Les commandes fondamentales de Linux

Introduction

Auteur : Riad MOKADEM
Dernière mise à jour : 07/06/2006
L'objectif de ce petit document est d'enseigner les commandes fondamentales de Linux (et donc d'Unix).

Table des matières

REMARQUE : SOUS LINUX (comme sous tout système UNIX) LES MINUSCULES ET LES MAJUSCULES NE SONT PAS ÉQUIVALENTES.

---

1. Commandes fondamentales

- Se déplacer dans l'arborescence de répertoires (cd)

Lorsque vous avez passé le login et le password de linux, vous vous retrouvez devant le prompt shell qui est le plus souvent celui de bash (sinon vous serez devant celui de csh). Il ressemble le plus souvent à ceci :

```
[root@mistra /root]$
```

Le mot **root** signifie que vous vous êtes "logué" sur le compte de l'administrateur système. Vous êtes donc en pleine possession de la machine, vous pouvez faire absolument n'importe quoi, jusqu'à supprimer tous les fichiers ... faites donc très attention ... En théorie *il ne faut utiliser la machine sous ce compte qu'afin de l'administrer*. Des comptes dits « d'utilisateurs » permettent sinon de travailler en temps normal. Nous verrons ci-après comment créer un compte utilisateur.

Le mot **"mistra"**représentera, dans ce document, le nom de votre ordinateur (pour le connaître invoquer la commande "hostname")

Actuellement vous vous trouvez sous le compte de l'administrateur système, c'est-à-dire que vous êtes dans le répertoire **/root** (sous Unix, les composants des noms de répertoires sont séparés par des "slash" "/" et non pas comme sous MS-DOS par des "anti-slash" "\").

Déplaçons-nous dans la "racine" du système :

```
[root@mistra /root]$ cd ..
```

Faites bien attention de séparer par un espace "cd" et "..", UNIX exige une grande précision dans la syntaxe des commandes. Soumettez la commande au système grâce à la touche « Entrée », évidemment !

Vous êtes maintenant dans le répertoire racine :

```
[root@mistra /]#
```

Que contient-il ? Tapez la commande **ls**, et voyez le résultat, vous devez obtenir quelque chose comme :
`bin boot cdrom etc usr var vmlinux` Si certains fichiers ou répertoires manquent ce n'est pas important.

Déplaçons-nous dans le répertoire qui contient une grande partie des programmes (souvent simplement appelés « binaires ») de linux : /usr/bin : **cd usr/bin**. Vous pouvez là aussi obtenir le contenu du répertoire en utilisant la commande **ls**.

Maintenant allons voir ce que contient le répertoire **/etc** (aperçu lorsque nous avons listé le répertoire racine /). Nous avons deux possibilités pour nous y rendre : soit nous revenons dans le répertoire racine et nous nous rendons ensuite dans le répertoire **etc**; soit nous nous rendons immédiatement dans le répertoire **/etc** :
- Méthode no 1 :
cd / (pour se rendre à la racine)
puis
cd etc

Cette méthode est fastidieuse car elle nécessite de taper deux commandes successives. Nous pouvons utiliser la deuxième méthode pour nous rendre directement dans le répertoire **/etc** en écrivant le chemin complet dans la commande **cd** :
- Méthode no 2 :
cd /etc

et nous sommes directement dans le répertoire **/etc**. Dans cette commande nous avons indiqué que pour se rendre dans le répertoire **etc**, il fallait d'abord se rendre dans le répertoire racine. Pour se faire nous avons placé un / devant **etc**.

Lorsque l'on ajoute un **~** au lieu d'un chemin à la commande **cd**, celle-ci nous replace automatiquement dans notre répertoire utilisateur. Si vous êtes en administrateur système la commande par **cd ~** vous placera dans le répertoire **/root**. Dans le cas où je suis (je suis loggé en tant qu'utilisateur *delcros*) je vais automatiquement me retrouver dans le répertoire de l'utilisateur *delcros* qui se trouve dans **/home/delcros**. Les répertoires des utilisateurs sont tous sous **/home**.

[delcros@mistra bin]$cd ~

- Ceci est la méthode orthodoxe, sinon vous pouvez faire simplement :
[delcros@mistra bin]$cd

et vous reviendrez ainsi dans votre répertoire personnel.

sommaire

Dans quel répertoire suis-je actuellement ? (pwd)

Lorsque l'on se déplace dans les répertoires, par défaut bash n'affiche que le « nom court » du répertoire où l'on se trouve. Le nom court ne comprend pas le chemin

complet. Or il peut arriver qu'un même nom court corresponde à plusieurs répertoires bien distincts, donc que seuls les chemins qui y mènent permettent de les distinguer. C'est par exemple le cas du nom court **bin**, que l'on trouve en **/bin** et en **/usr/local/bin**. Il existe beaucoup d'autres exemples. La solution pour connaître le chemin du répertoire où l'on se trouve est d'utiliser la commande **pwd** :

[delcros@mistra bin]$ pwd
```
/usr/bin
```
[delcros@mistra bin]$

sommaire

Lister les fichiers d'un répertoire (ls)

La commande **ls** et ses très nombreuses options vous permettront d'obtenir beaucoup d'informations sur les fichiers présents dans un répertoire : déplaçons nous par exemple dans le répertoire "/bin" et listons le contenu de ce répertoire :

[delcros@mistra bin]$ cd /bin
[delcros@mistra /bin]$ ls
```
arch            dd              gzip            nisdomainname   su
ash             df              hostname        ping            sync
awk             dmesg           kill            ps              tar
cp              fgrep           mount           sh
ypdomainname
cpio            gawk            mt              sleep           zcat
csh             grep            mv              sort            zsh
date            gunzip          netstat         stty            ls
```
Ceci est un listing "brut" du répertoire /bin qui contient les utilitaires de base de linux. On reconnaît par exemple la commande **ls** ...
De la même manière que sous MS-DOS (avec la commande dir), nous pouvons demander à Linux de lister seulement les fichiers dont les noms contiennent des caractères donnés. Demandons par exemple uniquement les noms des fichiers commençant par la lettre "l" :

[delcros@mistra /bin]$ ls l*
```
ln login ls
```
[delcros@mistra /bin]$

Voici quelques options intéressantes de la commande **ls** (les options sous UNIX suivent la commande et sont le plus souvent précédées d'un tiret) :
L'option **ls -l** permet de lister les attributs des fichiers (les droits de lecture, d'écriture et d'exécution, le propriétaire, le groupe, la taille en octets, sa date de création ou de modification) :

[delcros@mistra /bin]$ ls -l

```
total 3615
-rwxr-xr-x    1 root     root         2716 Apr 23  02:09  arch
-rwxr-xr-x    1 root     root        56380 Dec 23  1996   ash
lrwxrwxrwx    1 root     root            4 May 10  20:01  awk -> gawk
-rwxr-xr-x    1 root     root        18768 Mar  8  19:17  basename
-rwxr-xr-x    1 root     root       300668 Sep  4  1996   bash
lrwxrwxrwx    1 root     root            3 May 10  19:59  bsh -> ash
-rwxr-xr-x    1 root     root        16584 Dec 16  1996   cat
-rwxr-xr-x    1 root     root        17408 Nov 26  1996   chgrp
```

Notes : Ici, tous les fichiers appartiennent à l'administrateur système (root) et à son groupe (root), comme les sections consacrées à chmod et à chown l'exposerons). Nous traiterons du sens de la fin de chaque ligne, qui contient parfois une flèche visible ici sur la ligne `awk -> gawk`, dans la section consacrée aux liens ln.

ls -a liste tous les fichiers du répertoire, y compris les fichiers cachés. Cette option est très utile lorsque l'on se trouve dans son répertoire personnel car il contient les fichiers de configuration de l'utilisateur dont les noms commencent généralement par un point et seule l'option **-a** permet de détecter leur existence.

Exemple avec le répertoire de l'administrateur système :
voici une partie des fichiers listés avec la commande **ls** sans option :
[root@mistra /root]# ls

```
bookmarks.sgml     mc.hint            scrsh2             2494.html
Desktop            ftape.o            mc.hlp             scrsh3
FAQ.services.html  kbanner.kssrc      mc.lib             xdm-config
```

Et voici une partie du résultat avec la commande **ls -a**.
[**root@mistra /root]# ls -a**

```
.                               .kvtrc                          .xquadk
ey
..                              .letter                         .xquadk
ey~
.BitchX                         .mc.ext
2494.html
.Xmodmap~                       .peruser-newsrc-working
.amaya                          .peruser_config                 Desktop
.applications                   .peruser_spool
FAQ.services.html
...
```

On peut maintenant connaître tout (option 'a' : penser au mot "all") le contenu du répertoire.

D'autres options de **ls** sont utiles :
ls -m :
Affiche les fichiers en les séparant par une virgule au lieu de les présenter en colonnes.

ls -t :
Affiche les fichiers par date, c'est-à-dire en les classant du récent au plus ancien.

ls -lu :
Affiche les fichiers par date de dernier accès et indique cette date.

ls -F :
Affiche les fichiers par type. Ainsi un fichier suivi d'un slash (/) est un répertoire, un fichier suivi d'une étoile est un fichier exécutable et un fichier suivi d'un "@" est un lien (nous reviendrons sur les liens dans la section consacrée à **ln**).

ls -S :
Affiche les fichiers triés par ordre de taille décroissante.

ls -X :
Affiche les fichiers par type d'extension.

ls -r :
Affiche les fichier en ordre alphabétique inverse.
Cette option à la particularité d'inverser l'effet de tous les tris requis. Par exemple, la commande **ls -tr** affichera les fichiers par date en commençant par les plus anciens pour finir par les plus récents.

sommaire

Voir un fichier (cat et more)

La commande **cat** permet de lire des fichiers. Nous avons vu tout à l'heure que le répertoire **/root** contenait des fichiers de configuration. Ces fichiers sont simplement des fichiers textes avec un agencement et une syntaxe particulière. Regardons le contenu du fichier **.bashrc** qui permet de configurer à souhait son shell :

[root@mistra /root]# cat .bashrc
```
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
source .sd.sh
```
[root@mistra /root]#

Une option utile de **cat** est **-n** qui permet de numéroter les lignes (ne pas oublier que cat permet de *lire* et non de *modifier* un fichier. Ainsi la numérotation de ligne apparaît à l'écran mais le fichier .bashrc n'en est pas pour autant modifié).

[root@mistra /root]# cat -n .bashrc
```
1 # .bashrc
2
3 # User specific aliases and functions
4
5 # Source global definitions
6 if [ -f /etc/bashrc ]; then
7 . /etc/bashrc
8 fi
9 source .sd.sh
```
[root@mistra /root]#

Si vous souhaitez connaître les autres options de **cat**, tapez au prompt "**cat --help**".

Vous pouvez utiliser la commande **more** pour visualiser un fichier. La commande **more** a l'avantage d'afficher le fichier page par page. Pour passer d'une page à l'autre, tapez sur la touche **ESPACE**.

sommaire

- Éditer un fichier (vi, emacs, joe)

1. **vi (l'éditeur le plus ancien)**

   vi date des années 70 .... autant dire que cet éditeur a du métier et n'est toujours pas démodé. Ce n'est pas celui que j'utilise mais beaucoup en sont adeptes malgré son apparence fruste. Ceci s'explique par une grande puissance ... Si je m'attarde quelque peu sur **vi**, c'est que dans les moments critiques où rien ne fonctionne, où tout va mal, c'est l'éditeur qu'on ne peut éviter.
   Lançons Vi :
   [root@mistra /root]# vi

   Après le lancement de la commande vous allez vous trouver directement dans l'éditeur ... Pendant ce court apprentissage de vi, nous allons créer un fichier, le modifier, l'enregistrer, ... et quelques autres petites manoeuvres de survie :

   1. **Passer du mode commande aux mode texte, taper du mode texte, enregistrer.**

      vi comprend deux modes : un mode "commande" et un mode "insertion", après le lancement de vi nous sommes en mode commande : appuyez sur la touche "**Echap**" puis sur "**a**" ("a", comme "**a**ppend", permet d'ajouter du texte après le curseur). Vous voyez en bas de l'écran apparaître la ligne "-- INSERT --". Nous pouvons commencer notre texte :

      ```
      linux est gratuit puissant en perpetuelle
      evolution.
      Linux est stabble. Linux existe depuis 1991
      seulement
      et pourtant quel chemin parcouru !
      ```

      N'oubliez pas de placer retour chariot au bout de chaque ligne.

      Sauvons le fichier : nous sortons d'abord du mode texte en appuyant à nouveau sur la touche "**Echap**". La mention "-- INSERT --" disparaît, nous sommes en mode commande. Tapez maintenant " **:w linux-test**" et sur la touche retour chariot (afin d'écrire ("**w**rite") le fichier). Vous devez obtenir en bas de l'écran ceci :
      `"linux-test" [New File] 3 lines, 142 characters written`

2. **Supprimer du texte et quitter vi**
   Nous voyons qu'à la deuxième ligne, j'ai fait une grosse fôte d'aurtograffe. Nous allons supprimer le "b" qui est en trop dans stab*b*le : déplacez le curseur sur un des "b" en trop, passez en mode commande ("--INSERT --" ne doit pas apparaître à l'écran), appuyez sur "**x**", le b a disparu.

   Quittons vi, mais auparavant, nous devons sauver les modifications effectuées : Passez en mode commande et tapez " **:wq**" (**w**rite et **q**uit). Vous êtes sorti de vi et votre fichier a été sauvegardé sous linux-test. Pour revenir à vi en ouvrant le fichier linux-test au démarrage tapez :
   [root@mistra /root]# vi linux-test

   Si vous souhaitez quitter sans enregistrez les dernières modifications, il vous faudra passer en mode commande et taper " **: q!**".

   Ceci est une présentation très très courte de vi, mais qui vous permettra malgré tout de survivre au cas où vous devriez absolument l'utilisez. Voyons tout de même un rapide descriptif d'autres commandes vi :

3. **D'autres commandes vi.**

   A permet d'ajouter du texte à la fin de la ligne.
   i permet d'ajouter du texte avant le curseur.
   o permet d'ajouter une ligne en dessous du curseur.
   O permet d'ajouter une ligne au dessus du curseur.

   le retour chariot permet d'aller à la ligne suivante.

   - **dd** permet de supprimer la ligne courante.
   - **X** permet de supprimer le caractère avant le curseur.
   - **u** permet d'annuler la dernière commande effectuée.

2. **Emacs ... la puissance !**

   Emacs date de la fin des années 70 et ne cesse d'évoluer depuis, ce qui fait de lui, sans aucun doute possible, l'éditeur le plus puissant au monde. Bien plus qu'un éditeur, emacs est un environnement de travail : édition, programmation, mail, news, shell ... bref on peut rester sous emacs sans avoir besoin de quoi que ce soit d'autre.

   Ses adeptes sont très nombreux.

   Et surtout ne leur dites pas qu'emacs est lourd ...vous vous tromperiez lourdement (je sais de quoi je parle ... j'ai fait l'erreur et en ce moment je suis sous emacs ... ;-))
   Lançons emacs :
   [root@mistra root]$ emacs

Ouvrons maintenant le fichier linux-test que nous avons créé précédemment sous vi :
Pour cela utilisez la séquence de touches suivante : **Ctrl-x Ctrl-f**

Vous voyez apparaître en bas de l'écran :
Find File : ~/

tapez le nom du fichier et faites un retour chariot.

Nous retrouvons notre charmant petit texte.
Vous le comprenez, la touche **Ctrl** permet de passer des commandes et de passer du mode texte au mode commande. Vous pouvez le modifier à souhait.
Les touches **Backspace** et **Suppr** fonctionnent comme sous n'importe quel éditeur.

Pour **sauver** le fichier, tapez la séquence de touches suivante :

Ctrl-x Ctrl-s

Si vous êtes bloqués dans la ligne de commande d'emacs après avoir effectué de mauvaises manipulations et que vous souhaitez retrouver le mode texte, tapez la séquence suivante :
Ctrl-g

Si vous avez fait des erreurs dans le texte, la séquence suivante permet de **supprimer les dernières modifications** :
Ctrl-x u

Si vos touches de direction ne fonctionnent pas, voici plusieurs séquences de touches qui vous permettent de vous déplacer dans votre document :
Ctrl-p : monter d'une ligne.

Ctrl-n : descendre d'une ligne.

Ctrl-f : avancer d'un caractère.

Ctrl-b : reculer d'un caractère.

Ctrl-v : avancer d'un écran (ou d'une page si vous préférez).

Alt-v : reculer d'un écran.

Ctrl-d : supprimer le caractère sur lequel le curseur se trouve.

Une commande utile est :
Ctrl-s qui permet de faire une **recherche** "dynamique" ("incrémentale") sur une suite de caractères dans le texte.

Meta-% permet de lancer un "Rechercher et remplacer" . La touche **Meta** est en général confondue avec la touche **Alt** Pour **quitter emacs**, utiliser la

combinaison de touches suivante :
Ctrl-x Ctrl-c

Avec cette rapide présentation vous pourrez déjà "barboter" un peu sous emacs. Il m'est impossible de décrire dans ce document les milliers de fonctions disponibles .... si vous souhaitez en savoir plus, cette séquence vous permettra de rentrer dans l'**aide d'emacs** :
Ctrl-h

Ou bien lancez le "tutorial" :
Ctrl-h t

Si vous devenez un mordu d'emacs (ce qui est tout a fait normal : -) vous pourrez trouver quelques ouvrages sur emacs dans toutes les bonnes bibliothèques.

3. **joe : la simplicité.**

   joe est l'éditeur que j'utilise pour faire des petites modifications dans mes fichiers de configuration par exemple, il est très leger, il ne possède pas la puissance d'emacs mais rend lui aussi service :
   Pour appeler **joe** :
   [root@mistra /root]# joe

   joe est très intuitif (à la Wordstar), pas besoin de s'étendre sur les fonctionnalités textes. Trois opérations fondamentales à connaître :

   Ctrl-k e permet d'ouvrir un fichier

   Ctrl-k d permet de sauvegarder le fichier

   Ctrl-k x permet de sauvegarder le fichier et de quitter joe

   Ctrl-c permet de quitter joe sans sauvegarder les modifications.

   joe possède de nombreuses fonctions possibles qui sont décrites dans le man (nous verrons comment y accéder dans la section consacrée à man).

sommaire

Copier un fichier (ou un répertoire) : cp.

La syntaxe de la commande **cp** est la suivante :
cp [option] fichier-origine fichier-destination

ou

cp [option] fichier répertoire

par exemple pour faire une copie de notre fichier *linux-test* en un fichier *linux-test2*, il suffit de faire :
[root@mistra /root]# cp linux-test linux-test2

Nous possédons maintenant deux exemplaires de notre fichier dans /root.

ATTENTION ! : si vous effectuez une copie d'un fichier sur un fichier qui existe déjà, celui-ci sera effacé et remplacé par le nouveau fichier.

Si vous souhaitez copier le fichier *linux-test* dans un répertoire (par exemple **/home**) en gardant le nom du fichier, utilisez la commande suivante :
[root@mistra /root]# cp linux-test /home
Pour lui donner un autre nom :
[root@mistra /root]# cp linux-test /home/linux-test2

Nous venons de voir que l'utilisation de **cp** est dangereuse et l'on risque parfois d'effacer des fichiers importants. Les options de **cp** peuvent vous éviter des situations fâcheuses.

cp -i avertit l'utilisateur de l'existence d'un fichier du même nom et lui demande s'il peut ou non remplacer son contenu. Recopions à nouveau le fichier linux-test sur linux-test2 avec l'**option -i** :
[root@mistra /root]# cp -i linux-test linux-test2
cp : overwrite `linux-test2'?

cp vous demande s'il peut écraser linux-test2 : répondre par "y" (yes) ou "n".

Quelques options importantes de cp :
cp -b permet comme l'option -i de s'assurer que la copie n'écrase pas un fichier existant : le fichier écrasé est sauvegardé, seul le nom du fichier d'origine est modifié et **cp** ajoute un tilde (~) à la fin du nom du fichier.

cp -l permet de faire un lien "dur" entre le fichier source et sa copie. Ceci signifie que le fichier copié et sa copie partageront physiquement le même espace. Cela permet des gains de place non négligeables. Plus exactement, sur le disque dur le fichier et sa copie seront le même fichier alors qu'avec une copie classique, le disque dur contiendra deux exemplaires du fichier.

cp -s permet de faire un lien "symbolique" entre le fichier source et sa copie. Le lien symbolique est un pointeur. Ainsi si nous copions le fichier *linux-test* avec l'option **-s**, lorsque par exemple nous voudrons éditer le fichier copié, linux éditera en réalité le fichier original (voir la section consacrée à ln pour un descriptif plus complet des liens).

cp -p permet lors de la copie de préserver toutes les informations concernant le fichier comme le propriétaire, le groupe, la date de création (voir les sections consacrées à chmod et chown pour plus d'informations).

cp -r permet de copier de manière récursive l'ensemble d'un répertoire et de ses sous-répertoires.

Exemple :
Je possède dans mon répertoire **/home/delcros/personnel** un répertoire intitulé "mygale" et qui contient 3 sous répertoires ("echecs", "linux", xcaissa) :

```
/home/delcros/personnel/
/home/delcros/personnel/mygale/
/home/delcros/personnel/mygale/echecs/
/home/delcros/personnel/mygale/linux/
/home/delcros/personnel/mygale/xcaissa/
```

Je souhaite copier le répertoire mygale ainsi que ses sous-répertoires dans mon répertoire **/home/delcros/**" : j'utilise la commande (en supposant que je me suis au préalable déplacé dans le répertoire **/home/delcros/personnel/** :
[delcros@mistra personnel]$ cp -r mygale /home/delcros

cp -v permet d'afficher le nom des fichiers copiés. Utile si par exemple vous copiez plusieurs fichiers (à l'aide des occurences "*" et/ou "?") et que vous souhaitez voir le bon déroulement de la "multicopie". J'aurais pu par exemple utiliser cette option lors de ma copie récursive du répertoire "mygale".

J'aurais ainsi vu ceci en associant l'option **-v** et **-r** :
[delcros@mistra personnel]$ cp -rv mygale /home/delcros

```
mygale -> /home/delcros/mygale
mygale/index.html -> /home/delcros/mygale/index.html
mygale/logo.gif -> /home/delcros/mygale/logo.gif
mygale/linux -> /home/delcros/mygale/linux
mygale/linux/linux.html -> /home/delcros/mygale/linux/linux.html
.....
```

(c'est une partie du résultat).

sommaire

Supprimer un fichier "rm".

PREAMBULE :
Nous entrons maintenant dans une zone à risque, mieux vaut donc se loguer en tant qu'utilisateur de la machine et non pas en tant qu'administrateur système (root), car nous risquerions par une mauvaise manipulation de supprimer des fichiers fondamentaux nécessaires au bon fonctionnement de linux. Nous allons donc créer un compte utilisateur, lui attribuer un mot de passe et nous loguer sur ce compte. Exécutez les commandes suivantes, une explication détaillée interviendra ensuite dans la partie consacrée à l'administration système :
[root@mistra /root]#adduser le_nom_de_choix (votre prénom par exemple, mais sans accent et si possible long de moins de 8 caractères)

[root@mistra /root]#passwd le_nom_de_votre_choix (saisir deux fois le même mot de passe, la seconde sert à confirmer)

[root@mistra /root]#cp linux-test /home/le_nom_de_votre_choix (gardons notre fichier pour continuer nos petites expériences ;).

[root@mistra /root]#chow le_nom_de_votre_choix.le_nom_de_votre _choix /home/le_nom_de_votre_choix(L'administrateur donne généreusement le fichier linux-test au nouvel utilisateur avec la commande "chown" que nous verrons dans les commandes d'administration système, pour l'instant ne vous en souciez pas.)

[root@mistra /root]#su le_nom_de_votre_choix (la commande su permet de se loguer sur un autre compte).
Il suffira de saisir **exit** pour « retomber » dans la session de travail root.

Effectuons à nouveau une copie du fichier linux-test (tapez **cd** pour vous retrouver dans votre répertoire personnel) :
[delcros@mistra delcros]$ cp linux-test linux-test2

LA COMMANDE rm
Pour supprimer le fichier "linux-test2" :
[delcros@mistra delcros]$ rm linux-test2

LES OPTIONS de rm
Comme pour **cp**, l'option **cp -i** permet à **rm** de demander à l'utilisateur s'il souhaite vraiment supprimer le ou les fichiers en question :
[delcros@mistra delcros]$ rm -i linux-test2
rm : remove `linux-test2'?

(il vous suffit donc de répondre "y" ou "n")

rm -d permet de supprimer un répertoire qu'il soit plein ou non (attention dangereux ...)

rm -r permet de supprimer un répertoire et ses sous répertoires (attention TRÈS dangereux)

rm -f permet de supprimer les fichiers protégés en écriture et répertoires sans que le prompt demande une confirmation de suppression (à utiliser avec précaution ...)

sommaire

Créer un répertoire (mkdir)

Pour créer un répertoire, il suffit de taper la commande suivante (ici je crée le répertoire "personnel" dans /home/delcros :
[delcros@mistra delcros]$ mkdir personnel

Une option de **mkdir** est souvent utile :
mkdir -p permet de créer une suite de répertoire.

Supposons que je veuille créer dans mon répertoire **/home/delcros** la suite de répertoires suivante : **doc/mygale/mail**. Je peux faire soit :
[delcros@mistra delcros]$ mkdir doc

[delcros@mistra delcros]$ cd doc

[delcros@mistra delcros]$ mkdir mygale

[delcros@mistra delcros]$ cd mygale

[delcros@mistra delcros]$ mkdir mail

Ou bien utiliser l'option **-p** qui me permet de créer la suite de répertoires "parents" le plus simplement du monde :
[delcros@mistra delcros]$ mkdir -p doc/mygale/mail

sommaire

Déplacer ou renommer un fichier (mv)

Pour comprendre la commande **mv**, voyons une suite de commandes qui effectuent des opérations différentes :
[delcros@mistra delcros]$ mv linux-test perso
renomme le fichier "linux-test" en "perso"

[delcros@mistra delcros]$ mv perso perso
va écraser le fichier existant avec la source.

[delcros@mistra delcros]$ mv personnel mon-répertoire
va renommer le répertoire personnel en mon-répertoire

[delcros@mistra delcros]$ mv perso /home/delcros/mon-répertoire
va déplacer le fichier perso dans le répertoire **/home/delcros/mon-répertoire**

Les options :

- **mv -b** ('b' comme "**b**ackup") va effectuer une sauvegarde des fichiers avant de les déplacer :
  [delcros@mistra delcros]$ mv -b mon-répertoire/perso /mon-répertoire/linux-test
  Cette commande va renommer le fichier perso en linux-test, cependant vous trouverez dans le répertoire une sauvegarde de perso (perso~).
- **mv -i** ('i' comme «interactive») demande pour chaque fichier et chaque répertoire s'il peut ou non déplacer fichiers et répertoires.
- **mv -u** ('u' comme «update») demande à mv de ne pas supprimer le fichier si sa date de modification est la même ou est plus récente que son remplaçant.
  Exemple :

Déplaçons-nous vers notre répertoire personnel puis créons un nouveau fichier avec l'éditeur de texte **joe** :
[delcros@mistra personnel]$ joe linux-test2

saissons un petit texte :
"y en a marre de ces textes stupides !"

et finissons notre session joe par la séquence de touches suivante :
Ctrl-k x

qui permet d'enregister le fichier et de quitter joe.

Notre fichier linux-test2 est plus récent que notre fichier linux-test. Vous pouvez le vérifier en effectuant un "**ls -l**".

Nous souhaitons (naïvement, bien sûr !) renommer le fichier linux-test en linux-test2. Mais nous sommes attentifs et nous ne voulons pas que le fichier linux-test2 soit écrasé si celui-ci est plus récent que linux-test :
[delcros@mistra personnel]$mv -u linux-test linux-test2

L'option **-u** nous a évité d'écraser le fichier linux-test2. La commande **mv** n'a donc pas été effective.

sommaire

Retrouver un fichier ("find")

1- La commande find

Exemple simple : comment trouver un fichier portant un nom donné ?

[delcros@mistra delcros]$ find / -name linux-test2 -print

/home/delcros/linux-test2

(Un peu long n'est ce pas pour trouver la reponse dans tout cette grosse arborescence ? :-))

En general on recherche rarement un fichier depuis la racine.

Décomposition de la commande de l'exemple :
"/" indique que nous voulons chercher à partir de la racine notre fichier.
"-name " est l'option qui indique ici que nous voulons spécifier le nom d'un fichier.
"-print" demande à **find** d'afficher le résultat.
Pour chercher tous les fichiers commencant par "linux-tes" et définir à partir de quel répertoire on souhaite effectuer la recherche on utilise cette syntaxe :
[delcros@mistra delcros]$find /home/delcros -name 'linux-tes*' -print

Le nombre d'options de **find** est impressionnant. En voici quelques unes :
-type permet d'indiquer le type de fichier que l'on recherche. Si vous cherchez seulement un répertoire et non pas un fichier vous pourrez utilisez cette option :
[delcros@mistra delcros]$find /usr -type d -name bin -print

Ici, on demande à **find** de trouver les répertoires (l'argument "**d**" (comme "**d**irectory") de l'option **-type** indique que l'on cherche un répertoire) du nom de "bin" à partir du répertoire **/usr**.

-exec ou -ok permet d'exécuter une commande sur les fichiers trouvés. La différence entre **-exec** et **-ok** est que la deuxième vous demandera pour chaque fichier trouvé si vous souhaitez réellement réaliser l'opération :
[delcros@mistra delcros]$find -name 'linux-tes*' -print -ok rm {} \;

```
./linux-test
rm ... ./linux-test ? y
```

[delcros@mistra delcros]$

Dans l'option **-exec**, la paire d'accolades se substitue aux fichiers trouvés, et l'anti-slash lié au point virgule forme une séquence d'échapemment.

On peut dire que cette présentation de **find** est assez sommaire, mais j'espère qu'elle vous laisse deviner ses capacités.

2- La commande locate

La commande **locate** a la même mission que **find**. Pourtant vous verrez qu'en utilisant la commande **locate**, le fichier sera trouvé beaucoup plus rapidement. Pourquoi ? Parce que **locate** ne va pas chercher le fichier dans toute l'arborescence des répertoires mais va localiser la position du fichier dans une base de données qui contient la liste des fichiers existants. Cette base de données est en général automatiquement générée une fois par jour par le système grâce à une commande appelée **updatedb**. Sur un système Linux Redhat, cette base de donnée se trouve dans le répertoire **/usr/lib** et se nomme **locatedb**.

La syntaxe est donc simple:

[delcros@mistra delcros]$ locate nom_du_fichier

Bien que la commande **locate** soit très intéressante, elle ne possède pas la puissance des options de **find**. De plus, si vous créez des fichiers pendant la journée et que vous les recherchez avec la commande locate, il n'est pas sûr que la base de donnée ait été remise à jour. Bref, **locate** est un complément de **find**.

3-La commande which

which vous permet simplement de connaître le chemin d'un exécutable. Exemple:

[delcros@mistra delcros]$ which ls
/bin/ls
[delcros@mistra delcros]$

sommaire

Trouver du texte dans un fichier (grep)

La commande **grep** est un pivot des commandes UNIX. Elle cherche une expression rationnelle dans un ou plusieurs fichiers, exemple :
[delcros@mistra delcros]$grep fouille linux-commande.html

```
grep, la commande qui vous fouille les fichiers
```

La commande a donc affiché la ligne qui contient le mot "fouille" dans le fichier linux-commande.html.

La richesse de la commande **grep** permet de faire des recherches sur plusieurs fichiers et d'avoir un format de sortie adéquat. Par exemple, le fichier linux-commande.html est déjà assez important et il serait agréable de savoir où se trouve cette ligne qui contient le mot *fouille* dans le fichier :
[delcros@mistra delcros]$grep -n fouille linux-commande.html

```
902: Grep, la commande qui vous fouille les fichiers
```

Le mot fouille se trouve à la ligne numéro 902 et c'est l'option **-n** qui nous a permis de connaître ce numéro.

Une autre option très utile est **-l** qui permet de n'afficher que les noms des fichiers contenant ce que l'on cherche :
[delcros@mistra delcros]$grep -l fouille /home/delcros/personnel/html/*

```
/home/delcros/personnel/html/linux-commande.html
```

Ici, j'ai demandé à la commande **grep** de chercher l'occurence "fouille" dans les fichiers du répertoire /home/delcros/personnel/html/. Le résultat est le nom des fichiers qui contiennent l'occurence. Ici, seul le fichier "linux-commande.html" dans le répertoire contient le mot "fouille". Quelques-unes des autres options :
-c donne le nombre de fois où l'expression rationnelle a été rencontrée dans le fichier :
[delcros@mistra delcros]$ grep -c fouille linux-commande.html
```
10
```

-n est utile lorsque vous cherchez une expression rationnelle qui commence par un tiret car si vous n'utilisez pas l'option **-n**, **grep** la considèrera comme une option !

sommaire

Les liens (ln)

Les liens forment un axe central du fonctionnement de linux. Qu'est ce qu'un lien ?

Un lien est un type spécial de fichier qui permet à plusieurs noms de fichiers de faire référence au même fichier sur le disque.

On doit distinguer deux sortes de liens :

1. **les liens durs** associent deux ou plusieurs fichiers à un même espace sur le disque, les deux fichiers sont pourtant indépendants. On peut dire que physiquement les fichiers sont les mêmes mais que virtuellement ils ne le sont pas. Prenons un exemple :
   [delcros@mistra personnel]$ln linux-test /home/delcros/linux-test-lien-dur

   le fichier *linux-test-lien-dur* est créé dans le répertoire */home/delcros*. si vous faites un **ls -l** vous constaterez que *linux-test* et *linux-test-lien* ont la même taille. Au niveau de leur existence sous linux, ils sont indépendants. Mais sur le disque, il n'existe qu'un seul fichier, simplement *linux-test-lien-dur* et *linux-test* sont sur le même espace (ou inode) sur le disque dur lorsqu'on les appelle. Ainsi si nous modifions le *fichier linux-test-lien-dur*, nous aurons automatiquement une modification du fichier *linux-test* (et vice et versa), car la modification s'effectuera physiquement sur le disque dur sur l'inode "partagé" par les deux fichiers.

2. **Les liens symboliques :**
   si nous faisons maitenant un lien symbolique :
   [delcros@mistra personnel]$ln -s linux-test /home/delcros/linux-test-lien-symb

   Faites un **ls -F** dans le répertoire */home/delcros*, vous verrez que le fichier *linux-test-lien-symb* est précédé du signe "@". Ce fichier pointe sur linux-test. Si vous avez fait un peu de programmation en C, nous retrouvons le concept de **pointeur**. Quand on appelle le fichier *linux-test-lien-sym*, il va automatiquement se diriger vers le fichier *linux-test*.

Quelles sont les points communs entre les liens symboliques et les liens durs ?

Le lien symbolique fait référence à un fichier dans un répertoire alors que le lien dur fait référence à un espace sur le disque dur.

- Les liens symboliques sont des fichiers de petite taille qui ont une existence propre sur le disque dur. Ces fichiers contiennent les références des fichiers sources auquels ils correspondent.

- Dans le cas d'un lien dur, la suppression de l'un des deux fichiers n'affectera pas l'autre. Dans le cas d'un lien symbolique, la suppression du fichier source entraînera un changement de comportement du fichier lien qui ne correspondra plus à un fichier valide et sera donc dit "cassé" ("broken").

Utilité des liens

Les liens sont utiles si vous souhaitez qu'un fichier apparaisse dans plusieurs répertoires, ou sous un nom différent. Imaginez que ce fichier fasse quelques megaoctets ... une copie à l'aide "cp" entraînera une perte de place non négligeable alors qu'un lien permettra de limiter l'utilisation de l'espace disque. Mieux :un lien garanti que toute modification effectuée sur ce fichier concernera toutes les apparentes « copies » dispersées.

Syntaxe de **ln** :
ln fichier-source fichier-lien ln -s permet d'effectuer un lien symbolique.

ln -b réalise une sauvegarde d'un fichier existant et dont nous aurions utilisé le nom avant de l'écraser.

ln -i demande à l'utilisateur s'il souhaite écraser le fichier qui a un lien sur le fichier source au cas ou celui-ci existerait déjà.

ln -d effectue des liens durs sur des répertoires ... seuls les utilisateurs possédant les droits adéquats pourront le faire.

sommaire

Le compactage et le décompactage des fichiers au format .gz : la commande gzip

Pour compacter un fichier, taper la commande suivante :
[delcros@mistra delcros]$ gzip non_du_fichier

Pour décompacter un fichier, taper la commande suivante :
[delcros@mistra delcros]$ gzip -d non_du_fichier.gz

sommaire

Le décompactage des fichiers avec la commande uncompress

Si vous rencontrez un fichier au format **.Z** (un autre type de compression plus ancien, et moins performant), vous pouvez aussi utiliser **gzip -d**.

sommaire

Archivage de données : la commande "tar"

La commande **tar** permet d'archiver ou de désarchiver des répertoires et des fichiers de facon optimale.
Une des commandes dont vous aurez certainement le plus besoin est :
[root@mistra /]# tar xzf nom_du_fichier.tar.gz

Cette commande décompacte un fichier au format **.tar.gz** ou **.tgz** ; vous rencontrerez régulièrement ce genre de fichier en voulant par exemple récupérer des logiciels pour linux sur l'Internet. Le format **.tar.gz** indique que le fichier est en réalité une archive (**.tar**), c'est-à-dire que le fichier contient en réalité plusieurs fichiers, et qu'il est compacté (**.gz**). La commande précédente peut être ainsi comprise :
x (e**x**tract) permet d'extraire certains fichiers d'une archive (lorsque l'on ne spécifie pas les noms des fichiers que l'on souhaite extraire de l'archive, **tar** les extrait tous).

z décompacte l'archive

f extrait un fichier donné (ici le fichier est nom_du_fichier.tar.gz).

Une autre commande permet de connaître la liste des fichiers contenus dans un fichier **.tar.gz** ou **tgz** :
[root@mistra /]#tar tvzf nom_du_fichier.tar.gz

t affiche la liste des fichiers contenus dans une archive tar.

v est le mode "verbose", qui affiche les noms des fichiers tel qu'ils ont été archivés à l'origine.

C'est donc l'option **t** qui permet de voir comment les fichiers de l'archive seront désarchivés.

La commande suivante créera une archive de tout mon répertoire **/home/delcros/personnel** :

[delcros@mistra delcros]# tar cvfz personnel.tgz personnel

c indique à tar de créer une archive

z indique a tar de compacter une archive

Ainsi tout mon répertoire personnel, avec les sous répertoires et tous les fichiers, se trouveront rassemblés dans UN fichier archive : personnel.tgz

sommaire

Connaître l'espace disque utilisé (df et du)

La commande **df** permet de connaître l'emplacement de montage des systèmes de fichiers (partitions utilisables pour stocker des fichiers) accessibles sur votre système et les capacités restantes sur chacun d'eux.

```
[delcros@mistra delcros]$ df
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/sda5             298762  119387   163945     42%   /
/dev/sda1              41166   17116    24050     42%   /dos
/dev/sda6            1745186 1163946   491042     70%   /usr
[delcros@mistra delcros]$
```

La commande **du** permet de connaître l'utilisation disque en kilo-octet par le répertoire spécifié et ses sous répertoires.

```
[delcros@mistra html]$ du
56      ./config
224     ./images
185     ./commandes
28      ./.xvpics
2       ./docs/preparation_debutantlinux
203     ./docs
875     .
[delcros@mistra html]$
```

sommaire

Contrôler les ressources utilisées par les processus

1. La commande "**top**" :

   La commande **top** vous permet d'afficher des informations en continu sur l'activité du système. Elle permet surtout de suivre les ressources que les processus utilisent (quantité de RAM, pourcentage de CPU, la durée de ce processus depuis son demarrage).
   Vous pourrez utiliser l'option **-d** pour spécifier des délais de rafraîchissement (en secondes).
   En cours d'utilisation de top, il est possible de stopper un process de manière interactive en tapant **k**. top demande ensuite lquel signal il doit envoyer : 15 (SIGTERM) est le signal par défaut qui met fin à un process, 9 (SIGKILL) est plus brutal.
   Pour quitter top, appuyer simplement sur la touche "**q**".

2. La commande "**ps**" :
   La commande **ps** permet de connaître les processus actifs à un moment donné :
   [delcros@mistra delcros]$ ps

```
3.          PID   TTY STAT  TIME COMMAND
4.          341   p1    S     0 : 00    bash
5.          344   p2    S     0 : 00    bash
6.          1039   p3    S     0 : 00    bash
7.          1219   p3    R     0 : 00    ps
```

Le "PID" est l'identificateur d'un processus, c'est un nombre. Chaque processus est identifié dans le système par un nombre unique.

Le "TTY" indique à quel port de terminal est associé le processus.

"STAT" indique l'état dans lequel se trouve le processus. Dans l'exemple, trois processus sont endormis (S comme "sleep"), et un processus en cours d'exécution (R comme "run"). Le processus qui est en cours d'exécution n'est autre que la commande "**ps**" que nous venons de lancer.

Le "TIME" indique depuis combien de temps le processus utilise les ressources du microprocesseur.

Le "COMMAND" précise, comme son nom l'indique, la commande dont l'état est décrit par PID, TTY, STAT et TIME.

Ceci dit, une simple commande "**ps**" n'indique pas tous les processus du système. Le simple fait de lancer **ps** nous a juste indiquer les processus associés à un terminal et qui dépendent de l'utilisateur courant (ici "delcros"). En fait, il est tout a fait probable que d'autres processus non liés à un terminal aient été lancés par "delcros". J'en suis d'ailleurs sur, puisque actuellement j'utilise emacs pour réaliser cette modeste page de documentation et que pour visualiser le résultat, j'utilise netscape :
[delcros@mistra delcros]$ ps -x

```
  PID TTY STAT  TIME COMMAND
  240  ?  S    0:01 /usr/X11R6/bin/fvwm2
  246  ?  S    0:00 /usr/X11/bin/xautolock -corners ++++ -time
5 -locker /usr/X
  247  ?  S    0:00 /usr/X11/bin/unclutter -idle 3
  253  ?  S    0:00 /usr/local/bin/Periodic
  254  ?  S    7:34 emacs --background grey79 -geometry 80x58+-
4+-11
  257  p0 S    0:00 bash
  258  p2 S    0:00 bash
  259  p1 S    0:00 bash
  272  ?  S    0:00 /usr/lib/emacs/19.34/i386-gnu-
linux/emacsserver
 2134  ?  S    0:00 /usr/bin/ispell -a -m -d francais
 6431  p0 S    1:03 /usr/lib/netscape/netscape-navigator
 6441  p0 S    0:00 (dns helper)
 6741  p0 R    0:00 ps -x
```

Les commandes qui ne sont pas associées à un terminal sont reconnaissable par le point d'interrogation qui rempli le champs TTY.

Si vous voulez connaître tous les processus de la machine de tous les utilisateurs, il suffit d'utiliser l'option **ax**. Si en plus vous voulez connaître les utilisateurs associés à chaque processus, il vous suffit d'utiliser l'option **aux**. Vous verrez alors plusieurs colonnes s'ajouter dont "USER" qui indique à quel utilisateur appartient le processus. "%CPU" indique en pourcentage les

ressources du microprocesseur utilisées par le processus. "%MEM" montre en pourcentage les ressources en mémoire vive utilisées par le processus. "RSS" donne réellement la mémoire utilisée en kilobytes par le processus. "START" indique l'heure à laquelle le processus a été lancé.

Comment être plus précis ? : -)

8. La commande "**pstree**" :
   Cette commande permet d'afficher les processus sous forme d'arborescence et donc de voir leurs inter-dépendances :
   [delcros@mistra delcros]$ pstree
   init-+-crond
   |-emacs---emacsserver
   |-gpm
   |-inetd
   |-kerneld
   |-kflushd
   |-klodg
   |-kswapd
   |-loadmeter
   |-lpd
   |-6*[mingetty]
   |-named
   |-netscape---netscape
   |-4*[nfsiod]
   |-nxterm---slrn-gor---slrn
   |-portmap
   |-pppd |-rc.news---innwatch---sleep
   |-rpc.mountd
   |-rpc.nfsd
   |-rpc.yppasswdd
   |-sendmail
   |-syslogd
   |-update
   |-xconsole
   |-xdm-+-X
   | `-xdm---Xsession---fvwm---FvwmPager
   |-xterm---bash---su---bash---tail
   |-2*[xterm---bash]
   |-xterm---bash---pstree
   \ -ypserv

   On voit par exemple ici que j'utilise Fvwmpager qui depend en fait lui-même de fvwm et lui même dépend de Xwindow ici lancé grace à xdm (vous n'obtiendrez pas la même chose que moi si vous lancez Xwindow grâce à la commande startx, en effet xdm permet de lancer automatiquement Xwindow au démarrage de linux).

9. La commande "**kill**" :
   La commande "**kill**" permet d'expédier un signal à un processus en cours.

Sa syntaxe est la suivante :
kill [options] PID

Par exemple, si j'ai lancé une connexion à l'Internet en PPP, un processus pppd sera en cours. Pour tuer le processus, je peux d'abord faire un **ps -ax** pour connaître le numero du PID de pppd et ensuite si par exemple le PID est 592, je peux tuer la connexion en faisant :
[root@mistra delcros]# kill 592

Vous remarquerez que je suis logué en utilisateur "root" pour faire ceci, en effet le processus pppd appartenait à l'utilisateur "root" et un autre utilisateur ne peut pas lui expédier de signal.

Si un processus vous résiste, c'est à dire que vous n'arrivez pas à le tuer, vous devez utiliser la commande : **kill -9 PID** (PID étant toujours le numéro de de processus).

La commande "**killall**" permet aussi de tuer un processus mais au lieu d'indiquer le PID vous indiquerez le nom du processus.

Mais **attention**, plusieurs processus peuvent utiliser la même commande. Ainsi, si vous tapez :
[delcros@mistra delcros]# killall grep

Vous tuerez tous les processus qui contiennent la commande grep. Je vous recommande donc d'utiliser l'option "-i" qui vous demande une confirmation avant de tenter d'arrêter un processus..

sommaire

La connexion de plusieurs commandes : les pipes

Qu'est ce qu'un "pipe" (parfois appelé « tube ») ? Si on le décrit ce n'est rien d'autre que cette barre verticale que vous pouvez obtenir avec la combinaison de touches "Altgr + 6" sur les clavier français classiques, ou "Altgr + 1" sur les claviers franco-belges. Un tube permet de passer le résultat d'une commande à autre commande. Un exemple permettra de comprendre tout cela beaucoup plus facilement :

Je veux savoir quels sont tous les processus "bash" qui fonctionnent sur le système, mais je veux que la commande **ps aux** ne me fournisse les lignes que les lignes qui contiennent le mot "bash" pour m'eviter d'avoir à parcourir toute la longue liste qu'affiche **ps aux** :

```
[delcros@mistra html]$ ps aux | grep bash
delcros    367    0.0   1.8   1600   568   p2   S   18 : 14   0 : 00
bash
```

```
delcros    426    0.0   2.2   1624   704   p3   S   18 : 17   0 : 00
bash
delcros   1261   0.0   2.2   1608   692   p6   S   21 : 22   0 : 00
bash
delcros   1332   0.0   2.4   1616   772   ?    S   21 : 41   0 : 00
bash
delcros   1582   0.0   2.7   1604   844   p8   S   22 : 30   0 : 00
bash -rcfile .bashrc
delcros   2796   0.0   0.9    908   300   p3   S   02 : 17   0 : 00
grep bash
root       1162   0.0   2.1   1596   664   ?    S   21 : 06   0 :
00 bash
```

On peut dire que l'on a "connecté" deux commandes entre elles. Mais vous pouvez ainsi en connecter autant que vous voulez en utilisant cette syntaxe :
commande1 | commande2 | commande3 ... | commandeN Si on se rend compte de l'utilité des pipes, progressivement on les utilise et on fini par ne plus s'en passer.

sommaire

Les redirections

Quand on parle de redirection, on parle plus précisemment de la redirection des entrées-sorties que traitent ou engendrent les programmes. Par exemple, lorsque vous tapez des commandes au prompt de linux, vous effectuez une entrée de caractère grâce au clavier et linux vous donne une sortie en vous donnant à l'écran le résultat de votre commande. Mais l'entrée de données peut se faire autrement que par le clavier, en indiquant par exemple un fichier qui contient des données à traiter. La sortie peut aussi s'effectuer ailleurs que sur l'écran, sur l'imprimante par exemple.

Ainsi, lorsque nous parlons des entrées sorties, nous parlons aussi des périphériques de l'ordinateur. On considérera que les périphériques sont des fichiers a part entière car, sous UNIX, des fichiers spéciaux permettent l'accès aux périphériques se trouvent dans le répertoire **/dev**. Dans la plupart des cas ce que l'on y copie va vers le périphérique.

Mais comment faire pour rediriger une entrée ou une sortie ?

Comment faire par exemple pour que la commande **cat** qui affiche un fichier à l'écran, sorte plutot le fichier dans un autre fichier ou vers une imprimante ? C'est le signe > qui va nous permettre de réaliser ceci.

Il est temps de prendre un exemple....

Dans un premier cas, je veux que linux m'affiche le fichier test à l'ecran :
[delcros@mistra delcros]$ cat test

Vous allez voir s'afficher à l'écran le fichier test.

Dans un deuxième cas, je veux que linux place le fichier test dans un fichier test2 au lien de l'afficher à l'écran :
[delcros@mistra delcros]$ cat test > test2

Dans un troisième cas, je veux que linux imprime le fichier au lieu de l'afficher à l'écran :
[delcros@mistra delcros]$ cat test > /dev/lp0

Quelques constats s'imposent :
1- La sortie sur un autre fichier n'est rien d'autre avec la commande **cat** qu'une copie du fichier "test" en "test2". La commande **cp** nous permet aussi de faire cela.

2- Dans la redirection vers l'imprimante nous avons indiqué le fichier spécial **/dev/lp0** qui correspond au port **LPT1** où est connectée mon imprimante.

La commande **cat** affiche son résultat vers la sortie standard qui est le terminal.

Par défaut le terminal est la sortie standard, ce descripteur de fichier est désigné par le chiffre "1"

L'entrée standard dans un système UN*X est le clavier et est désigné par le chiffre "0".

Il existe un troisième **descripteur de fichier** qui est la sortie des erreurs produites par l'exécution d'une commande.
La sortie des erreurs se fait par défaut sur le terminal et est désigné par le chiffre "2".

Plusieurs types de redirection existent :

- "**> fichier**" qui permet de rediriger le résultat d'une commande vers une sortie que nous choisissons.
- "**< fichier**" permet de spécifier une entrée standard.
- "**>> fichier**" permet comme le signe ">" de rediriger la sortie standard vers un fichier, mais si le fichier spécifié existe déjà, la sortie sera ajouté à ce qui existe déjà dans le fichier alors qu'avec un simple "> le fichier spécifié serait écrasé.
- "**<> fichier**" permet de spécifier un fichier comme étant en même temps l'entrée standard et la sortie standard.
- "**n> fichier**" permet de rediriger la sortie d'un des descripteurs de fichiers vers un fichier. Par exemple, si vous souhaitez obtenir les erreurs standards dans un fichier vous n'aurez qu'à utiliser cette syntaxe :
  commande 2> erreurs
- "**n< fichier**" permet de spécifier un fichier comme étant un des descripteurs de fichier.
- "**>&n**" permet de dupliquer la sortie standard vers un des descripteurs de fichier.
- "**<&n**" permet de dupliquer l'entrée standard depuis un des descripteurs de fichier.
- "**&> fichier**" permet de rediriger la sortie standard et l'erreur standard vers un seul et même fichier.

À première vue, on se demande bien à quoi peut servir certaines des redirections ...

On les découvre au fur et à mesure, mais une des plus utiles est 2>&1 qui permet de rediriger les erreurs vers la sortie standard. Elle est très appréciée des utilisateurs lorsque par exemple ceux-ci n'arrivent pas à lancer l'interface X-Window. Il est alors courant de recourir à la commande suivante afin d'obliger X à placer tous ses messages dans un fichier nommé **erreursX** que l'on pourra consulter ensuite à loisir :
[delcros@mistra delcros]$ startx 2>&1 erreursX.tmp

2. bash et ses capacités

Le but de cette section n'est absolument pas d'expliquer la programmation et la configuration bash (loin de moi cette prétention ....), pour apprendre le bash, la lecture de Le Shell Bash, configuration et programmation" est fortement recommandée (sinon consultez la page de manuel sur bash "**man bash**").

Le shell bash, comme les autres shells (korn shell, C shell), permet ce qui a été vu précédemment, c'est-à-dire de lancer des commandes, de créer des pipes, de connecter par pipes des commandes ...

Mais avec les commandes décrites depuis le début de ce document et à l'aide d'une syntaxe proche de celle des langages de programmation courants comme le C ou le Pascal, on peut réaliser des scripts permettant d'automatiser certaines tâches. Nous n'allons pas décrire ici en détail ce language de programmation mais simplement montrer quelques exemples :
Un exemple, on utilise souvent cette syntaxe pour décompresser et désarchiver un fichier au format fichier.tar.gz :
gzip -dc fichier.tar.gz | tar xfBp -

(on peut aussi utiliser uniquement les options de la commande **tar** pour réaliser ceci.)

Il est assez pénible d'avoir à taper systématiquement cette longue commande.
Un script bash peut simplifier les choses :
#!/bin/bash
gzip -dc $1 fichier.tar.gz | tar xfBp - On enregistre ensuite le fichier sous le nom "montar" puis on le rend exécutable grace à la commande suivante :
chmod +x montar

pour décompressez un fichier il vous suffira de taper ceci :
montar fichier.tar.gz

Quelques remarques :
- Tout script bash doit commencer à la première ligne par une invocation du shell : *#!/bin/bash*

- les paramètres passés sur la ligne de commandepar l'utilisateur du script sont pour ce dernier des variables nommées : $1 pour le premier, $2 pour le deuxième, $3 pour le troisième, etc ... $0 étant la variable représentant le nom de la commande.

Voila qui simplifie déjà suffisamment la vie.

Nous pourrions améliorer ce script en voyant d'abord de quoi est composé le fichier.tar.gz avant la décompression.

#!/bin/bash
tar tvzf $1
gzip -dc $1 | tar xfBp -

Il serait cependant plus utile de pouvoir accepter ou non le désarchivage du fichier selon les informations fournies par la commande **tar tvzf** :
#!/bin/bash
tar tvzf $1
echo -n "Voulez vous désarchiver l'archive ? (o/n) : "
read archi
if [$archi = "o"]
then
gzip -dc $1 | tar xfBp -
else
exit
fi

La commande **echo** permet d'afficher un message sur la console, et l'option **n** permet de ne pas faire de retour chariot en fin de ligne.
La commande **read** attend une réponse de l'utilisateur, ici la réponse sera stockée dans la variable *archi*.
Les crochets ([ ]) encadrent tous types d'expression.
Enfin, la condition *if* permet de tester la valeur de la réponse donnée par l'utilisateur.

Voici la construction typique de l'instruction *if* :
if *condition*
then
instruction
else
instruction
fi

Si vous souhaitez insérer plusieurs conditions "**if**" utilisez la syntaxe suivante :
if *condition*
then
instruction
elif *condition*
then
instruction
else
instruction
fi

Nous pourrions utiliser aussi un menu qui nous permettrait de choisir entre une décompression immédiate ou une visualisation du contenu de l'archive :

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

```
PS3='votre choix ?'
select choix in "tar tvzf" "tar xvzf"
do
$choix $1;
done
```

La construction **select** permet de générer des menus avec une grande facilité.
PS3 est une variable qui permet de stocker une chaîne d'invite qui est utilisée par **select**.

"choix" est le nom de la variable qui contiendra un des éléments de la suite qui suit le mot clé **in**. Dans notre cas, "choix" contiendra soit la chaîne "tar tvzf" ou la chaîne "tar xvzf".

Dans la construction **do... done**, nous placons les commandes que nous voulons executer. Ici "$choix" contiendra donc soit "tar tvzf" soit "tar xvzf" et "$1" contiendra l'argument (ici le nom du fichier compressé) que l'on aura indiqué à l'execution de notre script.

Si notre script s'appelle "ctgz", son exécution se déroulera ainsi :

```
[delcros@mistra binaire]$ ./ctgz fichier.tar.gz
1) tar tvzf
2) tar xvzf
votre choix ?
```
L'utilisateur n'a plus qu'à taper "1" ou "2".
```
select nom [in liste]
do
instructions utilisant la $nom
done
```

Comme pour tout langage de programmation, bash contient des instructions de répétition :
La boucle **for** permet de réaliser une instruction un nombre de fois précis. Sa syntaxe est très proche de celle de **select** :

```
for nom [in liste]
do
instructions utilisant $nom
done
```

exemple :

```
#!/bin/bash
for fichier in $@
do
tar tvzf $fichier
done
```

Ce petit script permet de regarder le contenu de plusieurs fichiers compressés. $@ contient la liste des fichiers que l'utilisateur aura spécifié en argument de la ligne de commande :

```
[delcros@mistra binaire]$ ./utgz5 fichier1.tar.gz fichier2.tar.gz
```

La boucle **while** ainsi que la boucle **until** effectue la même chose que **for** à la différence que celle-ci répète une instruction tant que (while) ou jusqu'à ce que (until) une condition soit vérifiée.

Voici un exemple avec la boucle until :

```
#!/bin/bash
until tar tvzf $1; do
echo "tentative de decompression"
done
```

Avec cette boucle, tant que le fichier n'aura pas pu être décompressé et désarchivé, la commande **tar** sera répétée indéfiniment ... pour sortir de la boucle utilisez la combinaison de touches Ctrl-c.

Avec ces quelques structures de contrôle on voit bien la simplification des tâches quotidiennes que bash peut permettre, au prix d'un effort réduit.

La personnalisation des variables d'environnement :

bash contient des variables qui permettent d'adapter son environnement à ses besoins : Il existe un fichier qui met en place une grande partie des variables d'environnement : le fichier **.bash_profile** (ou .profile).

Pour que les variables d'environnement soit prises en compte vous devez vous reloguer sur votre compte(avec la commande "**su - nom_utilisateur** (si vous avez modifié le **.bash_profile**) ou alors passer les variables directement en ligne de commande (dans ce cas, les variables ne seront pas enregistrées dans le **.bash_profile**).

Vous trouverez par exemple la variable **PATH** qui définit les chemins existant pour les exécutables. Si par exemple, votre chemin **PATH** est de la forme :
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin

et que vous souhaitez ajouter dans ce chemin un répertoire /home/delcros/binaire qui contient votre script bash ou vos programmes personnels, il vous faudra ajouter ce chemin à la variable **PATH** :
PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/home/delcros/binaire

(notez la présence de "**:**" entre chaque nom de répertoire).

La variable **PS1** contient la forme de votre invite :
PS1="[\u@\h \w]" affichera votre nom d'utilisateur (\u); "@"; le nom de la machine (\w); un espace; le répertoire de travail courant (\w). Voilà ce que cela donne :
`[delcros@mistra /usr/X11]`

Voici une autre configuration d'invite qui contient quasiment toutes les options possibles :
PS1="[\t \d \u@\h \w \$]"

ce qui donne :
`[21 : 47 : 13 Sun Apr 26 delcros@mistra /usr/X11 $]` Une autre variable utile est **MAIL**. Normalement, vos mails arrivent dans le répertoire /var/spool/mail/nom_utilisateur

Vous pouvez placer cette variable dans votre .bash_profile avec cette forme :
MAIL=/var/spool/mail/nom_utilisateur

Les alias
Les alias sont une des choses les plus pratiques qui soient. Régulièrement on utilise les mêmes commandes avec parfois de nombreuses options. Les alias se placent habituellement dans le fichier de configuration **.bashrc**. Voici un exemple classique d'alias :
`alias l="ls --color=auto"` Avec cet alias, vous n'aurez plus besoin de spécifier systématiquement l'option "--color" qui permet de lister en couleur le contenu d'un répertoire. Il vous suffira simplement de taper l'alias "**l**".

Ainsi, le mini script que nous avions réalisé au début de cette section pourrait aussi se faire grâce à un simple alias :
`alias montar="tar xvzf"`

sommaire

---

3. Organisation des répertoires

Voici l'arborescence d'un système UNIX classique :
/ est le répertoire racine, tous les autres répertoires en dépendent. Par exemple le répertoire où est "monté" mon CD-ROM est sur /mnt/cdrom. On n'a donc pas comme sous MS-DOS, différentes lettres qui correspondent à différents lecteurs disctincts physiquement. Les lecteurs sont harmonieusement montés en répertoires dans l'arborescence UNIX.

/bin contient les binaires fondamentaux à la gestion de Linux. On y retrouve par exemple les commandes précédemment étudiées.

/dev contient une multitudes de fichiers dits spéciaux. L'un deux correspond à mon modem. Je dois indiquer ce fichier dans la configuration de mes outils de communication. De même **/dev/hda1** correspond à la première partition de mon disque dur IDE, si mon disque dur est un SCSI, son nom sera **/dev/sda1**. Un dernière exemple : /dev/fd0 correspond à mon lecteur de disquettes. Pour une application, allez voir la "section consacrée à mount".

/etc contient tous les fichiers de configuration de linux. On y retrouve par exemple le fichier /etc/passwd, qui définit les mots de passe des utilisateurs.

/sbin contient les binaires du système. On y trouve par exemple la commande **shutdown** qui permet d'arrêter l'ordinateur.

/home est le répertoire qui contient les répertoires des utilisateurs du système. Le répertoire des utilisateurs est automatiquement créé avec la création d'un compte. J'ai par exemple dans mon ordinateur un compte que j'utilise en permanence (comme maintenant, pendant la rédaction de ce petit guide), tous mes fichiers personnels sont dans /home/delcros. J'ai un autre utilisateur de ma machine, lui se logue en tant que « gorka ». Il stocke ses fichiers dans le répertoire **/home/gorka**).

/lost+found est le répertoire des fichiers perdus. Ces fameux fichiers qui, du fait d'erreur disque, se retrouvent sans chemin d'accès. Le binaire **fsck**, qui est lancé régulièrement au démarrage de linux, se charge de les détecter et de les stocker dans le répertoire /lost+found

/tmp est un répertoire accessible par tous les utilisateurs du système, il permet de ne pas encombrer son répertoire personnel par des fichiers que l'on souhaite de toute manière rapidemment détruire ou modifier.

/var/spool est le répertoire des fichiers qui servent de file d'attente. Par exemple, les files d'attente de l'imprimante se trouvent sous ce répertoire. Les données à imprimer, envoyer, ... sont stockées dans ces files d'attentes jusqu'à ce qu'elles soient traitées.

/usr contient grosso modo tout ce qui concerne les binaires utiles à tous les utilisateurs et quelques commandes d'administration. On y trouve cependant d'autres choses :
/usr/bin contient donc les binaires disponibles pour les utilisateurs et les scripts.

/usr/X11R6 contient tout ce qui concerne Xfree86 (les bibliothèques, les binaires, la documentation).

/usr/include contient tous les "headers" nécessaires à la programmation dans les différents languages.

/usr/lib contient toutes les bibliothèques nécessaires au fontionnement des logiciels. (comme par exemple la bibliothèque C ou C++ ou tcl/tk).

/usr/local on y met ce qu'on veut, mais surtout les fichiers d'usage local. J'y place les logiciels qui ne sont pas habituellement livrés avec linux et que j'ai trouvé dans d'autres CD-ROM ou sur l'Internet.

4. Quelques commandes d'administration système

-Placer les propriétés (chmod)

Introduction : linux permet de spécifier les droits qu'ont les utilisateurs sur un fichier. Pour voir ces droits, il suffit d'utiliser la commande **ls -l** :

[delcros@mistra delcros]$ ls -l perso
-rw-r--r-- 1 delcros delcros 9 Jul 19 12 : 39 perso

c'est la partie qui contient : *-rw-r--r--* qui nous intéresse pour l'instant.

Le premier tiret signifie que perso est un fichier tout ce qu'il y a de plus classique. Si à la place du premier tiret on observait un "**d**" cela signifierait qu'en réalité le fichier est un répertoire. Si à la place du premier tiret on observe un "**l**", cela signifie que le fichier est un lien.

Ensuite nous devons décomposer en trois parties les 9 dernières caractères :
rw- | r-- | r--

1. La première partie fixe les droits de propriétés pour le propriétaire du fichier.
2. La deuxième partie fixe les droits accordés aux utilisateurs faisant partie du groupe auquel appartient le fichier.
3. La dernière partie fixe les droits des autres utilisateurs.

Dans chaque partie, le premier caractère correspond au droit de lecture ("**r**"), la deuxième caractère correspond au droit d'écriture ("**w**"), le troisième caractère correspond au droit d'exécution ("**x**"). Si à la place d'un des caractères nous ne voyons qu'un tiret "-", c'est que le droit n'est pas autorisé.

On voit ainsi que tous les utilisateurs ont le droit de lire ("**r**" comme "read") le fichier et que seul son propriétaire a le droit de le modifier ("**w**" comme "write").

Par contre personne ne peut exécuter ce fichier (normal ce n'est ni un script, ni un binaire). Si par exemple tout le monde pouvait exécuter le fichier on aurait le dernier tiret de chaque partie remplacé par un "**x**" comme "eXécutable".

rwx | r-x | r-x

Cette spécificité d'UNIX sur la méthode de fixation des permissions sur un fichier assure une très grande sécurité et une très grande souplesse.

Dès maintenant, nous donnerons la lettre "**u**" pour le propriétaire du fichier, la lettre "**g**" pour le groupe d'utilisateur qui possède le fichier, la lettre "**o**" pour les autres utilisateurs. La lettre "**a**" nous permettra de faire référence à tous les utilisateurs. Cette notation est nécessaire car c'est celle que l'on doit utiliser avec la commande **chmod**.

C'est donc la commande **chmod** qui permet de modifier ces permissions qu'ont les utilisateurs sur le fichier. Évidemment, seul le propriétaire du fichier a le pouvoir de modifier ces permissions (à part bien sur le superutilisateur "root" qui peut faire absolulement tout ce que bon lui semble ...

Par exemple, nous décidons que n'importe qui pourra modifier notre fichier linux-test :
[delcros@mistra delcros]$ chmod a+w linux-test

"**a**" indique que tous les utilisateurs seront touchés par la modification des permissions

"**+**" signifie que c'est une permission supplémentaire que l'on donne. Pour en supprimer une il suffit de remplacer le signe "**+**" par "**-**".

"**w**" signifie que c'est la permission d'écriture que nous donnons.

Pour vérifier que tout a bien fonctionné, faites un "**ls -l linux-test**", nous obtenons :
-rw-rw-rw- 1 delcros delcros 9 Jul 19 19 : 03 linux-test

Si maintenant nous voulons supprimer ce droit d'écriture mais aussi le droit de lecture pour le groupe propriétaire et les autres utilisateurs nous utilisons la syntaxe suivante :
[delcros@mistra delcros]$ chmod go-wr linux-test

"**go**" signifie que la commande affectera le groupe propriétaire et les autres utilisateurs.

"**wr**" signifie que la modification portera sur les droits d'écriture ou de lecture. (on aurait pu aussi écrire la commande en mettant "rw", l'ordre n'a pas d'importance).

Dernier exemple : je souhaite que le propriétaire du fichier puisse exécuter ce fichier :
[delcros@mistra delcros]$ chmod u+x linux-test

Ainsi le propriétaire du fichier a le droit d'exécuter linux-test (ce qui de toute manière dans ce cas ci ne servira pas à grand chose puisque linux-test n'est ni un binaire ni un script ...)

Si nous souhaitons définir d'un seul mouvement toutes les permissions d'un fichier, on peut utiliser la syntaxe suivante (nous voulons que linux-test soit en lecture, en écriture et en exécution pour le propriétaire, que le groupe n'ait le droit que de le lire et d'écrire et que les autres utilisateurs ne puissent que le lire) :
[delcros@mistra delcros]$ chmod u=rwx,g=rw,o=r linux-test

En une seule ligne grâce au signe "=" nous avons définit l'ensemble des droits. Il existe une autre facon d'indiquer les permissions, nous aurions pu utiliser la syntaxe suivante pour l'exemple précédent :
chmod 764 linux-test

La syntaxe est vraiment très différente ...

En réalité, nous venons d'utiliser la notation binaire pour définir les droits :
Petit rappel :

```
Binaire ------- Logique ------ Décimal
    000 -------- (---) ------- 0
    001 -------- (--x) ------- 1
    010 -------- (-w-) ------- 2
    011 -------- (-wx) ------- 3
    100 -------- (r--) ------- 4
    101 -------- (r-x) ------- 5
```

```
110 -------- (rw-) ------- 6
111 -------- (rwx) ------- 7
```
Le 0 indique donc un tiret et le 1 indique que la lettre correspondant à la position doit être inscrite. Donc pour notre exemple, rwx (pour le propriétaire) correspond à 7, rw (pour le groupe correspond à 6, et r (pour les autres utilisateurs) correspond à 4. Nous avons bien la séquence 764. les chiffres doivent être dans l'ordre, le premier pour le propriétaire, le deuxième pour le groupe, le troisième pour les autres utilisateurs.

sommaire

- Définir le propriétaire et le groupe d'un fichier (chown)

Préambule : cette commande nécessite d'être administrateur système, il vous faut donc vous loguer en root (utiliser la commande "**su**" pour vous loguer en root) :
[delcros@mistra /home]$ su root
Password :

lorsque nous avons effectué un **ls -l** sur le fichier linux-test, nous avons obtenu :
-rw-r-r-- 1 delcros delcros 9 Jul 19 19 : 03 linux-test

Le premier nom "delcros" est le propriétaire du fichier, c'est lui qui peut placer les droits de propriété sur le fichier. Le deuxième nom "delcros" indique le groupe utilisateur du fichier. C'est l'administrateur système qui peut décider des utilisateurs qui feront partie du groupe (dans certains cas, l'administrateur système peut permettre à un utilisateur de déterminer lui même qui fera partie du groupe). Le fichier */etc/group* montre les différents groupes qui existent dans le système).

Je peux décider par exemple que le fichier linux-test n'appartienne plus à l'utilisateur "delcros" mais à l'utilisateur "thomas" :
[root@mistra delcros]# chown thomas.delcros linux-test

Vérifions :
[root@mistra delcros]# ls -l linux-test
-rwxrw-r-- 1 thomas delcros 9 Jul 19 19 : 03 linux-test

Le nouveau propriétaire du fichier est bien thomas.

Une option de **chown** est à connaître :
chown -R (récursif) permet de modifier les permissions de d'un répertoire et de ses sous-répertoires :
Il m'est arrivé par exemple de copier de la documentation qui se trouvait dans un répertoire "doc" dont le propriétaire était l'administrateur système dans le répertoire d'un utilisateur pour qu'il en ait la plus totale disposition.

J'ai donc d'une part copié tout le répertoire et ses sous répertoires dans le répertoire de l'utilisateur grâce à la commande "**cp**" et son option "**-r**" (voir la section consacrée à **cp**) et j'ai donc dû aussi modifier les droits de propriétés de tout ce répertoire et de ses

sous répertoires grâce à la commande **chown** et son option **-R** :
[root@mistra delcros]# chown -R delcros.delcros doc

ceci a permis de fixer en une seule fois le propriétaire de plusieurs sous répertoires et de fichiers.

sommaire

- Ajouter un utilisateur et changer le mot de passe

Utilisez la commande **adduser** pour ajouter un utilisateur :

Je veux par exemple créer un compte utilisateur "ernest" :
[root@mistra /]# adduser ernest

Le compte est créé, c'est-à-dire qu'un répertoire ernest a été créé dans le répertoire

/home et l'utilisateur ernest a été ajouté dans le fichier de configuration /etc/passwd.

Il ne vous reste plus qu'à déterminer un mot de passe pour l'utilisateur ernest à l'aide de la commande **passwd**

[root@mistra /]# passwd ernest

passwd vous demande de rentrer deux fois le même password.

Vous pouvez maintenant quitter la session en cours (commande "exit") puis vous loguer en tant qu'"ernest", ou bien utiliser la commande "**su**" :
[root@mistra /]# su ernest

Ou encore en ouvrant une nouvelle console (linux permet d'ouvrir plusieurs consoles) en utilisant la combinaison de touches suivante :
Alt-F2

pour revenir sur la première console vous devez simplement faire :
Alt-F1

(Sous l'environnement graphique X, on utilisera **Ctrl-Alt-F1**, **Ctrl-Alt-F2**, etc ...)

sommaire

- Décrire un utilisateur : "chfn"

Cette commande vous permet d'indiquer dans le fichier /etc/passwd différentes informations sur un utilisateur dont son nom, son bureau, ses numeros de téléphone, exemple :

```
[delcros@mistra html]$ chfn
Changing finger information for delcros.
Password :
Name [Armand Delcros] : Armand Delcros
Office [Farniente] : Le Mont Olympe
Office Phone [] : France telecom ?
Home Phone [] : Aie mes factures
```

sommaire

- Supprimer un utilisateur (userdel)

La suppression d'un compte utilisateur se décompose en deux phases :

1. La suppression de l'utilisateur dans les fichiers de configuration (/etc/passwd, /etc/group ...)
2. La suppression du répertoire et des fichiers de l'utilisateur.

la commande **userdel** permet de faire soit la première étape soit de réaliser les deux d'un coup.

Pour supprimer l'utilisateur ernest des fichiers de configuration du système, utilisez la commande suivante :
[root@mistra /]# userdel ernest

Pour supprimer d'un coup l'utilisateur et son répertoire (ici /home/ernest), utilisez la commande suivante :
[root@mistra /]# userdel -r ernest

- Affichage des dernières lignes ou des premières lignes d'un fichier

La commande **tail** est tout simplement inévitable.
Elle permet d'afficher les dernières lignes d'un fichier. Jusque là on pourrait se dire qu'après tout il suffit d'éditer le fichier et de se déplacer à la fin. D'une part c'est une méthode fastidieuse mais d'autre part, l'option **-f** va définitivement vous convaincre de l'utiliser :
L'option **-f** demande à **tail** de ne pas s'arrêter lorsqu'elle a affiché les dernières lignes du fichier et de continuer à afficher la suite du fichier au fur et à mesure que celui-ci grossit jusqu'à ce que l'utilisateur interrompe la commande avec la combinaison de touches d'interruption **Ctrl-c**.

Les deux grands cas classique de l'utilisation de **tail** avec l'option **-f** est le suivi des fichiers de log */var/log/secure* et */var/log/messages*. Le premier fichier permet de surveiller les connexions que peuvent effectuer d'autres utilisateurs sur votre machine et le deuxième fichier permet de connaître les différents événements qui se produisent sur le système (impression, connexion à l'Internet, tâche de maintenance système...) :

```
[root@mistra /]# tail -f /var/log/messages
Apr 26 14 : 34 : 39 mistra kernel :  PPP line discipline registered.
```

```
Apr 26 14 : 34 : 39 mistra kernel :  registered device ppp0
Apr 26 14 : 34 : 40 mistra pppd[26252] :  pppd 2.2.0 started by root,
uid 0
Apr 26 14 : 34 : 41 mistra chat[26254] :  send (ATZ^M)
Apr 26 14 : 34 : 41 mistra chat[26254] :  expect (OK)
Apr 26 14 : 34 : 43 mistra chat[26254] :  ATZ^M^M
Apr 26 14 : 34 : 43 mistra chat[26254] :  OK -- got it
```

Ici, on voit le déroulement d'une connexion à l'Internet.

la commande **head** réalise la même chose que tail mais elle affiche les premières lignes du fichier au lieu d'afficher les dernières. **tail** et **head** ont une option commune qui permet d'afficher le nombre de ligne que l'on souhaite :
"tail -5 nom_du_fichier" affichera les 5 dernières lignes du fichier

"head -15 nom_du_fichier" affichera les 15 premières lignes du fichier.

Par défaut, **tail** et **head** affichent 10 lignes.

sommaire

- Utilisez votre cdrom, votre lecteur de disquette ... etc .. (mount)

La commande **mount** est utilisée par linux dès son démarrage. Elle permet de monter une système de fichier, c'est-à-dire de le rendre accessible. Ce montage est parfois effectué automatiquement grâce au fichier de configuration /etc/fstab. Ce fichier contient tout ce que linux doit monter lors de son démarrage.
Une question souvent posée dans les forums est "comment puis-je lire un CD-ROM ou une disquette". Il faut d'une part créer un point de montage, puis monter le medium et enfin savoir le démonter si on veut pouvoir en mettre un autre.

- **Créer un point de montage**

  Créer un point de montage signifie tout simplement créer un répertoire où l'on pourra à chaque fois qu'on le souhaite regarder le contenu d'un CD-ROM. Le plus souvent ce répertoire est créé dans le répertoire /mnt. Pour ma part je l'ai monté dans la racine et je l'ai appelé tout simplement cdrom :
  [root@mistra /]# mkdir /mnt/cdrom

- **Monter le cdrom :**
  La première chose à connaître est le nom du fichier spécial qui correspond à votre cdrom. Les fichiers spéciaux sont ces fameux fichiers "device" ("dipositif" en francais ...) que l'on trouve dans le répertoire /dev. C'est en quelque sorte des drivers.
  Les lecteurs IDE commencent par les lettres "hd" alors que les lecteurs scsi commence par les lettres "sd". Si vous avez deux lecteurs IDE (un disque dur et un cdrom par exemple), le disque dur s'appelera normalement hda et le cdrom hdb. Si par exemple le disque dur contient 4 partitions, la première

s'appelera hda1, la deuxième hda2 , etc ...
Donc logiquement si vous êtes dans la situation classique où vous possédez un disque dur et un cdrom, la commande suivante vous permettra de monter le cdrom sur le point de montage /mnt/cdrom :
[root@mistra /]# mount -t iso9660 /dev/hdb /mnt/cdrom

iso9660 : est le type de formatage du support : pour les cdrom c'est le format "iso9660", pour une disquette MS-DOS, c'est le format "ms-dos", "hpfs" pour une partition OS/2 et pour linux c'est le format "ext2", etc

- **/dev/hdb** est le "device" du cdrom
- **/mnt/cdrom** est le point de montage.

Vous n'avez plus qu'à vous déplacer dans le répertoire /mnt/cdrom et lister le contenu de ce répertoire.

- **Démonter un cdrom : umount**

  Pour changer de CD-ROM, il ne suffit pas d'appuyer sur le bouton eject du lecteur, de changer le CD-ROM et de relister le contenu du point de montage. Il faut d'une part démonter le CD-ROM en place pour ensuite le remplacer par un autre qui devra lui même être "monté" de la manière qui a été expliquée au point 2. La commande pour démonter le cdrom est :
  [root@mistra /]# umount /mnt/cdrom

  Ne restez pas dans le répertoire /mnt/cdrom pour le faire, soyez par exemple à la racine.

- Mettre à jour le cache et les liens des bibliothèques (ou comment évitez les "can't load lib..." au démarrage d'un logiciel)

Linux fonctionne maintenant avec un système de bibliothèques dynamiques. Les logiciels utilisant la même bibliothèque pourront accéder tous les deux à la même copie placée en mémoire, ce qui permet un gain de mémoire important.

Il vous est peut-être déjà arrivé d'avoir un problème au lancement d'un logiciel avec un message d'erreur qui peut revêtir cette forme :
"can't load libXpm.so.4.7"

Il vous faudra donc récupérer et installer la bibliothèque manquante sur votre système. Mais une fois installée, la bibliothèque devra être signalée au système. La commande **ldconfig** permettra de mettre à jour les liens symboliques des bibliothèques et des caches.
Vous pourrez enfin lancez votre application normalement.

[root@mistra /etc]# ldconfig -v

sommaire

- Arrêter le système : la commande shutdown

Je ne vous montrerai que les deux options que j'utilise sous linux :
[root@mistra /root]# shutdown -r now
Cette commande vous permet de rebooter l'ordinateur.

[root@mistra /root]# shutdown -h now
Cette commande vous permet d'arrêter complètement le système. Vous pouvez éteindre l'ordinateur lorsque vous verrez affiché :
"System halted
The system is halted"

- Voilà, c'est fini mais comment puis-je en savoir plus sur les commandes ?

La commande "**man**" est là pour vous aider. Toutes les commandes possèdent une "page de manuel" qui vous est livrée avec linux :
[delcros@mistra delcros]$ man cp

Et vous obtiendrez toute la documentation de **cp**.
Pour quitter la page de manuel, vous pouvez appuyer à n'importe quel moment sur la touche "**q**".

sommaire

---

5. Bibliographie

- Les "man pages" (une version française existe)
- Les bases de l'administration système d'AEleen Frisch (traduction de Céline Valot), aux éditions O'Reilly (France)
- "Linux in a Nutshell", "A Desktop Quick Reference" de Jessica Perry Hekman, aux éditions O'Reilly (USA)
