POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

TERMINAL ET INTERPRÉTEUR DE COMMANDES

25 SEPTEMBRE 2023

JÉRÔME COLLIN, CHARLES DE LAFONTAINE, MERIAM BEN RABIA
GIGL | Polytechnique Montréal

N.B. Le masculin est utilisé pour alléger le texte.

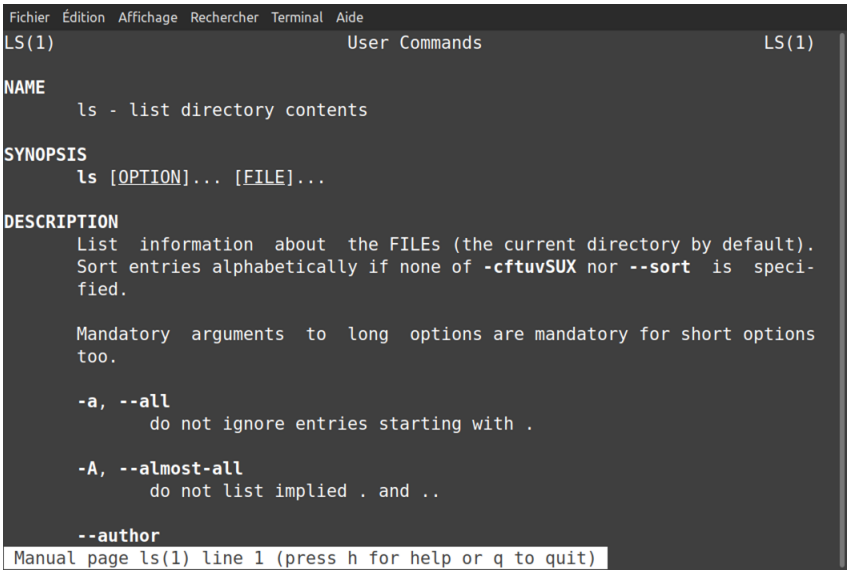
TABLE DES MATIÈRES

TERMINAL ET INTERPRÉTEUR DE COMMANDES

1. DÉFINITION ET BUT

Plus la programmation devient élaborée, plus le recours à la ligne de commande devient inévitable. Tout entrer ce qu'on veut que l'ordinateur fasse avec une souris a ses limites. Les commandes textuelles entrées au clavier sont plus puissantes, car on peut ajouter à ces commandes de nombreuses options (ou arguments) pour modifier les actions. De plus, il devient possible de rassembler ces commandes dans un *script* (ou fichier de commandes) pour automatiser une série d'actions sans recours à un utilisateur qui devrait autrement toujours refaire les mêmes opérations avec la souris. Plus particulièrement, sur les serveurs informatiques, il peut s'agir de l'unique façon de procéder, étant donné qu'il pourrait y avoir aucune interface graphique même si la machine effectue de nombreuses opérations complexes.

Il existe, en théorie, une différence entre un terminal (l'outil permettant d'entrer des commandes) et l'interpréteur de commandes (il y en a quelques-uns avec des syntaxes différentes). Cette distinction n'est pas vraiment importante pour débuter puisqu'avec le temps, sous *Linux*, presque tous utilisent *bash* comme interpréteur de commande avec le terminal fourni avec l'environnement graphique de la distribution utilisée. L'outil est toujours disponible dans l'équivalent *Linux* du menu *Windows* « Démarrer » sous la catégorie « Accessoires » (rechercher le nom « terminal »).



Exemple d'un terminal *Linux*

Sous *macOS*, on retrouve un interpréteur de commande pratiquement identique à celui sous *Linux*. *Windows* a aussi son interpréteur de commandes. Le plus vieux (encore présent) est souvent appelé *CMD*, mais le *PowerShell* est tout aussi intéressant et plus récent.

2. UTILISATION

Une fois le terminal ouvert dans l'environnement graphique du système d'exploitation, un interpréteur de commande affiche toujours un invité de commandes (connu en anglais sous le terme « *prompt* ») qui indique simplement qu'il est dans une position pour recevoir une commande d'un utilisateur.

Par convention, sous *Linux* ou *macOS*, les symboles `$` ou `%` sont utilisés comme invités. On peut aussi voir le symbole `#` quand la commande à entrer est pour le mode administrateur (*super-user* ou *root*). Sous *Windows*, l'équivalent est `>`. On n'a donc pas à taper cette partie au clavier pour entrer une commande puisque le terminal le fait pour l'utilisateur et indique simplement qu'il est prêt à exécuter une commande. Par contre, dans tous les documents traitant de lignes de commandes, **on a tendance à le placer dans les exemples**, ce qui peut engendrer un peu de confusion !

L'exemple si dessous montre que l'utilisateur a tapé la commande `ls` puisque le terminal lui avait donné l'invité `$` et le résultat de la commande est affiché en deuxième ligne et les suivantes s'il y a lieu :

```
$ ls
Bureau      Téléchargement      Photos      Vidéo      Musique
```

3. COMMANDES

Il y a de nombreuses commandes possibles. La création d'un programme avec un certain nom (aussi simple que « **`tp1.exe`** » par exemple) produit une commande supplémentaire qui peut être entrée au terminal comme n'importe quelle autre ! Cependant, au quotidien, on utilise une variété assez limitée de commandes et on effectue une recherche sur Internet pour les commandes plus spécialisées.

Voici une liste de commandes de base suffisantes pour démarrer et opérer efficacement au début :

Commandes	Actions	Arguments notables
`man`	Manuel du nom de la commande qui suit.	
`ls`	Liste les fichiers et répertoires.	**`-a, -l, -R, -F`**
`cat`	Affiche le contenu complet d'un fichier texte qui suit à l'écran.	**`-n`**
`more`	Semblable à la commande cat mais le résultat est affiché page par page.	
`pwd`	Affiche le chemin complet du répertoire courant.	
`mkdir`	Crée un répertoire vide ayant le nom qui suit sur la ligne de commande.	**`-p`**
`cd`	Changer de répertoire.	
`mv`	Pour déplacer un fichier ou un répertoire d'endroit ou pour changer son nom.	
`grep`	Permets d'afficher les lignes d'un fichier texte contenant une chaîne de caractères recherchée.	**`-A, -B`**
`find`	Pour chercher un fichier ayant certaines caractéristiques dans des répertoires.	Assez complexe…
`make`	Pour compiler du code lorsqu'un fichier *Makefile* est dans le répertoire courant.	
`git`	Pour utiliser l'outil de contrôle de version *Git* avec la ligne de commande.	

Quelques commandes *Linux* de base

Il existe plusieurs sites francophones pour apprendre les commandes de base sous *Linux*. En voici un, et un autre, mais vous pourrez en trouver par vous-même facilement. De même, une fois habile avec ces commandes de base, on pourra être tenté d'écrire un premier *script* sous *Linux* éventuellement…

4. INFORMATIONS COMPLÉMENTAIRES

Assez souvent, le résultat d'une commande se fait assez rapidement et l'usager est vite présenté avec la venue d'un nouvel invité de commande pour, justement, entrer une commande additionnelle à sa guise. Par contre, à l'exécution de certaines commandes, on peut se retrouver sans invité de commande avant un très long moment, ce qui prive l'usager de pouvoir exécuter de nouvelles commandes. Un exemple représentatif de cette situation peut être le démarrage d'un éditeur comme *VS Code* en ligne de commande. Généralement, on veut pouvoir conserver l'éditeur actif pendant plusieurs heures, ce qui fait que la commande reste active tout aussi longtemps. On peut, dans ce cas, envoyer la commande en arrière-plan (« *background* ») pour son exécution tout en retrouvant immédiatement un nouvel invité de commande. Pour arriver à passer la commande en arrière-plan, on a recours au symbole `&` à la fin de la commande. Ainsi, on peut démarrer *VS Code* en ligne de commande et en arrière-plan de cette façon :

```
$ code &
```

De même, il arrive qu'on puisse vouloir que le texte produit par une commande soit redirigé dans un fichier. C'est connu sous le nom de redirection et peut s'effectuer à l'aide de symbole `>` comme on peut le voir dans l'exemple suivant :

```
$ ls > listeDeFichiers.txt
$ cat listeDeFichiers.txt
Bureau    Téléchargement    Photos    Vidéo    Musique
```

Un peu dans la même veine, le résultat produit par une commande peut servir d'entrée à une seconde commande qui suit. On dit alors que les deux commandes sont reliées par une sorte de « tuyau » (*pipe*), représenté par une barre verticale. Un exemple pourrait être de rechercher où se situe le répertoire « **tp1** » à partir du répertoire courant en sachant qu'il se situe quelque part dans un sous-répertoire plus bas dans la hiérarchie. Comme la commande `ls` avec l'option `-R` listera tous les répertoires et fichiers sous-jacents, il pourrait s'avérer assez long d'effectuer la recherche visuelle nécessaire pour trouver l'élément recherché parmi ce qui est produit par la commande.

On peut donc filtrer avec la commande `grep` pour le terme recherché directement. La commande `grep` reçoit donc son entrée directement de la sortie de la commande `ls` et c'est donc la commande `grep` qui produit à l'écran le résultat final en sortie :

```
$ ls-R | grep tp1
. /cours/ing1001/tp/tp1
```

Une commande peut prendre du temps à s'exécuter et produire un résultat. On peut vouloir mettre fin à son exécution avant la fin en appuyant simultanément sur CTRL C .

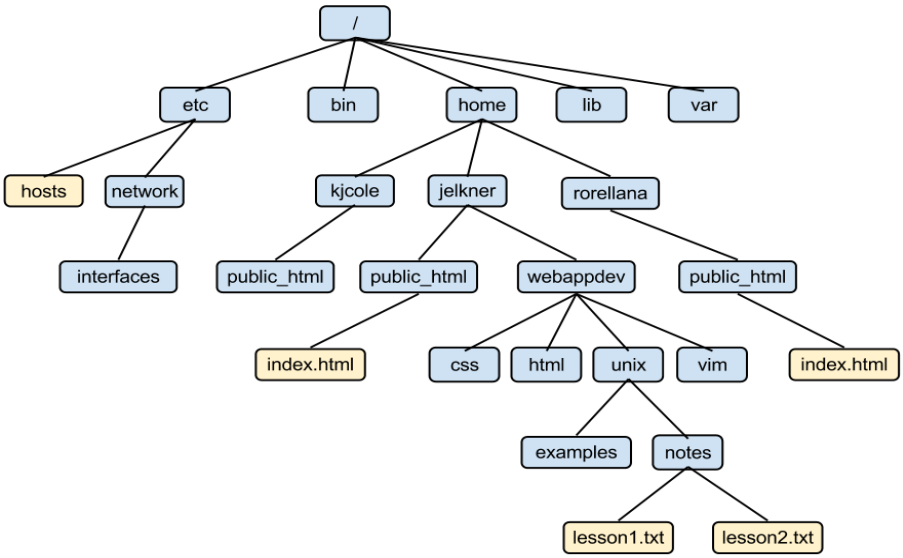
5. FICHIERS ET RÉPERTOIRES

Dans la présentation des différentes commandes, on remarque que la question des répertoires revient souvent. Ces commandes permettent de « naviguer » à travers des répertoires et fichiers sans l'aide d'un gestionnaire graphique de fichiers (« *file manager* »).

Un répertoire (« *directory* ») contient des fichiers ou d'autres répertoires. L'ensemble des répertoires en vient à former un système de fichiers (« *file system* ») ayant une racine. Sur *Linux*, le répertoire le plus haut se nomme souvent « *root* » et on le dénote par une barre oblique « **/** ». Sur *Windows*, ce même point est toujours associé à un support physique (clef *USB*, disque dur, etc.) associé à une lettre. C'est d'ailleurs cette lettre qui forme la première partie de l'invité de commande comme `C:\>`. Au contraire, sur *Linux*, il n'y a qu'une seule racine unique et un système de fichier présent sur une clef *USB* sera, par exemple, placé automatiquement par le système sous une branche principale du système de fichiers comme :

```
/media/usager1/USB_S
```

Sur *Linux*, comme on le voit dans l'exemple précédent, on sépare les répertoires par la barre oblique (« **/** », « *slash* »), comme c'est aussi le cas pour les chemins d'accès des pages d'un site Internet. À l'inverse, sur *Windows*, c'est la barre oblique inversée (« **** », « *back slash* ») qui est utilisée comme séparateur de répertoires. L'idée est toujours d'arriver à former une structure en arbre représentant les différents fichiers et répertoires :



Arbre qui donne vue sur divers fichiers (en jaune) et répertoires (en bleu) en partant de la racine

[https://www.openbookproject.net/tutorials/getdown/unix/lesson2.html]

Il y a toujours deux répertoires qui apparaissent toujours dans un répertoire (mais non sur la figure précédente) et qui portent toujours les mêmes noms « **.** » et « **..** » et qui désignent respectivement les répertoires courant et parent. Ajouter « **..** » à une commande permet toujours d'accéder au répertoire parent du présent répertoire, ce qui est fort utile. Il est peut-être un peu moins évident de voir l'utilité de passer « **.** » en argument à une commande, mais c'est souvent nécessaire.

6. CONSEILS UTILES POUR LA PROGRAMMATION

Typiquement, dans une organisation de répertoire contenant du code, on utilise des noms de fichiers et de répertoires courts, sans espaces et sans accents. On aura par exemple un nom de fichier ayant le nom « **`devoirEconomie.cpp`** » et non « **`Devoir Économie à Remettre.cpp`** ». Assez souvent, on ne place jamais une lettre majuscule au début du nom et on limite son utilisation à l'intérieur du nom.

Il y a plusieurs raisons à ces restrictions. La première vient du fait des restrictions imposées aux plus anciens systèmes d'exploitation qui ne supportaient pas nécessairement les accents ou les longs noms de fichiers. Aussi, il arrive que le code doive être écrit en anglais (surtout s'il est mis à disposition sur le web). De plus, bien que ce soit possible, la manipulation de noms de fichiers avec des espaces reste pénible avec la ligne de commande. Certains outils, plus anciens, peuvent ne pas supporter les espaces dans les noms de fichiers. Bref, on essaie d'éviter les problèmes et de conserver une lisibilité accrue dans les noms de répertoires et de fichiers, tout en conservant aussi une certaine efficacité.