d'une page Web. Ce type d'interface est amplement suffisant pour des besoins modérés. Mais dès qu'il s'agit de fouiller à grand renforts d'ascenseurs dans une hiérarchie complexe de fichiers, d'aller chercher la 1252^{e} photo d'un répertoire parce que c'est justement celle qui nous intéresse, ou de sélectionner un grand nombre de fichiers selon certains critères, la frustration s'installe rapidement, surtout si après avoir passé quelques minutes à sélectionner avec précaution une série de fichiers, on a le malheur de cliquer une fois de trop et de perdre toute la sélection... Nous avons tous été confrontés un jour ou l'autre à ce genre de désagréments. Le fait est qu'il n'y a précisément pas de solution graphique à ces problèmes : la souris est un mode d'expression limité qui ne convient que pour des besoins précis, pour lesquels des programmes spécifiques ont été écrits, ou bien pour des tâches graphiques[2].

Certes, utiliser des raccourcis clavier peut grandement faciliter certaines tâches et permet ainsi de reporter les limites tolérables de la frustration un peu plus loin. Cependant, même avec leur aide, il reste difficile, voire impossible, d'exprimer des idées très naturelles, comme « tous les fichiers au format PDF ».

La ligne de commande est exempte de ces tares. Le mode d'expression qu'elle emploie, plus proche du langage humain, permet de faire passer des idées beaucoup plus complexes. Si l'on devait comparer la souris et le clavier aux différents modes de communication entre êtres humains, la souris

1. Et non, curieusement, en train de cliquer à gauche et à droite sur un écran.

2. Personne ne contredira le fait qu'il est sensiblement plus facile de dessiner à la souris qu'au clavier – sauf dans le cas de fonctions mathématiques.

s'exprimerait uniquement par gestes – et les raccourcis claviers seraient des grognements – alors que le clavier parlerait un véritable langage, légèrement différent du nôtre. Ainsi, en ligne de commande, sélectionner un fichier parmi un million n'est pas plus long que parmi une dizaine, puisqu'on l'appelle par son nom. C'est d'autant plus facile que l'on peut compter sur un système de complétion automatique qui se charge de taper pour vous toutes les parties « évidentes » du nom du fichier ; nous y reviendrons au chapitre suivant. Cette même complétion automatique permet de traverser très rapidement une hiérarchie complexe de fichiers. Par ailleurs, plus de risque qu'un clic malencontreux vous fasse perdre deux minutes passées à sélectionner des fichiers, d'autant plus qu'il ne vous faudra généralement que quelques secondes pour sélectionner les mêmes fichiers en ligne de commande, grâce à une notation simple et efficace.

Indépendamment de ces avantages indéniables, la ligne de commande permet de réaliser simplement et rapidement[3] des opérations qui serait autrement très fastidieuses et répétitives, voire impossibles, comme :

- créer une sauvegarde de tous vos fichiers modifiés cette semaine ;
- extraire la première page d'une série de fichiers PDF pour en faire des images ;
- redimensionner une série d'images et leur ajouter une notice de copyright pour les diffuser sur une page Web ;
- classer des dossiers en fonction du nombre de chansons, d'images ou de documents qu'il contiennent ;
- télécharger d'un coup les cibles de tous les liens contenus dans un message électronique (ou n'importe quel autre texte) ;
- renommer rapidement les fichiers d'un dossier selon une règle simple.

Les possibilités sont illimitées : imaginez l'effet de passer d'une communication rudimentaire à un vrai discours élaboré !

Par ailleurs, l'utilisation de la ligne de commande est indispensable dans certains cas, en particulier lors de connexions distantes à un autre ordinateur. S'il est possible d'utiliser des interfaces graphiques à distance grâce à des protocoles spécialement étudiés, ces derniers sont par essence plus gourmands en ressources réseau[4]. Pour une connexion lente, le dialogue est nécessairement plus fluide en ligne de commande.

1.3 À qui s'adresse ce livre ?

Ce livre s'adresse à tous ceux qui se sentent parfois limités et frustrés par les interfaces graphiques, et qui aimeraient découvrir une autre manière d'utiliser leur ordinateur. Il s'adresse aussi aux utilisateurs réguliers de la ligne

3. Il s'agit de votre temps, pas de celui que l'ordinateur va y passer, mais ce dernier ne connaît ni l'ennui, ni la rébellion.

4. De plus, ces protocoles ne sont pas forcément autorisés par l'hôte distant.

de commande qui sentent qu'ils n'exploitent pas au maximum les possibilités offertes par cette dernière[5]. Enfin, il y a fort à parier que même les gourous de la ligne de commande y trouveront des idées intéressantes !

1.4 Les atouts de la ligne de commande

Il est vraisemblable qu'apprivoiser la ligne de commande changera définitivement votre perception des ordinateurs. Pour vous aider à faire ce pas, voici quelques-uns des avantages dont vous bénéficierez :

Rapidité : avec un peu d'habitude, vous vous rendrez compte qu'en réalité, il est toujours plus rapide d'utiliser la ligne de commande qu'une interface graphique fournissant des fonctionnalités équivalentes.

Expressivité : le langage de la ligne de commande permet d'exprimer efficacement des idées complexes de manière compacte. Si les ordinateurs ont du mal à nous comprendre directement, il nous est en revanche très facile d'apprendre un langage qu'ils comprennent.

Automatisation : si l'on sait effectuer une opération une fois, il est très simple de la réaliser un grand nombre de fois sans avoir à le faire manuellement. C'est l'ordinateur qui doit travailler pour vous et non le contraire !

Personnalisation/optimisation : il est très facile de créer des « raccourcis » pour les opérations que l'on effectue couramment : pas besoin de taper l'intégralité des commandes à chaque fois. Avec le temps, vous pourrez ainsi bâtir un environnement spécialement adapté à vos besoins.

Puissance : la ligne de commande permet de combiner efficacement des opérations élémentaires pour réaliser des opérations complexes. L'interpréteur de commandes est en réalité un véritable langage de programmation se prête facilement à l'écriture de programmes élaborés. Par ailleurs, il existe de nombreux programmes utilisables en ligne de commande pour réaliser toutes sortes de tâches, comme l'envoi automatique de courriers électroniques, la manipulation d'images, de vidéos, de données scientifiques...

Coexistence avec les interfaces graphiques : rien ne vous oblige à abandonner votre interface graphique pour passer à la ligne de commande. Les deux peuvent en effet coexister de manière pacifique sur votre ordinateur. Mieux, ils peuvent même coopérer : de nombreux programmes graphiques fournissent des moyens simples de les piloter en ligne de commande. On peut ainsi lancer l'ouverture d'une série de documents ou de pages Web, choisir les prochaines chansons que l'on va écouter...

Rentabilité : l'apprentissage de la ligne de commande est un investissement à long terme – s'il est vrai les premiers essais seront longs et laborieux par rapport à l'utilisation de votre interface graphique favorite, cette tendance va peu à peu s'inverser, et vous vous demanderez bientôt comment vous avez pu supporter de perdre autant de temps avec une interface graphique !

5. C'est le cas par exemple si vous ne savez pas comment réaliser toutes les opérations mentionnées page précédente.

1.5 Qu'entend-on exactement par « ligne de commande » ?

La ligne de commande est constituée de plusieurs éléments intimement liés. Au cœur se trouve l'interpréteur de commandes, que l'on appelle couramment SHELL, comme en anglais. C'est lui qui déchiffre ce que vous tapez et se charge de lancer les programmes correspondants ; c'est donc lui qui est déterminant pour le confort de l'utilisation de la ligne de commande. Les trois shells les plus couramment utilisés sont `bash`, `tcsh` et `zsh`[6]. Ces shells sont dotés de fonctionnalités avancées : édition, historique, complétion automatique... Nous centrerons notre propos sur `bash`, qui est le shell par défaut sur toutes les distributions Linux, ainsi que sur Mac OS X[7]. Les autres shells n'en diffèrent que par des détails, que nous avons rassemblés dans une annexe page 160.

Le shell traite vos commandes en faisant appel à des utilitaires de base présents dans tous les environnements de type Unix, définis par la norme POSIX : Linux, Mac OS X, toutes les variantes d'Unix (HP-UX, IRIX, Solaris, etc.), et même sous Windows si l'on installe la suite d'applications Cygwin[8]. Ces outils permettent des manipulations de fichiers et de textes. Combinés au shell, ils forment un véritable langage de programmation.

À cela s'ajoutent les programmes installés sur votre ordinateur, qui peuvent tous être invoqués en ligne de commande. Même si ceux-ci ne font pas partie des utilitaires standard, il peuvent révéler une puissance insoupçonnée lorsqu'ils sont combinés avec la ligne de commande.

La ligne de commande et les logiciels libres

Un grand nombre des logiciels qui constituent la ligne de commande, comme les shells et les utilitaires de base, sont des programmes développés selon le modèle des LOGICIELS LIBRES : leur code source est disponible pour tous, et ils sont souvent distribués gratuitement. Pour une part importante, les contributeurs de ces projets sont des programmeurs passionnés bénévoles ; ceux-ci ont une certaine tendance à préférer ajouter de nouvelles fonctionnalités à leurs logiciels plutôt que passer du temps à en peaufiner la documentation ou à le traduire dans d'autres langues. Gardez ceci à l'esprit lors de vos pérégrinations dans la ligne de commande !

1.6 Les objectifs de ce livre

Apprivoiser la ligne de commande demande de repenser sa manière d'interagir avec l'ordinateur. Pour faciliter cet apprentissage, nous avons organisé ce livre en chapitres de complexité croissante, depuis les rudiments jusqu'aux incantations avancées. Comme pour l'apprentissage d'une langue vivante, il est indispensable d'appliquer ses nouveaux acquis à des cas concrets pour progres-

6. Dans cet ordre, d'après les statistiques d'utilisation rassemblées sur `popcon.debian.org`.
7. Les premières versions de Mac OS X utilisaient `tcsh` comme shell par défaut.
8. Que l'on peut télécharger gratuitement à l'adresse `www.cygwin.com`.

ser. C'est pourquoi nous vous recommandons de consacrer un peu de temps à la mise en pratique de chaque chapitre avant de passer au suivant.

Le chapitre 2, « Kit de survie », est un passage obligé : dans la suite, nous supposerons que ce qui s'y trouve est acquis. Les chapitres 3 à 5 sont de difficultés comparables et peuvent être lus indépendamment. Les chapitres 6 et 7, « Les outils du sorcier » et « Lire et écrire des programmes », supposent une certaine familiarité avec les concepts et les commandes des chapitres précédents.

Vous trouverez à la fin de cet ouvrage un récapitulatif des conseils les plus précieux et des erreurs les plus courantes. Un index vous permet de retrouver facilement les concepts et les exemples d'utilisation des divers programmes présentés dans ce livre, et un lexique contient la définition des termes les plus importants ; ceux-ci sont indiqués au fil du texte en PETITES CAPITALES. De plus, nous avons préparé un tableau récapitulatif des commandes utilisées dans ce livre page 9 ; si un exemple fait appel à une commande qui vous est inconnue, c'est là qu'il faut la chercher.

Nous avons enfin isolé trois annexes : une où l'on explique comment trouver de l'aide, une autre sur les shells les plus couramment utilisés et leurs principales différences, et une dernière dans laquelle nous présentons quelques rudiments d'administration système, et en particulier les méthodes pour installer les programmes qui seraient absents de votre système.

Avertissement

Les logiciels qui forment la ligne de commande existent en de nombreuses versions offrant des fonctionnalités différentes. Il est donc possible que les programmes que vous utilisez ne se comportent pas exactement comme ce qui est décrit ici, en particulier si vous utilisez Mac OS X. Les cas gênants seront indiqués par des notes de bas de page.

Chaque chapitre de ce livre contient des exercices pour mettre en pratique vos acquis. Parmi ceux-ci, les mots croisés nécessitent une introduction particulière : il s'agit en réalité de « commandes croisées » et, en conséquence, ils peuvent contenir des espaces.

Bonne lecture, et bonnes commandes !

Kit de survie

Chapitre 2

Objectifs

- ▹ Prendre en main le shell
- ▹ Observer : naviguer dans la hiérarchie, lire et rechercher des fichiers
- ▹ Modifier : déplacer, copier et supprimer fichiers et répertoires

Les outils de la ligne de commande sont pré-installés sur toutes les machines Linux, Mac OS X ou Unix ; nous allons d'abord vous montrer où les trouver sur votre ordinateur, puis comment les utiliser pour les opérations de base.

2.1 Lancer une commande

a. Ouvrir un terminal

La ligne de commande fonctionne uniquement en « mode texte » : on tape du texte et l'ordinateur répond avec du texte. Pour l'utiliser, il est indispensable d'ouvrir un TERMINAL, ou CONSOLE, c'est-à-dire un programme qui permet cette interaction au sein d'un environnement graphique : il dirige les entrées au clavier vers le shell et affiche les réponses.

Comment lancer un terminal	
GNOME	cliquez sur « Terminal » () du menu « Accessoires »
KDE	cliquez sur « Système/Konsole » ( ou) du menu K.
Mac OS X	ouvrez l'application « Applications/Utilitaires/Terminal »
En général	cherchez à ouvrir une application nommée « XTerm », « ATerm », « ETerm », « Terminal », « Konsole », etc.

b. L'invite

Le terminal s'ouvre. Il se présente sous la forme d'un grand rectangle ne contenant, au début, qu'un petit texte en haut à gauche, que l'on appelle l'INVITE, ou PROMPT en anglais. Ce texte signifie « je suis prêt, j'attends une commande ». Sa teneur exacte varie d'un shell à un autre et d'une configuration à une autre. Parmi les plus courants, on peut citer « ~ », « # », « % », « > », « $ », « `totoro ~` », où `totoro` est le nom de l'ordinateur.

Quelle est la différence entre shell et terminal ?

Le terminal se borne à permettre l'interaction avec le shell en lui transmettant les entrées clavier de l'utilisateur et en affichant ses réponses. À son ouverture, il lance le shell favori[1] de l'utilisateur. C'est ce dernier qui se charge d'interpréter les commandes et de les exécuter.

Commençons par déterminer le shell utilisé par votre terminal. Pour cela, tapez le texte « `echo $0` »[2] à côté de l'invite, représentée ici par %. Appuyez alors sur la touche Entrée pour lancer la commande :

```
% echo $0      La commande
bash           La réponse
%      Tant que l'invite n'a pas reparu, la commande n'est pas encore finie !
```

Une ligne s'affiche (le résultat de la commande, ici `bash`) suivie par une nouvelle invite qui indique que la commande précédente est terminée et que le shell est prêt à en exécuter une nouvelle. Selon votre configuration, cette commande répond `bash`, `zsh`, `tcsh`, etc. Il s'agit du nom de votre interpréteur de commandes. En ce qui concerne les chapitres 2 à 4, tous les shells sont équivalents ; pour la suite, reportez-vous à l'annexe page 160 si ce n'est pas `bash`.

Utilisez les touches usuelles pour éditer la ligne de commande en cours : flèches gauche (←) et droite (→), touches de suppression (Retour Arrière et Suppr), début (Orig) et fin (Fin) de ligne. Nous présenterons d'autres raccourcis très pratiques page 42.

Majuscules/minuscules

Le shell fait systématiquement la différence entre majuscules et minuscules, que ce soit dans le nom des commandes ou des fichiers[3]. Veillez à bien respecter la casse des caractères !

2.2 Un exemple de commande : `ls`

Commençons notre exploration des commandes par « `ls` », qui est l'abréviation de *list* (« faire l'inventaire de », en anglais). Cette commande affiche le contenu d'un RÉPERTOIRE (ou « DOSSIER »). Comme elle ne modifie aucun fichier, son utilisation est sans risques : une fausse manœuvre est sans conséquences.

a. Arguments et options

Voici plusieurs façons d'utiliser `ls` :

```
% ls                    Affiche le contenu du répertoire courant
Admin          Chansons      H&K     Programmation
```

1. Si votre shell par défaut ne vous convient pas, nous verrons page 100 comment en changer.
2. La commande `echo` se contente d'afficher du texte, nous la décrirons page 99. Par ailleurs, les `$NOM` sont des *variables d'environnement*, nous y reviendrons page 99. Enfin, la variable spécifique `$0` contient le nom du shell utilisé ; elle sera détaillée page 141. On aurait aussi pu utiliser `echo $SHELL`, qui affiche votre shell par défaut (voir page 100) ; cependant, ce dernier peut être différent de celui lancé par votre terminal.
3. Sauf Mac OS X, qui, en général, ne fait pas la différence entre minuscules et majuscules.

```
Affiche.jpeg  debian.html  LaTeX
article.pdf   doc.tar.gz   mail

% ls Admin     Affiche le contenu du répertoire Admin
Comptes.xml  Impots
```

`ls` affiche le contenu du répertoire courant (voir page 22) ou celui du répertoire donné sur la ligne de commande, son argument.

La plupart des commandes peuvent naturellement travailler sur plusieurs arguments en même temps :

```
% ls Admin LaTeX     Affiche le contenu des répertoires
Admin:               Admin et LaTeX
Comptes.xml  Impots
LaTeX:
utils.sty
```

Nous vous signalerons systématiquement quand ce n'est pas le cas.

Par défaut, `ls` n'affiche pas les fichiers dont le nom commence par un point car ces derniers sont par convention « cachés » dans le monde Unix : un programme se doit de les ignorer à moins qu'on lui demande explicitement le contraire. `ls` les affiche si on le lance de la façon suivante :

```
% ls -a
.              .bash_history  .emacs.d  Programmation
..             .bashrc        H&K       .ssh
Admin          Chansons       LaTeX
Affiche.jpeg   debian.html    .lesshst
article.pdf    doc.tar.gz     mail
Les fichiers commençant par . sont souvent des fichiers de configuration

% ls -a Admin                          . et .. sont deux répertoires
.  ..  Comptes.xml  Impots             particuliers, voir page 23.
```

`-a` (pour *all*, tout) est une option qui modifie le comportement de `ls` sans en changer l'objet : afficher le contenu du répertoire courant ou des répertoires passés en argument. Les options se différencient des arguments par le fait qu'elles commencent systématiquement par un ou deux tirets (`-`).

Récapitulons : arguments et options

Les commandes acceptent généralement un ou plusieurs arguments, séparés par des espaces :

```
% ls Admin LaTeX     Affiche le contenu des dossiers
                     Admin et LaTeX
```

Elles peuvent aussi accepter des options qui modifient leur comportement. Ces dernières commencent par un ou deux tirets.

```
% ls -a Admin LaTeX  Montre aussi les fichiers cachés
```

des éléments issus d'un autre programme ou de la sortie de commandes précédentes (comme `find` ou `locate`, voir pages 37 et 38). Il faut savoir que le texte sélectionné à la souris dans le terminal est automatiquement copié et peut être collé par un clic sur le bouton du milieu[66]. Par ailleurs, un double clic sur un mot le sélectionne en entier, et un triple clic sélectionne la ligne entière[67].

Résumé du chapitre

- On communique avec le shell par des commandes qui acceptent des arguments et des options (précédées d'un tiret).
- Les fichiers sont organisés selon une grande arborescence unique dont le répertoire « / » est la racine.
- `ls` affiche le contenu des répertoires.
- `mv`, `cp` et `rm` servent respectivement à déplacer, copier et effacer.
- Il existe deux grandes classes de fichiers, les fichiers binaires et les fichiers texte. On peut lire ces derniers grâce à `less` et les éditer avec `nano`.
- On cherche des fichiers globaux avec `locate` et des fichiers locaux avec `find`.
- `man commande` est souvent la réponse à bien des questions sur `commande`.
- La touche Tab complète automatiquement ce que l'on tape.

EXERCICES

2.1 Caractères spéciaux

Parmis les noms de fichiers suivants, lesquels contiennent des caractères spéciaux ?

```
UnNomLong          un_autre!     Laurel&Hardy.mpeg
mon_article.pdf    Deux Mots     ++-+_+.jpeg
Version[1].doc     $Des_sous$    chapitre-3.txt
```

2.2 Les mots pour le dire

On interagit avec le *shell* en tapant des ␣␣␣␣␣␣␣␣␣␣. Celles-ci acceptent des ␣␣␣␣␣␣␣␣␣␣ et des ␣␣␣␣␣␣␣␣ (qui commencent toujours par un tiret -). ␣␣ permet d'afficher le contenu des répertoires passés en argument. ␣␣, `cp` et ␣␣ permettent respectivement de déplacer, ␣␣␣␣␣␣ et supprimer des fichiers et des répertoires. Pour ces derniers, il est nécessaire de passer l'option ␣␣ à `cp` et ␣␣. Enfin, ces trois commandes acceptent une option ␣␣ pour demander confirmation avant d'écraser ou d'effacer un fichier.

66. Sur la molette, ou bien sur les boutons droite et gauche en même temps s'il n'y en a pas.
67. La définition exacte de « mot », en particulier en ce qui concerne les signes de ponctuation, varie d'un terminal à un autre et est généralement configurable.

2.3 Jouez vos jokers

Voici le contenu d'un répertoire (les symboles ♣, ♦, ♥et ♠ ne font pas partie des noms) :

```
% ls -F
compile           Fromages.tex~ ♣♦ Infos/ ♠
compile~ ♦        image-01.jpeg ♥  notes.txt
Fromages.jpeg ♣   image-02.jpg ♥   Sauvegardes/ ♠
Fromages.pdf ♣    image-03.JPG ♥   script-sauvegardes
Fromages.tex ♣    Image-04.jpeg ♥
```

Pour chacun des symboles ♣, ♦, ♥et ♠, écrivez un glob qui cible spécifiquement les fichiers et/ou répertoires correspondants.

2.4 Commandes croisées

Horizontalement

1. Lire des fichiers texte
2. Effacer un répertoire et tout son contenu
3. Trouver tous les fichiers du répertoire courant et de ses sous-répertoires qui finissent par ~
4. Trouver des fichiers dans l'ensemble du disque
5. Effacer un répertoire vide

Verticalement

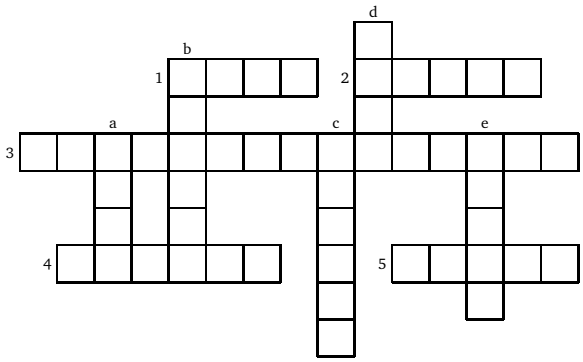
a. Éditer des fichiers texte

b. Afficher en détail le contenu du répertoire courant (dont les fichiers cachés)

c. Tout ce que vous voulez savoir sur `ls`, et plus

d. Afficher une hiérarchie de fichiers

e. « tous les fichiers PDF » (du répertoire courant)



SOLUTIONS

2.1

Sans caractères spéciaux		Avec caractères spéciaux	
`UnNomLong`	`mon_article.pdf`	`un_autre!`	`Laurel&Hardy.mpeg`
`++-+_+.jpeg`	`chapitre-3.txt`	`Deux Mots`	`Version[1].doc`
			`$Des_sous$`

2.2 On interagit avec le *shell* en tapant des *commandes*. Celles-ci acceptent des *arguments* et des *options* (qui commencent toujours par un tiret `-`). *`ls`* permet d'afficher le contenu des répertoires passés en argument. *`mv`*, `cp` et *`rm`* permettent respectivement de déplacer, *copier* et supprimer des fichiers et des répertoires. Pour ces derniers, il est nécessaire de passer l'option *`-R`* à `cp` et *`rm`*. Enfin, ces trois commandes acceptent une option *`-i`* pour demander confirmation avant d'écraser ou d'effacer un fichier.

2.3 Voici quelques globs répondants aux critères :

♣ : `Fromages.*` ou `F*`

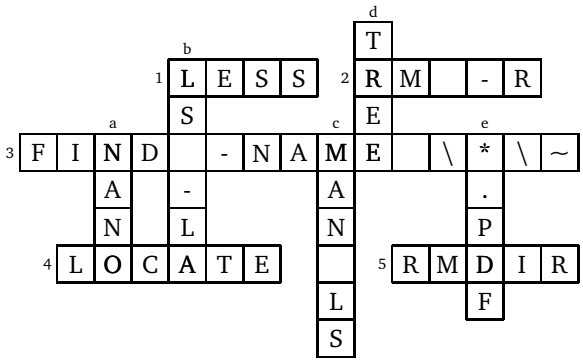
♦ : `*~` (les fichiers finissant par `~` sont souvent des copies de sauvegarde)

♥ : `i*.jpeg *.JPG *.jpg` ou `i*g *G` ou `*e-*`

♠ : `*/`

Cette liste n'est bien sûr pas exhaustive ; en revanche, elle montre qu'il y a souvent plusieurs manières d'écrire des globs. Entraînez-vous à trouver le plus compact !

2.4



Les outils de l'utilisateur

Objectifs

- ▹ Créer : des liens symboliques
- ▹ Agir : effectuer des sauvegardes, manipuler des documents
- ▹ Trouver : des fichiers ou des programmes
- ▹ Contrôler : les autorisations de fichiers

Maintenant que vous avez acquis les bases de la ligne de commande, nous allons aborder une série d'outils pour manipuler fichiers et répertoires : gestion des accès, archivage, manipulation de documents, recherche avancée de fichiers et de programmes.

3.1 Créez-vous des liens !

Il est souvent pratique de disposer de « raccourcis » pour des fichiers ou des répertoires. Si l'on prépare un congrès à Marseille dans un répertoire nommé `Travail/Congres/Marseille2012`, il est commode de pouvoir y accéder directement depuis le répertoire *home*, sans avoir à taper tout le chemin. On utilise dans ce but un LIEN SYMBOLIQUE (ou SYMLINK en anglais), que l'on crée ainsi, depuis le répertoire *home* :

```
% ln -s Travail/Congres/Marseille2012 Congres
```

N'oubliez pas le `-s` pour « symbolique » !

Le lien symbolique `Congres` est un raccourci vers le répertoire `Travail/Congres/Marseille2012`: travailler sur « `Congres` » revient à travailler sur `Travail/Congres/Marseille2012`. Ainsi,

```
% cd Congres
```

nous emmène en réalité dans `Travail/Congres/Marseille2012`.

La commande `ln -s`[1] s'utilise tant sur les fichiers que sur les répertoires, avec une syntaxe similaire à `mv`. Deux différences importantes : d'une part, le fichier ou répertoire original reste en place, inchangé ; d'autre part, il faut toujours lancer la commande depuis le répertoire où l'on veut placer le lien[2]. `ls -F` (voir page 23) indique les liens symboliques par un arobase (`@`) :

1. Le `-s` est important ; sans celui-ci, `ln` crée un autre type de lien, qualifié de « dur » ou « physique », similaire à première vue à un lien symbolique, mais avec des différences techniques qui dépassent le cadre de cet ouvrage.
2. En effet, lancer `ln -s a/b a/c` crée un lien `a/c` qui pointe vers `a/a/b` et non vers `a/b`.

```
% ls -F Congres
Congres@    @ pour un lien symbolique
% ls -l Congres
lrwxrwxrwx 1 vf vf 29 mai  8 11:09 Congres ->
↖ l pour lien          Travail/Congres/Marseille2012
```

Un lien symbolique est un fichier spécial qui contient un chemin, celui que l'on a passé en premier argument de `ln -s`. Quand on ouvre un lien symbolique, le système d'exploitation suit le chemin qu'il contient. S'il est absolu, il ouvre le fichier ou répertoire correspondant. S'il est relatif, il ouvre le fichier ou répertoire désigné par le chemin, *en partant du répertoire dans lequel le lien se trouve*. On peut lire le chemin contenu par le lien avec `readlink` :

```
% readlink Congres
Travail/Congres/Marseille2012
```

Supprimer ou déplacer un lien symbolique n'affecte que le lien (dans ce cas, `Congres`), pas la cible (ici, `Travail/Congres/Marseille2012`). Si la cible est supprimée ou déplacée, le lien est orphelin : il ne pointe vers rien. Essayer de l'utiliser aboutit à une erreur du type « fichier non trouvé ». C'est aussi généralement ce qu'il arrive lorsqu'on déplace un lien contenant un chemin relatif.

À quoi servent les liens symboliques ?

- Faciliter l'accès à des répertoires très hiérarchisés en fournissant un accès rapide à ceux que l'on utilise couramment.
- Donner plusieurs noms à un unique fichier ou répertoire.
- Utiliser un fichier ou un répertoire depuis plusieurs endroits.
- Éviter de copier le même fichier dans plusieurs répertoires.
- Centraliser la maintenance d'un programme, d'un fichier de style ou d'une bibliothèque utilisé à plusieurs endroits dans la hiérarchie.

3.2 Qui a le droit d'utiliser les fichiers ?

Dans les systèmes de type Unix/Linux, les actions des utilisateurs sur les fichiers (lecture, modification et exécution, s'il s'agit d'un programme) sont limités par un contrôle d'accès strict qui repose sur l'une des pierres angulaires de ce type de systèmes : la notion d'utilisateurs et de groupes (d'utilisateurs).

a. Utilisateurs et groupes

Les « groupes d'utilisateurs »[3] servent à donner des permissions spécifiques aux utilisateurs qui en font partie. Par exemple, les membres du groupe `audio`

2. C'est exactement celui que `ls -l` affiche après `->`.

3. La création et la modification des groupes est réservée au super-utilisateur : on utilise `addgroup` ou `groupadd` pour créer des groupes, et `adduser` ou `useradd` pour ajouter un utili-

Gérer le temps et le réseau

4 Chapitre

Objectifs

- ▹ Interagir : avec des serveurs (Web, SSH, FTP) distants
- ▹ Agir : planifier des tâches
- ▹ Contrôler : l'exécution de commandes

Dans ce chapitre, nous allons aborder quelques-unes des possibilités offertes par la ligne de commande pour voyager dans l'espace (interagir avec des ordinateurs distants) et dans le temps (le futur surtout, via la planification de tâches).

4.1 Sessions shell distantes

Un des atouts de la ligne de commande est qu'elle se prête facilement à une utilisation à distance, parce que les informations échangées se limitent au minimum : quelques lignes de texte.

a. SSH : un shell distant sûr

Le système SSH (pour *secure shell*, « shell (distant) sécurisé ») offre la possibilité de se connecter à une machine distante, d'y ouvrir un shell et d'y lancer des commandes comme sur une machine locale. Outre le côté pratique, ce système est sûr, car la connexion est chiffrée : ni les mots de passe, ni les données échangées ne peuvent être interceptés par un tiers[1]. On l'utilise ainsi :

```
% ssh vf@machine.distante.org        Mot de passe pour vf sur
vf@machine.distante.org password:    machine.distante.org
```

Cette commande ouvre une session shell de l'utilisateur `vf` sur le serveur SSH `machine.distante.org`. Pour que ceci fonctionne, il est nécessaire qu'un serveur SSH soit installé sur `machine.distante.org`, et bien sûr que vous y possédiez un compte au nom de `vf`[2]. Une fois votre mot de passe accepté, une nouvelle invite apparaît :

```
Linux machine 3.1.0 Description de la machine distante
Last login: Thu Nov 03 17:10:44 2011 from [...]
            Date et origine de la session shell précédente
%           Invite du shell de la machine distante
```

1. On utilisait auparavant `telnet` pour exécuter des commandes à distance, mais ce dernier n'est pas chiffré et inférieur en tous points à SSH.

2. Il n'est nul besoin d'avoir du matériel de « qualité serveur » pour abriter un serveur SSH ; il s'agit juste d'installer le logiciel approprié. Nous verrons dans l'encadré page 77 comment mettre en place des serveurs SSH sur un réseau domestique.

Cette nouvelle invite est affichée par le shell de la machine distante : les commandes suivantes tourneront sur celle-ci, jusqu'à ce que vous lanciez la commande `exit` pour terminer le shell distant et revenir au shell local[3].

La première fois que vous vous connectez à une machine distante, `ssh` vous signale qu'elle lui est inconnue :

```
The authenticity of host 'machine.distante.org'
  can't be established.
L'authenticité de machine.distante.org n'a pas pu être établie.
RSA key fingerprint is   L'empreinte de sa clé RSA est ...
  f9:5d:f4:d8:5c:d0:84:b2:6c:17:3b:d3:74:7c:e5:f6.
Are you sure you want to continue connecting (yes/no)?
Êtes-vous sûr de vouloir vous connecter (oui/non) ?
```

En effet, à chaque serveur SSH correspond une clé qui l'identifie de manière certaine. Ce message vous invite à vérifier que le serveur est bien celui que vous pensez[4] ; tapez « `yes` » (oui) pour accepter la clé. Si, lors d'une connexion ultérieure, la clé a changé, `ssh` vous le fait savoir et refuse catégoriquement de se connecter :

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
[...] La clé problématique est à la ligne 9 :
Offending key in /home/vf/.ssh/known_hosts:9
```

Ceci peut indiquer une tentative d'interception de la connexion, mais plus souvent, c'est la conséquence d'une mise à jour du serveur. Il faut dans ce cas éditer le fichier `~/.ssh/known_hosts` pour supprimer l'empreinte obsolète.

Astuce : ouvrir un fichier à une ligne précise avec `nano`

Pour supprimer la clé fautive, on peut exploiter le fait que `nano`, ainsi que `emacs`, `vim` et d'autres éditeurs de texte, se positionne directement à la ligne numéro 9 si on le lance de cette manière :

```
% nano +9 ~/.ssh/known_hosts
```

Une session `ssh` ordinaire ne permet pas de lancer de programmes graphiques. Si vous essayez, vous obtiendrez une erreur du type :

```
% display image.jpeg
display: unable to open X server ‘’.
```

3. On peut aussi utiliser `^D`, voir page 91. Tous deux fonctionnent aussi pour terminer un shell local. Si jamais le shell distant ne répondait plus, on peut terminer une session en appuyant sur Entrée puis les deux lettres « `~.` ».

4. En pratique, cela n'est pas souvent possible, à moins que la machine sur laquelle vous vous connectez publie une page Web présentant l'empreinte de sa clé (*fingerprint* en anglais).

Le shell à votre service

5 Chapitre

Objectifs

- Maîtriser : le shell au clavier
- Créer : des alias
- Configurer : les préférences du shell, les couleurs, etc.

Un des grands atouts de la ligne de commande est son côté extensible et malléable : il est facile de construire petit à petit un environnement adapté à ses besoins, que ce soit via l'utilisation de raccourcis clavier ou par la personnalisation du comportement du shell et de son apparence. Commençons par les raccourcis.

5.1 De `^A` à `^Z`

a. Raccourcis clavier courants

Les shells se pilotent via de nombreux raccourcis clavier. Nous allons vous présenter ceux qui sont communs à tous les shells — et qui sont aussi utilisables avec d'autres programmes. Ils sont pour la plupart basés sur la touche Ctrl, que l'on note `^` en abrégé : « `^A` » veut dire Ctrl suivi de la touche « a » (en minuscule).

Navigation	
`^A` ou Orig	début de la ligne
`^E` ou Fin	fin de la ligne
`^F` ou →	caractère suivant
`^B` ou ←	caractère précédent
`^P` ou ↑	remonter dans l'historique (commandes plus anciennes)
`^N` ou ↓	descendre dans l'historique (commandes plus récentes)

Édition	
`^D` ou Suppr	supprime le caractère sur le curseur[1]
`^H` ou Retour Arrière	supprime le caractère à gauche du curseur
`^W`	« coupe » le mot à gauche du curseur[2]
`^K`	« coupe » du curseur jusqu'à la fin de la ligne[2]
`^U`	« coupe » du début de la ligne jusqu'au curseur[2]
`^Y`	« colle » ce qui vient d'être coupé par `^K`, `^W` ou `^U`[2]

1. Si aucun caractère n'est présent, `^D` a une autre fonction, voir page suivante.

S'il peut sembler redondant d'apprendre les raccourcis en Ctrl + une lettre quand les mêmes fonctions sont accessibles par des touches plus intuitives, ceci est tout de même une bonne idée, pour les raisons suivantes :

– Les puristes diront qu'il est plus rapide de les utiliser que les flèches et autres, car on n'a pas besoin de déplacer les mains.

– Il arrive que les flèches refusent de fonctionner, sur un terminal distant inhabituel, ou après un `zcat` intempestif sur un fichier binaire (voir encadré page 59). Dans ces cas, les raccourcis basés sur Ctrl ont plus de chances de fonctionner : mieux vaut avoir plus d'une corde à son arc !

À éviter !	
`^S`	gèle le terminal jusqu'à l'appui sur `^Q`

La touche `^S` a pour effet de geler complètement le terminal, ce qui peut parfois laisser penser qu'il a « planté ». Il n'en est rien ; il suffit d'appuyer sur `^Q` pour lui rendre vie. Si un terminal semble vous ignorer complètement, c'est peut-être parce que vous avez appuyé par inadvertance sur `^S`. Dans le doute, appuyer sur `^Q` ne peut pas faire de mal.

Autres actions	
`^I` ou Tab	complétion automatique (page 42)
`^J`, `^M` ou Entrée	lance la commande
`^L`	efface l'écran
`^G`	efface la commande en cours d'édition
`^D`	« fin de transmission »

`^D` signale aux programmes qui attendent une entrée de votre part (comme `at` ou `mail` par exemple) que vous avez fini. S'il est utilisé dans le shell lorsque la ligne est vide, le shell se termine, et le terminal se ferme (sauf dans le cas de sessions SSH, où seul le shell distant est terminé). `^G` peut être utilisé pour tout annuler quand on se trompe dans l'édition d'une commande de manière difficile à récupérer, comme un guillemet pas fermé.

b. Gestion des commandes

Le shell permet de gérer des commandes en cours d'exécution (ou encore des « tâches ») via des raccourcis clavier[3] :

2. Il ne s'agit pas du couper/coller classique : celui-ci est interne au shell et ne permet pas de dialoguer avec d'autres applications de la session graphique en cours. Utilisez la souris pour cela, mais sachez qu'un texte sélectionné à la souris peut être collé dans un terminal avec Shift + Ins..

3. Ces raccourcis ne fonctionnent que si le terminal est la fenêtre active, ce qui est particulièrement important si l'on a lancé un programme graphique depuis la ligne de commande.

Les outils du sorcier

6 Chapitre

Objectifs

- Combiner des programmes
- Agir en masse
- Contrôler les entrées/sorties des programmes

Nous avons décrit jusqu'à présent des outils utiles et performants, sans détailler leurs interactions potentielles. Les deux derniers chapitres de ce livre vont pallier ce manque et vous présenter ce qui fait toute la puissance de la ligne de commande : les mille et une manières de combiner des programmes simples en des tâches complexes, précises, et généralement difficiles à réaliser autrement. Nous avons passé les chapitres précédents à décrire les différents ingrédients de la ligne de commande, c'est maintenant que vous allez apprendre à les combiner pour créer vos propres recettes !

6.1 Rediriger les informations

Le shell permet de stocker l'entrée ou la sortie d'un programme en utilisant une REDIRECTION.

a. > : rediriger la sortie

Le symbole « > » redirige la sortie d'un programme dans un fichier, au lieu de l'afficher dans le terminal. Ainsi,

```
% ls Musique > mes_chansons.txt
```

crée un fichier `mes_chansons.txt` contenant la liste des fichiers présents dans le répertoire `Musique`, et n'affiche aucun texte dans le terminal[1]. Cette construction sert à stocker le résultat d'une commande sur lequel on doit travailler par la suite, ou bien pour l'envoyer à quelqu'un d'autre. On peut aussi user de cette possibilité pour créer des petits fichiers texte :

```
% echo texte > mon_fichier.txt
```

ou transformer une commande complexe en script[2] en la rappelant dans l'historique et en la préfixant de `echo`, comme[3] :

1. Sauf si `ls` affiche un message d'erreur, nous y reviendrons page 113.
2. Voir page 137.
3. Il faut entourer la commande de guillemets appropriés si elle contient des caractère spéciaux ou des guillemets, voir page 125.

```
% find -name '*.pdf' -mtime -10 -size +2M
Pour réutiliser cette commande, on la stocke dans « script »
% echo "find -name '*.pdf' -mtime -10 -size +2M" > script
```

Suivant la configuration du shell[4], si le fichier cible existe, soit la redirection > l'écrase sans confirmation, soit elle donne une erreur[5], auquel cas la commande n'est pas exécutée. Indépendamment de sa configuration, on peut forcer le shell à écraser le fichier cible en utilisant >| en lieu et place de >[6].

Il est parfois désirable d'ajouter le résultat de la commande à la fin du fichier en question plutôt que d'en remplacer le contenu ; on utilise pour ceci >>. Pour compléter le fichier `mes_chansons.txt` avec le contenu d'un autre répertoire, on pourrait ainsi lancer :

```
% ls AutreMusique >> mes_chansons.txt
```

`cat` : concaténer des fichiers

`cat` affiche dans le terminal tous les fichiers qui lui sont donnés en argument, les uns à la suite des autres. Ceci permet de concaténer des fichiers, c'est-à-dire de les fusionner en un seul :

```
% cat chapitre_*.txt > livre.txt
```

Cette commande construit le fichier `livre.txt` en ajoutant les fichiers `chapitre_*.txt` les uns à la suite des autres.

b. < : rediriger l'entrée

« < » est le pendant de « > » pour préparer à l'avance l'entrée d'une commande. Par exemple, plutôt que de taper la liste des commandes à planifier directement dans `at`, il est plus confortable de préparer à l'avance avec un éditeur de texte[7] un fichier contenant les commandes, par exemple :

```
tar cvzf docs-‘date +%Y-%m-%d‘.tgz docs
scp docs-‘date +%Y-%m-%d‘.tgz serveur.org:archives
```

et de lancer `at` de la manière suivante :

```
% at now + 4 hours < commandes.txt Sauvegarde dans 4 heures
```

Un programme avec lequel il est fréquent d'utiliser ce genre de redirections est `mail`, qui permet comme son nom l'indique d'envoyer des courriers électroniques :

```
% mail -s 'Bonjour !' h-k@example.com < message.txt
```

4. Voir `set -C` page 107 pour changer le comportement par défaut de votre shell.
5. `bash : mon_fichier.txt : impossible d'écraser le fichier existant.`
6. Vous pouvez aussi utiliser `rm` pour enlever la cible au préalable.
7. Il existe une redirection <<, nommée *here document* (littéralement document ici), qui permet de profiter des facilités d'édition du shell pour préparer l'entrée d'une commande — une sorte d'intermédiaire entre l'entrée directe et l'utilisation d'un éditeur.

Lire et écrire des programmes

Chapitre 7

Objectifs

- Automatiser toutes vos tâches répétitives
- Comprendre les scripts écrits par d'autres personnes
- Maîtriser l'utilisation des variables
- Et surtout : ne plus oublier de fêter les anniversaires !

En plus de son utilisation pratique au quotidien, le shell cache en son sein un véritable langage de programmation qui vous permettra d'écrire des SCRIPTS, c'est-à-dire des petits programmes. Ces scripts augmenteront notablement votre efficacité et le confort d'utilisation de votre ordinateur en automatisant la plupart de vos tâches répétitives.

Pour illustrer les concepts présentés dans ce chapitre, nous construirons pas à pas un script, `latex2pdf`, qui gagnera en généralité et s'enrichira de fonctionnalités au fur et à mesure de l'avancée dans le chapitre.

7.1 Écrire un programme

Un programme « en shell », ou « script shell », ou seulement « script », est un simple fichier texte (que l'on crée avec n'importe quel éditeur de texte comme `emacs`, `nano` ou `gedit`), qui contient des commandes, à raison d'une par ligne :

```
latex doc.tex && \
dvips doc.dvi -o && \
ps2pdf doc.ps
```

Notez l'absence de prompt : ceci n'est pas tapé depuis le shell, mais dans un éditeur de texte.

Lorsque les lignes sont trop longues, ou, comme c'est le cas ici, lorsque cela facilite la lecture, on peut séparer une commande en plusieurs lignes se finissant par des « \ »[1]. Ce script complète l'exemple de la page 114 pour réaliser la compilation en trois étapes du document LaTeX `doc.tex` en un fichier PDF. Les scripts évolués naissent souvent de scripts spécifiques comme celui-ci ; nous verrons dans ce chapitre comment étendre leurs fonctionnalités pour gagner en généralité et offrir plus de flexibilité.

On exécute ce script, que l'on nomme `latex2pdf` (même s'il ne mérite pas encore ce nom), de la façon suivante :

```
% sh latex2pdf
```

Il faut donner le chemin si latex2pdf n'est pas dans le répertoire courant

1. Pour que ceci fonctionne, il faut que \ soit le *dernier* caractère sur la ligne.

On utilise le plus souvent le shell `/bin/sh` (qui est en général un lien symbolique vers `/bin/bash`) pour intepréter les scripts, car des programmes qui fonctionnent avec lui ont plus de chances de marcher sur un autre ordinateur. Cela dit, rien n'empêche d'utiliser un autre shell comme `bash` ou `zsh`, si l'on a besoin de fonctionnalités spécifiques à ces derniers. Pour se passer de l'invocation explicite du shell et utiliser `latex2pdf` comme une vraie commande, il faut le rendre exécutable (avec `chmod a+x latex2pdf`), le placer dans un répertoire présent dans le `$PATH` (comme `~/bin` si on a pris le soin de l'inclure dans le `$PATH`, voir pages 101 et 104), et rajouter une ligne au début du fichier qu'on appelle SHEBANG et qui indique au système d'exploitation quel shell il faut utiliser pour interpréter le programme :

```
#! /bin/sh      L'espace entre ! et / est optionnel
```

Ces modifications réalisées, on peut alors exécuter ce script comme on le ferait pour n'importe quelle autre commande :

```
% latex2pdf
```

La seconde amélioration de ce script est d'utiliser `set -e` (voir page 106) qui force le shell à interrompre l'exécution du script à la première commande qui retourne une erreur, ce qui rend l'utilisation de `&&` inutile (parce que implicite). Utilisez `set -e` de manière systématique, car les commandes d'un script sont en général interdépendentes, et il n'est pas utile de continuer si l'une d'elles ne s'est pas déroulée correctement.

```
#! /bin/sh
set -e
latex doc.tex
dvips doc.dvi -o
ps2pdf doc.ps
```

Puisqu'il ne change pas, nous allons omettre le shebang des autres programmes présentés dans ce chapitre. Ne l'oubliez pas pour autant ! Nous ferons de même pour la ligne `set -e`.

Toutes les constructions utilisables depuis la ligne de commande le sont aussi depuis les scripts, et vice versa, à une exception notable près : puisque les shells non-interactifs ne lisent en général pas leurs fichiers de configuration, aucun alias n'y est défini[2].

Comme dans les fichiers de configuration, on peut rajouter des commentaires dans les programmes en les faisant précéder d'un signe `#`, comme ceci :

```
# une ligne de commentaires
```

On peut aussi les utiliser en fin de ligne :

```
ps2pdf doc.ps # transformation finale en PDF
```

Pour faciliter la relecture et la modification des programmes, il est particulièrement recommandé de bien les commenter.

2. Par contre, vos variables d'environnement personnelles sont accessibles, car c'est le shell depuis lequel vous lancez le script qui les met en place, et non celui qui interprète le script.

7 conseils

① **Utilisez les raccourcis** La complétion automatique et l'historique font gagner du temps.

② **Utilisez des alias** Ils permettent de gagner du temps et surtout de bénéficier sur le long terme, sans effort de mémorisation, de l'expérience acquise au quotidien, par exemple en utilisant systématiquement les options que vous jugez utiles.

③ **Agissez en masse** Pour des actions sur plusieurs fichiers, efforcez-vous d'utiliser `xargs` ou des boucles `for`. Avec l'habitude, vous gagnerez énormément de temps. L'intervention manuelle fichier par fichier ne doit être qu'un ultime recours.

④ **Commentez** Ajoutez autant que possible des commentaires dans vos scripts et surtout dans vos fichiers de configuration, cela évite de perdre beaucoup de temps quand il faut les modifier.

⑤ **Essayez** La ligne de commande trouve tout son intérêt quand on arrive à combiner des actions simples pour réaliser une tâche complexe — mais ceci n'est possible qu'en vous entraînant : efforcez-vous d'utiliser des pipes dès que c'est possible.

⑥ **Écrivez des scripts** Ils vous permettront de mieux maîtriser tous les aspects de la ligne de commande. De plus, contrairement aux commandes, ils laissent une trace écrite : vous pourrez retrouver dans un an comment vous aviez fait !

⑦ **Rédigez des aide-mémoire** Écrivez dans un fichier les résultats de vos recherches dans les documentations : fonctionnalités insoupçonnées, combinaisons particulières de programmes ou d'options pour obtenir un résultat donné, etc. Vous ne perdrez pas de temps la prochaine fois ! Cerise sur le gâteau, l'envoi d'un aide-mémoire bien fourni à un néophyte dans le besoin est toujours bien reçu...

7 pièges à éviter

① **Attention aux caractères spéciaux** La plupart des erreurs des débutants en ligne de commande proviennent de caractères spéciaux employés tels quels, sans échappement. Consultez la liste page 20 si vous avez un doute.

② **Ne travaillez pas sans filet** Pour des opérations délicates sur un grand nombre de fichiers, utilisez d'abord `ls -l`, `echo`, l'option `-i` de `rm` ou `-n` de `rename` pour vérifier que vous allez bien faire ce que vous voulez, ou bien réalisez une archive au préalable pour pouvoir revenir à la case départ.

③ **Prudence** Ne lancez pas une commande « pour essayer » sans la comprendre, surtout si elle a été trouvée sur Internet. Les conséquences peuvent être immédiates et irréversibles.

④ **Globs et programmes de conversion** Beaucoup de programmes de conversion (comme `convert` ou `ps2pdf`) interprètent un second argument comme le nom du fichier cible. N'utilisez pas de globs directement avec eux, mais plutôt `xargs` en mode « un par un » ou des boucles `for`.

⑤ **Espaces et =** Dans les définitions des variables (exportées dans l'environnement ou non) et des alias, faites attention à ne mettre aucun espace autour du signe `=`.

⑥ **Testez vos scripts** Avant de lancer toute une série d'opérations « pour de vrai » via un script, assurez-vous via des `echo` bien choisis que les variables intérmédiaires pointent bien vers les fichiers cibles — y compris quand il y a des espaces dans les arguments.

⑦ **Les shells du super-utilisateur** Tous ces avertissements s'appliquent à la puissance dix dans le cas de shells avec des droits d'administrateur : non seulement vous pouvez perdre toutes vos données, mais vous pouvez aussi endommager sérieusement votre système voire l'effacer intégralement. La prudence est de mise !

Comment trouver de l'aide ?

La ligne de commande ne se laisse pas aisément découvrir : impossible de chercher dans des menus la fonctionnalité dont on a besoin. Voici cependant quelques astuces qui permettent en général d'obtenir des informations sur un programme (dont on aura trouvé le nom en suivant les conseils page 68), son fonctionnement, et les options qu'il accepte.

Page de manuel

Le premier réflexe à avoir est de chercher sa page de manuel, soit directement, soit en utilisant `apropos` :

```
% man rsync
% apropos backup
```

Documentation au format `info`

Si les commandes précédentes ne donnent pas d'informations, on a parfois accès à une documentation au format `info` en lançant la commande :

```
% info make
```

`info` est un système de documentation complémentaire à `man` ; il s'utilise de manière similaire aux navigateurs `lynx` et `w3m`.

Aide en ligne du programme

Si ceci échoue, on peut demander directement de l'aide au programme grâce aux options `--help` ou `-h`, voire en le lançant sans arguments :

```
% handbrake --help
% handbrake -h
% handbrake
```

Testez dans cet ordre

Attention, ces deux dernières commandes peuvent lancer des actions non voulues ; soyez prêt à taper `^C` pour arrêter le programme s'il ne redonne pas la main rapidement.

Autres documentation locales...

Des documentations au format texte, HTML ou PDF sont fréquemment installées dans les répertoires `/usr/doc`, `/usr/share/doc` ou éventuellement des variantes avec `/usr/local` ou `/opt` en lieu et place de `/usr`. Pour les trouver :

```
% locate mdadm | grep -i doc | less
```

... et distantes

Une autre possibilité est d'utiliser un moteur de recherche pour trouver la documentation du programme sur Internet. Pour cibler la recherche, il peut être utile de rajouter Linux et `documentation` dans la requête.

Installation de programmes

Nous allons décrire comment installer des logiciels qui ne seraient pas présents sur votre machine.

Devenir root

Avant toute chose, pour effectuer des tâches d'administration système, il faut acquérir les droits d'administrateur, c'est-à-dire ouvrir une session super-utilisateur, `root`. Il existe pour cela deux méthodes :

```
% sudo bash      ou bien « sudo -s »
Password:        Il faut donner son mot de passe, pas celui de root
```

ou bien :

```
% su
Password:        Il faut donner le mot de passe de l'utilisateur root
```

Les deux méthodes aboutissent à une session shell en tant que super-utilisateur, ce que l'on peut vérifier ainsi :

```
% whoami                    Qui suis-je ?
root                        Le super-utilisateur
```

Depuis ce shell, on peut effectuer diverses tâches d'administration : manipuler les utilisateurs et les groupes, gérer les services, etc., et, ce qui nous intéresse ici, installer des programmes.

Bien évidemment, aucune de ces deux méthodes ne fonctionne si vous n'avez pas soit des droits administrateur spécifiques pour la première[1], soit le mot de passe de `root` pour la seconde.

Attention !

Il faut être très prudent quand on passe `root`, une maladresse peut vite se transformer en catastrophe !

Installer avec son gestionnaire de paquets…

Toutes les distributions Linux modernes sont fournies avec un gestionnaire de paquet (*package manager* en anglais), qui permet d'installer, de désinstaller et de garder à jour une quantité impressionnante de logiciels. Ces gestionnaires fonctionnent tous approximativement de la même manière[2].

1. En particulier, il faut que le fichier être autorisé à lancer un shell, dans le fichier de configuration de `sudo` (qu'on édite avec `visudo`).

2. Si dans bien des cas, l'utilisation de gestionnaires de paquets est aussi simple que cette section le suggère, cela ne se passe malheureusement pas toujours aussi facilement ; cette annexe n'a pas pour vocation de se substituer à la documentation de votre gestionnaire de paquets.

Lexique

Ce lexique regroupe des explications sur tous les termes techniques rencontrés dans cet ouvrage. Au fil du texte, nous avons signalé les mots présents ici au moyen de PETITES CAPITALES. Nous conservons cette convention dans les définitions.

Alias : raccourci pour une commande, éventuellement munie d'OPTIONS et d'ARGUMENTS, défini par l'utilisateur. On le définit dans le FICHIER DE CONFIGURATION DU SHELL. *Voir aussi p. 95*

Archive : manière de regrouper plusieurs fichiers et répertoires en un fichier unique, généralement compressé, pour en faciliter l'échange ou la sauvegarde. *Voir aussi p. 60*

Arguments : liste d'objets (généralement des fichiers) séparés par des espaces sur lesquels une commande doit travailler. *Voir aussi p. 19*

Autorisations : une série de permissions associées à un fichier ou à un répertoire (lecture, écriture, exécution), sur lesquelles le système d'exploitation se base pour autoriser ou refuser l'accès à un utilisateur. *Voir aussi p. 48*

Bash : le SHELL le plus couramment utilisé, successeur de `sh` ; c'est l'archétype du shell « Bourne ». C'est l'abréviation de *Bourne again shell*. *Voir aussi p. 160*

Binaire : voir FICHIER BINAIRE.

Caractères spéciaux : caractères qui ont une signification particulière pour le SHELL et qui doivent être ÉCHAPPÉS si l'on souhaite les utiliser tels quels dans une commande. *Voir aussi p. 20*

Chaîne de caractères : données textuelles, sous forme d'une série de caractères. C'est ce que contiennent les VARIABLES.

Chemin absolu : chemin vers un fichier ou répertoire repéré par rapport à la racine ; il commence par `/`. On parle aussi de CHEMIN COMPLET. *Voir aussi p. 21*

Chemin complet : synonyme de CHEMIN ABSOLU.

Chemin relatif : chemin d'un fichier ou répertoire repéré par rapport au RÉPERTOIRE COURANT ; il ne commence pas par `/`. *Voir aussi p. 22*

Commentaire : ligne commençant par un dièse (`#`) dans un FICHIER DE CONFIGURATION DU SHELL ou un SCRIPT, qui est ignorée par le shell et qui permet de documenter ce qui suit. *Voir aussi p. 105*

Index