Shell Linux

Commandes de bases

Table des matières

Systèmes de gestion de fichiers (Point de vue utilisateur)

1. Le système de gestion de fichiers

Le système de fichiers racine (root file system), soit le système de fichiers primaire est associé au répertoire le plus haut **/** :

/bin commandes binaires utilisateur essentielles (pour tous les utilisateurs)

/boot fichiers statiques du chargeur de lancement

/dev fichiers de périphériques

/etc configuration système spécifique à la machine

/home répertoires personnels des utilisateurs (optionnel)

/lib bibliothèques partagées essentielles et modules du noyau

/mnt point de montage pour les systèmes de fichiers montés temporairement

/proc système de fichiers virtuel d'information du noyau et des processus

/root répertoire personnel de root (optionnel)

/sbin binaires système (binaires auparavant mis dans /etc)

/sys état des périphériques (model device) et sous-systèmes (subsystems)

/tmp fichiers temporaires

2. Commandes de bases Linux

Le but de cette partie est la prise en main des commandes de base de l'environnement Linux.

a) Introduction

- Qu'est-ce que le **shell** ?

C'est l'interpréteur de commandes (l'interface) entre l'utilisateur et le système d'exploitation, d'où son nom anglais « shell », qui signifie « coquille ».

Le shell est ainsi chargé de faire l'intermédiaire entre le système d'exploitation et l'utilisateur grâce aux lignes de commandes saisies par ce dernier. Son rôle consiste ainsi à lire la ligne de commande, interpréter sa signification, exécuter la commande, puis retourner le résultat sur les sorties.

Chaque utilisateur possède un shell par défaut, qui sera lancé à l'ouverture d'une invite de commande. Le shell par défaut est précisé dans le fichier de configuration **/etc/passwd** dans le dernier champ de la ligne correspondant à l'utilisateur. Il est possible de changer de shell dans une session en exécutant tout simplement le fichier exécutable correspondant, par exemple : **/bin/bash**

b) Commandes pour débuter

Avant de commencer, il faut savoir que Linux est **sensible à la casse** (*case sensitive* en anglais), c'est à dire qu'il distingue les majuscules des minuscules. Ainsi, pour créer un répertoire, la commande est 'mkdir', ce n'est pas la peine d'essayer MKDIR ou mKdiR, cela ne fonctionnera pas. De même, les noms de fichiers et de répertoires sont également sensibles à la casse.

De plus, sous Unix, les chemins sont séparés par des slash : /etc//init/xfs mais jamais etc\init\xfs.

Répertoires spéciaux :

. représente le répertoire courant,

.. représente le répertoire parent

~ représente le répertoire maison (home) de l'utilisateur

Fichiers cachés :

sous Unix, les fichiers cachés commencent par un point. Par exemple, ~/.bashrc est un fichier caché, dans le répertoire maison de l'utilisateur, qui contient la configuration de son shell.

Jokers : ? et *

Les caractères ? et * dans les noms de fichiers et de répertoires permettent de représenter des caractères quelconques. '?' représente un seul caractère, tandis que '*' en représente un nombre quelconque.

Par exemple, « *.jpg »représente tous les fichiers se terminant par jpg ; « *toto* »tous les fichiers contenant« toto ».

Il faut également savoir que c'est le shell qui interprète ces caractères avant de transmettre la ligne de commande.

Par exemple, si vous tapez :

```
rm f *.tmp
```

le shell transformera cette ligne de commande en :

```
rm truc1.tmp truc2.tmp truc3.tmp
```

c) *Commandes*

Une commande est l'exécution d'un programme dans l'interprète (**Shell**). Elle prend en entrée des options et/ou des paramètres. Elle peut renvoyer de l'information à l'écran ou dans un fichier, modifier un fichier, ou produire un message d'erreur.

Une description de toutes les commandes est disponible avec la commande **man** ou **help**. N'hésitez pas à l'utiliser.

voici les commandes de base sous Linux :

Commandes linux	équivalent MsDos	à quoi ça sert	Exemples :

`cd`	`cd`	change le répertoire courant.	`cd ..`
			- va dans le répertoire parent du répertoire courant
			
			`cd /home/user/.nsmail`
			- va dans le répertoire désigné

`ls`	`dir`	affiche le contenu d'un répertoire	`ls`
			- affiche le contenu du répertoire courant
			
			`ls -l`
			- affiche le contenu du répertoire courant de manière détaillée
			
			`ls -a /home/user`
			- affiche le contenu du répertoire désigné (ainsi que les fichiers cachés)

`cp`	`copy`
	`xcopy`
		copie un ou plusieurs fichiers	`cp toto /tmp`
			- copie le fichier toto dans le répertoire `/tmp`
			
			`cp toto titi`
			- copie le fichier `toto` sur le fichier `titi`
			
			`cp -R /home/user /tmp/bak`
			- copie le répertoire `/home/user` ainsi que tout ce qu'il contient dans `/tmp/bak`

`rm`	`del`	efface un ou plusieurs fichiers	`rm toto titi`
			- efface les fichiers `toto` et `titi`
			
			`rm -f toto titi`
			- efface les fichiers `toto` et `titi` sans demander confirmation

`rm -rf`	`deltree`	efface un répertoire et son contenu	`rm -rf /tmp/*` - efface (sans demander de confirmation) tous les fichiers et répertoire de `/tmp`
`mkdir`	`md`	crée un répertoire	`mkdir /home/user/mes documents` - crée le répertoire `"mes documents"` dans le sous répertoire `/home/user`
`rmdir`	`rm`	efface un répertoire s'il est vide	`rmdir /home/user/.nsmail` - efface le répertoire `.nsmail` de `/home/user` si celui-ci est vide

`mv`	`ren`
	`move`
		déplace ou renomme un ou des fichiers	`mv tata titi`
			- renomme tata en titi
			
			`mv * *.bak`
			- ne fonctionne pas !!!!
			
			`mv * /tmp/bak`
			- déplace tous les fichiers du répertoire courant vers le répertoire `/tmp/bak`

`chown`	pas d'équivalent	modifie le propriétaire d'un fichier	`chown user unfichier` rend `user` propriétaire de `unfichier`.
`chgrp`	pas d'équivalent	modifie le groupe proprétaire d'un fichier	`chgrp -R nobody /home/httpd` - rend le groupe : `nobody` (un groupe ayant très peu de droit sur un système linux) propriétaire de `/home/httpd` ainsi que tout les fichiers qu'il contient (`-R`)
`ln -s`	pas d'équivalent	crée un lien vers un fichier	`ln -s /dev/fd0 /dev/disquette` crée un lien vers `/dev/fd0` (le lecteur de disquette) nommé `/dev/disquette`. La manipulation de `/dev/fd0` et `/dev/disquette` (sauf l'effacement).
`grep`	pas d'équivalent	recherche une chaine dans un fichier (en fait recherche une expression régulière dans plusieurs fichiers)	`grep chaine *.txt` - recherche la chaine "`chaine`" dans tous les fichier se terminant par `.txt`.
`which`	pas d'équivalent	trouve le répertoire dans lequel se trouve une commande	`which emacs` - retourne le nom du répertoire dans lequel se trouve la commande `emacs`.
`cat`	`type`	affiche un fichier à l'écran	`cat ~/.bashrc` - affiche le contenu du fichier `~/.bashrc`

`mv`	`ren`
	`move`
		déplace ou renomme un ou des fichiers	`mv tata titi`
			- renomme tata en titi
			
			`mv * *.bak`
			- ne fonctionne pas !!!!
			
			`mv * /tmp/bak`
			- déplace tous les fichiers du répertoire courant vers le répertoire `/tmp/bak`

`find`	`dir -s`	trouve un fichier répondant à certains critères	`find /home -name "*bash*"` - trouve tous les fichiers contenant le mot `bash` dans leur nom se trouvant dans le répertoire `/home`
`locate`	`dir -s`	trouve un fichier d'après son nom	`locate bash` - trouve tous les fichiers contenant le mot `bash` dans leur nom complet (avec le répertoire) : à la différence de `find`, locate trouve ses informations dans une base de donnée créée par `updatedb`
`man`	`help`	affiche l'aide concernant une commande particulière	`man ls` - affiche l'aide (page de `man`uel) de la commnade `ls`. On quitte man en appuyant sur la touche "`q`"

`chmod`	pas d'équivalent	modifie les permissions d'un fichier	`chmod o+r /home/user`
			- autorise les autres (`o=other`) (ie: ceux qui ne sont ni le propriétaire, ni membre du groupe propriétaire) à lire (`r=read`) le répertoire `/home/user`
			
			`chmod a+rw /home/user/unfichier`
			- autorise tout le monde (`a=all`) à lire et écrire (`w=write`) dans le fichier `/home/user/unfichier`

d) Aliases

Plutôt que de taper de longues commandes, ou bien parce que vous préférez vous rappeler d'un nom plutôt que du vrai nom Unix, vous pouvez définir des aliases. Pour ce faire, utilisez la commande **alias** comme suit :

```
alias md=mkdir
alias ls='ls --color'
```

Ainsi pourrez-vous taper md au lieu de mkdir, et affichera une sortie en couleurs...

3. le système de fichier UNIX

a) La commande ls

Cette commande est omniprésente, aussi il est bon d'en présenter les basiques.

Afficher le listing page par page :

ls | less (less est une version améliorée de more)

Afficher le listing en couleurs :

ls --color

Afficher aussi les fichiers cachés (commençant par un point) :

ls -a

Mettre un '/' après les noms de répertoires :

ls -p

Afficher le listing détaillé :

ls -l

Tri sur la date

Pour afficher les fichiers d'un répertoire en triant sur la date de mise à jour des fichiers

Afficher les fichiers les plus récents en premier :

ls -t

Afficher les fichiers les plus vieux en premier :

ls -rt

Mixer avec l'option« l »afin d'afficher le listing détaillé :

ls -rtl *ou* *ls -tl*

Bien sûr, toutes ces options sont mixables, ainsi« ls -altp »affiche tous les fichiers, de façon détaillée, dans l'ordre chronologique, an ajoutant '/' après chaque nom de répertoire.

Exercice :

Dans les systèmes d'exploitation dérives d'Unix, le codage des droits se fait sur 9 bits groupes par 3 bits. Ces droits sont codés en un entier. Pour ce faire, on convient de la correspondance : r = 4; w = 2 et x = 1.

Ainsi, les droits rw-, correspondent à l'entier (r=)4+(w=)2=6. Donc rw-rw-rw- correspond à l'entier 666.

Questions :

1. A quels droits correspondent les entiers 751; 521; 214 et 150 ?
2. Par quels entiers sont codés les droits rw-r- -r- - et rwxr-xr-x ?

4. Editeur de texte (vi)

Un éditeur de texte permet de rentrer du texte dans un fichier afin de le conserver.

vi (prononcez vie-aïe ou [vi:ai]) est l'éditeur de texte de base sous Linux, vous risquez bien d'avoir à vous en servir au plus mauvais moment, c'est à dire lorsque plus rien d'autre ne fonctionne.

a) Lancer VI

Si vous tentez d'ouvrir un fichier inexistant, vi créera ce fichier. Vous pouvez donc créer un nouveau fichier simplement en tapant

```
vi nom_du_fichier
```

b) Les modes de VI

vi possède deux modes : le mode« **Insert** »et le mode **commande**. En mode commande, vous ne pouvez pas insérer de texte dans le fichier, mais les touches du clavier sont autant de touches de commandes. En mode Insert, les touches de commandes (notamment les lettres !) se transforment en vraies lettres que vous pouvez insérer dans le fichier.

Insérer du texte

Lorsque vi s'ouvre, il est en mode commande. Pour passer en mode Insert :

tapez [i] ou [Insert] pour insérer du texte à l'endroit où se trouve le curseur,

tapez [A] pour ajouter du texte à la fin d'une ligne.

En mode Insert, vous pouvez taper du texte, effacer avec la touche [Suppr] ou [Bkspace].

Pour quitter le mode Insert, tapez [Esc].

Remarque : à la suite de votre fichier, vous voyez des lignes vides commençant par le caractère '~'. C'est normal : cela signifie juste que ces lignes sont vides, et les caractères '—' ne seront bien sûr pas enregistrés dans votre fichier.

c) Les commandes

Après avoir quitté le mode Insert, ou avant d'y être entré, les touches du clavier correspondent à

des commandes.

Voici ci-dessous les commandes de base qui vous permettront de vous y retrouver :

:q! [Entrée] pour quitter sans sauver,

:w [Entrée] pour enregistrer,

:wq [Entrée] pour enregistrer et quitter,

x efface le caractère qui se trouve sous le curseur,

dd efface la ligne sur laquelle se trouve le curseur,

:u[Entrée] permet d'annuler (ou :undo).

5. La commande Cat

La commande **cat** constitue un éditeur (très) simplifié. Elle permet également d'afficher le contenu d'un fichier entier à l'écran.

```
cat > fich1 (Enter)
```

Entrer le texte à stocker dans le fichier (Enter) **CTRL D**

6. Redirections

Linux, comme tout système de type Unix, possède des mécanismes permettant de rediriger les entrées-sorties standards vers des fichiers.

Ainsi, l'utilisation du caractère « **>** » permet de rediriger la sortie standard d'une commande située à gauche vers le fichier situé à droite :

```
ls -al /home/hk/ > toto.txt
echo« Toto »> /etc/monfichierdeconfiguration
```

La commande suivante est équivalente à une copie de fichiers :

```
cat toto > toto2
```

La redirection « **>** » a pour but de créer un nouveau fichier. Ainsi, si un fichier du même nom existait, celui-ci sera écrasé. La commande suivante crée tout simplement un fichier vide : **>** fichier

L'emploi d'un double caractère « **>>** » permet de concaténer la sortie standard vers le fichier, c'est-à-dire ajouter la sortie à la suite du fichier, sans l'écraser.

```
cat fich1.txt >> fich2.txt
```

De manière analogue, le caractère « **<** » indique une redirection de l'entrée standard. La commande suivante envoie le contenu du fichier toto.txt en entrée de la commande cat, dont le seul but est d'afficher le contenu sur la sortie standard (exemple inutile mais formateur) : cat **<** toto.txt

Enfin l'emploi de la redirection « **<<** » permet de lire sur l'entrée standard jusqu'à ce que la chaîne située à droite soit rencontrée. Ainsi, l'exemple suivant va lire l'entrée standard jusqu'à ce que le mot STOP soit rencontré, puis va afficher le résultat :

```
cat << STOP
```

Les messages d'erreur peuvent être dirigés séparément dans un fichier avec 2**>** :

```
startx > startx.log 2> startx.err
```

ou dirigés vers le même fichier que les messages normaux :

```
startx > startx.log 2>&1
```

7. Liens physiques et liens symboliques

a) Les liens physiques

Le système identifie les fichiers (physiquement) par un identificateur unique qui s'appelle le numéro d' i-noeud.

L'i-noeud est en fait un numéro unique qui identifie le fichier :

Par exemple, pour le fichier /root/anaconda-ks.cfg si vous faites :

```
# ls -i /root/anaconda-ks.cfg
```

vous obtiendrez son numéro i-noeud.

Pour créer un autre lien, on utilise la commande ln :

```
# ln /root/anaconda-ks.cfg /root/anaconda
```

Vérification:

```
# ls -i /root/anaconda-ks.cfg /root/anaconda
```

Nous obtenons le même numéro

La commande ls -l indique le nombre de liens que comporte un fichier : c'est le chiffre venant après les permissions.

b) Les liens symboliques

Cette sorte de lien permet de donner un autre nom au fichier, mais n'utilise pas l'i-noeud physique du fichier.

Pour créer un lien symbolique, il suffit de passer l'option **-s** à la commande **ln** :

```
# ln -s /root/anaconda-ks.cfg /root/anaconda
```

Cela va créer un fichier /root/anaconda avec un autre i-noeud :

Vérifiez avec :

```
# ls -i /root/anaconda-ks.cfg /root/anaconda
```

Nous avons dans ce cas 2 numéros différents.

```
# ls -l /root
```

Indique que anaconda pointe (→) bien sur anaconda-ks.cfg

Avec le lien symbolique, les permissions de /root/anaconda-ks.cfg seront les mêmes que pour /root/anaconda.

Il est donc facile d'identifier le lien symbolique d'un fichier avec la commande ls -l, alors qu'il n'en va pas de même pour un lien physique.

8. Compression de fichiers : Gzip, Zip, et Tar

a) Compression avec Gzip et Zip

Les fichiers comprimés utilisent moins d'espace disque et se téléchargent plus rapidement que les grands fichiers non comprimés. Vous pouvez comprimer les fichiers Linux à l'aide de l'instrument de compression open-source Gzip ou Zip, qui est reconnu par la plupart des systèmes d'exploitation.

Par convention, les fichiers comprimés se voient attribuer l'extension .gz. La commande Gzip crée un fichier comprimé terminant par .gz; Gunzip extrait les fichiers comprimés et efface le fichier .gz.

Pour comprimer un fichier, entrez la commande suivante à l'invite du shell :

```
gzip filename.ext
```

Le fichier sera comprimé et sauvegardé comme filename.ext.gz.

Pour décompresser un fichier comprimé, tapez :

```
gunzip filename.ext.gz
```

Le filename.ext.gz est effacé et remplacé par filename.ext.

Si vous échangez des fichiers avec des utilisateurs non LINUX, vous devriez utiliser zip pour éviter les problèmes de compatibilité. Linux peut facilement ouvrir des fichiers zip ou gzip, mais les systèmes d'exploitation non-Linux pourraient avoir des problèmes avec les gzip.

Pour comprimer un fichier à l'aide de zip, entrez ceci :

```
zip -r filename.zip files
```

Dans cet exemple, filename représente le fichier que vous créez, et files représente les fichiers que vous voulez placer dans le nouveau fichier :

Pour extraire le contenu d'un fichier zip, entrez :

```
unzip filename.zip
```

Vous pouvez compresser plusieurs fichiers en même temps avec zip ou gzip. Énumérez les fichiers en les séparant par un espace :

```
gzip filename.gz file1 file2 file3 /user/work/school
```

La commande ci-dessus compresse les file1, file2, file3, et le contenu du répertoire /user/work/school pour les placer dans filename.gz.

b) Archiver avec Tar

Les fichiers **tar** placent plusieurs fichiers ou le contenu d'un répertoire ou de plusieurs répertoires dans un seul fichier. Il s'agit d'une bonne manière de créer des sauvegardes et des archives. Généralement, les fichiers tar terminent par l'extension .tar.

Pour créer un fichier tar, tapez :

```
tar -cvf filename.tar files/directories
```

Dans cet exemple, filename.tar représente le fichier que vous créez et files/directories représente les fichiers ou répertoires que vous voulez placer dans le nouveau fichier.

Vous pouvez utiliser des noms d'accès absolus ou relatifs pour ces fichiers et répertoires. Séparez les noms de fichiers et de répertoires par un espace.

La saisie suivante créera un fichier tar en utilisant un nom d'accès absolu :

```
tar -cvf foo.tar /home/mine/work /home/mine/school
```

La commande ci-dessus placera tous les fichiers dans les sous-répertoires /work et /school dans un nouveau fichier appelé foo.tar dans le répertoire dans lequel vous travaillez actuellement.

Pour afficher la liste du contenu d'un fichier tar, entrez :

```
tar -tvf foo.tar
```

Pour extraire le contenu d'un fichier tar, entrez :

```
tar -xvf foo.tar
```

Cette commande n'élimine pas le fichier .tar, mais elle place des copies du contenu de .tar dans le répertoire dans lequel vous travaillez actuellement.

La commande tar ne compresse pas automatiquement les fichiers. Vous pouvez compresser les fichiers tar avec :

```
tar -czvf foo.tar
```

Les fichiers tar compressés se voient attribuer l'extension .tgz et sont comprimés avec gzip.

Pour décompresser un fichier tar, entrez :

```
tar -xzvf foo.tgz
```

9. Voir un fichier (cat et more)

a) cat

La commande cat permet de lire des fichiers. Nous avons vu dans le TD1 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 :

```
cat .bashrc # .bashrc
```

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).

```
cat -n .bashrc
```

Si vous souhaitez connaître les autres options de cat, tapez au prompt« cat –help ».

b) more

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.

10. Les commandes head et tail

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.

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 lignes 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.

11. La commande find

La commande find permet de trouver des fichiers dans un répertoire.

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

```
find directory -name targetfile -print
```

Par exemple :

```
find / -name linux-test2 -print /root/linux-test2
```

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 que nous voulons spécifier le nom d'un fichier ;

« -print » demande à find d'afficher le résultat.

Un peu long n'est ce pas pour trouver la réponse dans tout cette grosse arborescence ? En général on recherche rarement un fichier depuis la racine.

Pour chercher tous les fichiers commençant par« linux-tes »et définir à partir de quel répertoire on souhaite effectuer la recherche on utilise cette syntaxe :

```
find /home/user -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 :

```
find /usr -type d -name bin -print
```

Ici, on demande à find de trouver les répertoires (l'argument« d »(comme« directory") 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 :

```
find -name 'linux-tes*' -print -ok rm {} \;
./linux-test
rm ... ./linux-test ? y
```

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'échappement.

a) Exercice :

Dans le répertoire d'accueil, créez un répertoire **rep**. Dans ce sous répertoire, créez trois fichiers toto.c, tata.h et lala.o.

1. A partir du répertoire d'accueil, en utilisant la commande **find** affichez tous les fichiers du répertoire **rep** commencent par la lettre t.
2. De même, toujours à partir du répertoire d'accueil, en utilisant la commande find, affichez tous les fichiers du répertoire **rep** de type .c

12. 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, cette base de donnée se trouve dans le répertoire /usr/lib et se nomme locatedb.

La syntaxe est donc simple:

```
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.

13. La commande which

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

```
which ls /bin/ls
```

14. La commande file

La commande file permet d'afficher le type du fichier (exécutable, répertoire, ASCII, ...)

```
file /bin/ls
file /etc/passwd file /usr
```

15. Trouver du texte : commande grep

La commande grep est un pivot des commandes UNIX. Elle cherche une expression rationnelle dans un ou plusieurs fichiers, exemple :

```
grep ip_tables /var/log/messages
```

grep est la commande qui vous fouille les fichiers

La commande a donc affiché la ligne qui contient le mot« ip_tables »dans le fichier /var/og/messages.

La richesse de la commande grep permet de faire des recherches sur plusieurs fichiers et d'avoir un format de sortie adéquat.

Le fichier /var/og/messages est déjà assez important et il serait agréable de savoir où se trouve cette ligne qui contient le mot Netfilter dans le fichier :

```
grep -n ip_tables /var/log/messages
```

Une autre option très utile est -l qui permet de n'afficher que les noms des fichiers contenant ce que l'on cherche :

```
grep -l ip_tables /var/log/*
```

Quelques-unes des autres options :

- -c donne le nombre de fois où l'expression rationnelle a été rencontrée dans le fichier :

```
grep -c fouille linux-commande.html
```

- -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 considérera comme une option !

16. La commande wc

La commande wc (word count) lit à partir du clavier les caractères, lignes et fichiers, en effectuant le décompte des caractères, des mots et des lignes.

```
wc NomDeFichier
```

- quelle commande utiliser pour afficher le nombre de caractères dans le fichier /etc/passwd ? **wc –c fichier**
- quelle commande utiliser pour le nombre de mots dans le fichier /etc/passwd ? **wc –w fichier**
- quelle commande utiliser pour le nombre de lignes dans le fichier /etc/passwd ? **wc –l fichier**

17. La commande diff

diff nous permet de découvrir les différences entre deux versions d'un fichier. Pour situer les différences dans leurs contextes, on vous conseille d'utiliser l'option -c.

```
diff [-c] fichier fichier
diff -c prog.c old_prog.c
```

Les lignes marquées par un point d'exclamation ! sont les lignes qui ne sont pas identiques.

Parce diff risque d'afficher beaucoup plus de lignes que l'écran (ou fenêtre) puisse accommoder, on fait souvent un tube (pipe) qui dirige les résultats de diff à la commande more. Par exemple:

```
diff -c prog.c old_prog.c | more
```

18. Redirection entrée/sorties et pipes

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 claviers français classiques. 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 que les lignes qui contiennent le mot« bash »pour m'éviter d'avoir à parcourir toute la longue liste qu'affiche ps aux :

```
ps aux | grep 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 finit par ne plus s'en passer.

19. 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.

```
df -h
```

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

```
du -h
```

Contrôle des processus

Il se peut qu'une commande ne termine pas ou dure trop longtemps. Cela arrivera certainement par erreur durant les travaux dirigés de programmation !

Il y aussi des commandes comme xterm qui ne se terminera que si l'on termine son propre shell (par exit).

Enfin certaines commandes sont volontairement conçues pour ne pas terminer, comme yes ou xeyes, par exemple.

Normalement, l'interpréteur de commandes attend la fin de la commande en cours avant d'en accepter une nouvelle et il se trouve donc bloqué dans les situations envisagées ci-dessus. Heureusement, on dispose de plusieurs moyens d'action sur la commande en cours.

Le plus brutal, est Ctrl-c (Control c) qui interrompt définitivement la commande, sans possibilité de reprise.

On peut aussi suspendre la commande courante par Ctrl-z.

Une commande que l'on vient de suspendre peut être relancée par la commande fg (comme foreground).

On peut également la relancer en arrière-plan par la commande bg (comme background). Une commande qui s'exécute en arrière-plan laisse l'interpréteur de commande disponible.

Une commande peut être directement lancée en arrière-plan en ajoutant un « & » à la fin.

Par exemple, la commande :

```
ping 8.8.8.8 > /dev/null &
```

est équivalente à la séquence suivante :

```
ping 8.8.8.8 > /dev/null
Ctrl-z
bg
```

Et si on a lancé une commande comme sleep 1000 &, que peut-on faire ? Il est clair que Ctrl-c ou Ctrl-z n'agissent que sur la commande de premier-plan.

Et, comme on peut avoir plusieurs commandes en arrière-plan, il faut pouvoir désigner celle sur laquelle on veut agir. On lance alors la commande ps qui nous donne le pid du processus concerné que l'on utilise ensuite dans une commande kill appropriée, voir ps et kill ci-dessous.

On remarquera que l'on peut ainsi tuer son shell... mais on s'en lasse vite !

Les commandes indispensables :

```
Ctrl-c
```

interrompt définitivement la commande de premier plan, sans possibilité de reprise.

```
Ctrl-z
```

suspend la commande de premier plan. Elle peut être relancée par les commandes fg ou bg.

```
fg
```

La commande fg permet de relancer au premier plan la dernière commande, qu'elle soit suspendue ou non (cas d'une commande d'arrière-plan).

```
bg
```

La commande bg permet de relancer en arrière-plan la dernière commande suspendue.

```
ps
```

La commande ps, utilisée sans argument, liste tous les processus attachés au terminal. Chaque ligne affichée commence par le PID qui est le numéro qui identifie chaque processus dans le système. On repère ainsi le pid d'un processus que l'on pourra supprimer par la commande kill.

```
kill
```

La commande kill s'utilise en général avec une option qui indique l'action voulue et prend en paramètre le pid du processus concerné. On utilise surtout :

```
kill -KILL pid
```

qui interrompt définitivement

ou

```
kill -STOP pid
```

qui suspend. On peut relancer ensuite grâce à

```
kill -CONT pid
```

a) Une commande instructive : « top »

La commande top permet d'observer la vie d'un système Unix et de détecter les processus« gourmands ». Elle 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 démarrage).

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 processus de manière interactive en tapant k. top demande ensuite quel signal il doit envoyer : 15 (SIGTERM) est le signal par défaut qui met fin à un processus, 9 (SIGKILL) est plus brutal.

Pour quitter top, appuyer simplement sur la touche« q ».

b) La commande « ps »

La commande ps permet de connaître les processus actifs à un moment donné :

```
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 indiqué les processus associés à un terminal et qui dépendent de l'utilisateur courant (ici « ser »).

En fait, il est tout a fait probable que d'autres processus non liés à un terminal aient été lancés par « user » .

Les commandes qui ne sont pas associées à un terminal sont reconnaissable par le point d'interrogation qui remplit le champ 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.

20. 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 numéro du PID de pppd et ensuite si par exemple le PID est 592, je peux tuer la connexion en faisant :

```
kill 592
```

Il faut être 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 :

```
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..