Manipulations en ligne de commandes

L'administrateur sur un serveur utilise la ligne de commandes pour ses manipulations de fichiers textes, scripts, périphériques, etc. Certaines techniques s'apparentent aux figures imposées du patineur artistique : peu valorisantes et attractives, elles sont néanmoins nécessaires pour maîtriser son métier. Pour suivre l'esprit de l'ouvrage résolument tourné vers la distribution Ubuntu et non sur l'apprentissage du système Linux, les paragraphes suivants ne présenteront qu'une base de ces techniques.

1. Expressions régulières

Une expression régulière (ou rationnelle) décrit de façon générique une chaîne de caractères contenue dans un fichier ou nom de fichier. En fait, on peut simplifier en disant qu'une expression régulière est une manière plus compacte (mais pas forcement plus lisible...) de définir un modèle (pattern) de chaîne de caractères. Les objectifs s'identifient comme une demande de sélection suivant ce critère, ou d'effectuer des traitements comme des substitutions sur les chaînes trouvées. Leur utilisation s'étend à partir de certains filtres `shell` (`grep`, `sed`, `awk`...) à des langages de scripts pour administrateurs : `perl`, `python`...

Attention ! Certains caractères spéciaux ressemblent aux caractères génériques de désignation de fichiers (caractères d'englobements) : elles ne se confondent pas avec les 'jokers' ou caractères spéciaux du `shell`. Même s'il existe des similitudes, ils ont une interprétation différente.

a. Expressions régulières atomiques ou ERA

L'ERA représente un seul caractère du type `ch` (type normal) ou `sp` (type spécial). Concrètement, si vous voulez rechercher toutes les lignes d'un fichier qui contient la chaîne de caractères coucou (qui n'est, après tout, qu'une succession de caractères, soit `c o u c o u`), vous aurez la syntaxe suivante :

```
(commande de recherche) 'coucou' [nom_de_fichier]
```

Cela affiche les lignes d'un fichier contenant le mot "coucou". Vous pouvez aussi utiliser une ERA avec un type spécial :

`^` : indique le début d'une ligne.

`$` : indique la fin d'une ligne.

`.` : indique tout caractère sauf nouvelle ligne (new line ou retour chariot).

`*` : indique la répétition de 0 à n fois du caractère.

`[]` : indique un ensemble de caractères.

`[^]` : correspond à l'un des caractères non inclus dans l'ensemble.

`\<` : correspond au début de mot.

`\>` : correspond à la fin de mot.

`\` : enlève la signification spéciale du caractère suivant.

Exemples :

`i` : le caractère i.

`.` : un caractère quelconque.

`\.` : le caractère . : ce n'est plus un caractère spécial car précédé de \.

`[aceg]` : soit `a`, `c`, `e` ou `g`.

`[^aceg]` : tous les caractères autres que `a`, `c`, `e` ou `g`.

`[a-f]` : tout caractère compris entre `a` et `f` suivant l'ordre alphabétique.

`[^a-f]` : tout caractère n'appartenant pas à l'intervalle `a-f`.

\<p : tous les mots commençant par p.

e\> : tous les mots finissant par p.

Certains caractères comme $, *, [etc. ont une signification particulière pour le shell : vous devez donc mettre en quotes l'expression régulière.

b. Expressions régulières simples ou ERS

L'ERS s'obtient en juxtaposant (concaténation) des ERA. L'ERA de droite s'applique à l'ERA de gauche. Dans certains cas, l'ERA se "multiplie" par d'autres expressions :

x? : recherche dans la chaîne une fois au maximum le caractère représenté par x.

x* : recherche dans la chaîne 0 ou plus le caractère représenté par x.

x+ : recherche dans la chaîne au moins une fois le caractère représenté par x.

x\|y : recherche dans la chaîne le caractère représenté par x ou y.

x\{n\} : recherche dans la chaîne précisément n fois le caractère représenté parx.

x\{n,\} : recherche dans la chaîne contenant au moins n fois le représenté par le caractère x.

x\{,m\} : recherche dans la chaîne 0 ou m fois le caractère représenté par x.

x\{n,m\} : recherche dans la chaîne de n à m fois le caractère représenté par x.

Exemples :

Prenons comme test le fichier texte (les numéros de lignes ne font pas partie du fichier, elles sont là pour identifier les lignes) suivant :

```
1    mars
2
3    le marsupilami
4    marsupial
5    barre mars
6    martinet 1515
7    une mare
8    bien
9    mar
```

'mars' : chaîne mars (retourne les lignes 1,3,4 et 5).

'mars.' : chaîne mars suivie de n'importe quel caractère (lignes 3 et 4).

'mars*' : chaîne mar, ou mars ou marss ou marsss... (toutes sauf la 2 et 8).

Rappel : * n'a pas la même signification qu'en shell. Cet ERA s'applique au caractère de gauche, le s et demande un nombre n de s.

'[mars]' : lignes contenant (sans exclusive) des m, a, r, s (toutes sauf la 2 et 8).

'[^mar]' : lignes ne contenant pas exclusivement des m, a, r (toutes sauf la 2).

'[r]\{2\}' : lignes où les chaînes (mots) ayant 2 r se suivant (ligne 5).

Notez le \ enlevant la signification spéciale des accolades. On peut spécifier un opérateur logique de cette façon :

'rs$\|al$' : lignes se terminant par rs **ou** al (lignes 1,4,5).

c. Expressions régulières étendues ou ERE

L'ERE s'obtient en juxtaposant (concaténation) des ERA ou des ERS.

Exemples d'ERE sur le même fichier :

`'[^a-zA-Z]'` : chaînes contenant a contrario des espaces ou des chiffres (lignes 3,5,6,7).

`'^mar'` : lignes commençant par `m`, `a`, ou `r` (lignes 1,4,6).

`'mars$'` : même chose mais en fin de ligne (1,5) avec `s`.

`'[0-9]'` : désigne un chiffre quelconque (6).

`'^[^m]'` : ligne ne commençant pas par un `m` (3,5,7,8).

`'ar\{2,\}'` : ligne contenant au moins deux fois `ar` (5).

d. Utilisation des expressions régulières avec grep

Cette commande se décline en fait sur trois niveaux :

- `grep` utilise les expressions régulières simples (par défaut `grep -G`).
- `egrep` utilise les expressions régulières étendues (ou `grep -E`).
- `fgrep` n'utilise pas d'expression régulière mais recherche une chaîne littérale (ou `grep -F`).

Syntaxe :

```
grep [-options] expression [fichier]
```

Avec `grep`, les motifs doivent être saisis entre guillemets (exemple : "UnMotif") ou entre apostrophes si le motif contient un "$" (exemple : '$UnMotif'). Ces commandes disposent aussi d'options (voir le manuel en ligne pour plus de détails).

Exemples d'utilisation :

sans expressions régulières

```
grep 'es' fichier.txt       # toutes les lignes contenant 'es'
grep -n 'es' fichier.txt    # toutes les lignes numérotées contenant
                            # 'es'
grep -i 'il' fichier.txt    # toutes les lignes contenant 'il' en
                            # minuscules ou majuscules
grep -l 'Z' *               # tous les noms de fichiers contenant 'Z'
```

avec expressions régulières

```
grep 'ss.' fichier.txt      # toutes les lignes contenant ss plus un
                            # caractère quelconque
grep '[0-9]' fichier.txt    # toutes les lignes contenant un
                            # caractère numérique
grep '^print' fichier.pl    # toutes les lignes contenant print en
                            # début de ligne
grep '^[^a-z]' fichier.txt  # les lignes ne contenant pas des
                            # minuscules en début de ligne
```

Exemple montrant les caractères spéciaux dans une expression régulière :

```
grep '\\home\\a[a-z]*' test.txt
```

Si on suppose que `test.txt` contient un listing de noms de fichiers avec leur répertoire, la commande retournera les fichiers contenant la séquence `\home\`, commençant par `a` et suivi de zéro ou plusieurs minuscules `alphonse`, `albert`, `aurélien`, `andré`...

En décomposé :

- on a \\ qui donne \ pour le *back slash* ou symbole de division inversé
- home pour ... home
- \\ pour encore le *back slash*
- a pour la lettre ... a
- [a-z] pour les minuscules avec * s'appliquant donc zéro ou plusieurs minuscules appartenant à l'intervalle.

2. Commandes d'édition ou filtres de fichiers

Pour voir ces quelques commandes (il ne sera pas exposé ici les commandes sed et awk), la base de travail portera sur le fichier texte neveux.txt contenant les lignes :

```
riri:rouge:1
fifi:vert:2
loulou:bleu:3
```

La syntaxe de ce fichier se rapproche de celle de beaucoup de fichiers de données sous Linux, à commencer par un des premiers d'entre eux : le fichier /etc/passwd.

La commande cut

Cette commande coupe en plusieurs champs les lignes d'un fichier ou de l'entrée standard :

```
cut -cliste {position des caractères} fichier
cut -fliste [-dx] {position des champs, x séparateur} fichier
```

Note : liste -n équivaut à 1-n et p- à p-x (ou x représente le dernier).

Avec la commande cut, le premier champ a comme numéro 1, le deuxième 2 est ainsi de suite.

Exemples :

```
cut -c10 neveux.txt
  # Affichage en colonne du 10éme caractère de chaque ligne e,:,e
cut -c1-4 neveux.txt
  # Affichage en colonne des caractères entre le 1er et 4ème caractère
  # de chaque ligne riri, fifi, loul
cut -f2 -d: neveux.txt
  # Affichage en colonne du deuxième champ de toutes les lignes avec
  # comme séparateur les deux points rouge,vert,bleu
cut -f1,3 -d: neveux.txt
  # Affichage en colonne du premier et troisième champ de toutes les
  # lignes avec comme séparateur les deux points
```

La commande wc

Cette commande assez simple, compte le nombre de lignes, mots et caractères dans un fichier :

```
wc [-wcl] fichier
```

On a l'option -w pour les mots, -c pour les caractères et -l pour les lignes.

Exemple :

```
wc -wcl neveux.txt
  # L'ensemble donne : 3  3  39 neveux.txt. L'ordre de sortie est
  # toujours ligne, mot et caractère, séparateur du mot : l'espace
```

La commande sort

Cette commande lit mais surtout tri le contenu d'un fichier. Par défaut le tri porte sur un seul champ (premier caractère de la ligne) selon l'ordre lexicographique (table ASCII) :

```
sort [-options] [+pos1 [-pos2]] fichier
```

Les principales options :

`-d` : trie par ordre alphanumérique (caractères, chiffres et espace).

`-o` : fichier_résultat renvoie le résultat du tri dans le fichier indiqué.

`-tx` : donne le séparateur (x) au lieu de l'espace ou tabulation (par défaut).

`-f` : pas de différence entre minuscule et majuscule.

`-u` : supprime les lignes doublons.

`-n` : trie sur des chiffres.

`-r` : renverse l'ordre de tri.

Exemples :

```
sort -o resultat.tri test.txt
  # Tri du fichier selon l'ordre lexicographique, résultat dans
  # resultat.tri
sort -t: +3n -4 -o res.tri /etc/passwd
  # Tri numérique sur le 3ème champ, pos1 et pos2 sont sous la forme
  # numéros de champ : +pos1 enlève du tri les champs de 1 à pos1 et
  # -pos2 les champs qui suivent pos2 (jusqu'à la fin de la ligne si
  # pos2 est omis)
```

Soit sur le fichier `neveux.txt` :

```
sort neveux.txt
  # Affiche le fichier trié par ordre alphabétique
sort -rn -t: +2 neveux.txt
  # Affiche le fichier en sens inverse trié numériquement sur le
  # 3ème champ
```

La commande split

Très utilisée en programmation, la commande `split` coupe un fichier en morceau (en plusieurs fichiers) et ce par nombre de lignes :

```
split -l nombre_de_lignes fichier [fichier1 fichier2 ...]
```

Exemple :

```
split -l 1 neveux.txt
  # Découpe en trois fichiers parce que trois lignes dans xaa, xab,
  # xac par défaut en cas d'absence de noms de fichiers de sortie
```

Il existe la commande inverse `paste` qui regroupe les fichiers par ligne (voir le manuel en ligne pour plus de renseignements).

La commande cmp

La commande `cmp` indique si deux fichiers sont identiques :

```
cmp fichier1 fichier2
```

Si les deux fichiers sont identiques, la commande ne génère aucune sortie. Dans le cas contraire la commande indique la position de la première différence (ligne et caractère).

Exemple d'utilisation :

(on a un autre fichier `neveux1.txt` avec 4 au lieu de 2 en fin de deuxième ligne) :

```
cmp neveux.txt neveux1.txt
```

Retourne :

```
neveux.txt neveux1.txt sont différents: octet 24, ligne 2
```

La commande `diff` recherche aussi les différences entre deux fichiers et `comm` qui affiche les lignes communes (voir le manuel pour plus de renseignements).

La commande `tail`

Cette commande, déjà vue, édite un fichier par la fin, son inverse est `head` :

```
tail -f /var/log/messages
  # Montre la fin du fichier
```

Autre utilisation :

```
tail -50 /var/log/syslog
  # Montre les 50 dernières lignes du fichier
```

La commande `cat`

Cette commande aux multiples usages affiche, crée, copie et concatène des fichiers. Elle s'emploie seule ou avec des commandes spéciales (comme les filtres) en utilisant les mécanismes de redirections. Les principales fonctionnalités de cette commande sont :

- La lecture au clavier : lecture des données au clavier et affichage en écho à l'écran, la fin de saisie par `<ctrl -d>`.

  ```
  cat
    # saisie d'un texte quelconque, fin par <ctrl-d>
  ```

- Copie de fichier : copie du ficher `nifnif.txt` dans le fichier `nafnaf.txt` en utilisant la redirection de la sortie standard (alternative à la commande `cp`).

  ```
  cat nifnif >nafnaf
  ```

- Affichage d'un fichier : affichage à l'écran du contenu du fichier `mechantloup.txt`

  ```
  cat mechantloup.txt
  ```

- Création d'un fichier : création du fichier `noufnouf.txt` avec lecture au clavier par l'entrée standard et redirection de la sortie standard, fin par `<ctrl -d>`.

  ```
  cat >noufnouf.txt
  ```

- Concaténation de fichier : regroupement dans le fichier `maisonbriques.txt` par la redirection de la sortie standard.

  ```
  cat nifnif.txt nafnaf.txt noufnouf.txt >maisonbriques.txt
  ```

La commande `tr`

La commande `tr` convertit une chaîne de caractères en une autre de taille égale. Les options sont les suivantes :

`-c` : les caractères qui ne sont pas dans la chaîne d'origine sont convertis selon les caractères de la chaîne de destination.

-d : destruction des caractères appartenant à la chaîne d'origine.

-s : si la chaîne de destination contient une suite contiguë de caractères identiques, cette suite est réduite à un caractère unique.

Cette commande, plus complexe qu'elle en a l'air, nécessite un fichier (ou une commande par un tube) en entrée standard, sortie à votre convenance. Nous ne l'utiliserons que dans le cas d'une substitution d'éléments. Exemple d'utilisation :

```
tr ":" "#" <neveux.txt neveux.txt
  # Remplace les deux points par le caractère dièse
```

La commande grep

Cette commande très utilisée par l'administrateur, outre son utilisation avec les expressions régulières (voir plus haut), accompagne souvent d'autres commandes dans un tube.

Principales options :

-n : fait précéder chaque ligne d'un numéro de ligne.

-v : affiche toutes les lignes sauf celles contenant "expression".

-i : pas de distinction entre minuscules et majuscules.

-c : affiche le nombre de lignes contenant "expression".

Utilisation de grep avec un tube et la commande wc :

```
grep "le" test.txt | wc -l
  # Compte le nombre de lignes contenant le mot le dans le fichier
```

Ou par exemple pour interroger la base de données des paquetages avec un mot :

```
dpkg -l | grep -i bind
  # dpkg -l retourne tous les paquetages, grep les filtre sans
  # distinction des majuscules/minuscules et affiche tous les
  # paquetages contenant le mot bind
```

3. Entraînement

Toujours pour vous aider dans la maîtrise des commandes, voici une liste d'exercices (les solutions se trouvent toujours en Annexe) :

Cadre de travail

Session ouverte en root sur une version Ubuntu serveur. Reprenez l'exemple du fichier des ERS et ajoutez-lui quelques lignes de façon à ce qu'il contienne ceci :

```
mars

le marsupilami
marsupial
barre mars
martinet 1515
une mare
bien
mar
mar mare
langage
language
Langage
Language
le le le langage gage ge jjjjjjjava jjjaaavvvaaa jaaavvaaaaa jaavvaaaaa
05.06.07.08
0 1 2 3 5 6 7 8 9
0123456789
riri  05-55-35-22-55 23 ans
```

```
fifi  10-23-22-55-63 35 ans
loulou 03-02-21-12-25 99 ans
06/01/03
```

Donnez-lui pour nom `test1.txt`.

Exercices - Partie 1

- Affichez les lignes contenant la chaîne `ar`.
- Affichez les lignes dont au moins une chaîne contient au minimum 3 `j` consécutifs.
- Affichez les lignes ayant au moins une chaîne se terminant par `age`.
- Affichez les lignes ayant au moins une chaîne se terminant par `guage` ou `gage`.
- Affichez les lignes se terminant par `age` ou `ava`.
- Affichez les lignes vides.
- Affichez les lignes contenant les sous-chaînes `35` ET `ans` (vous ferez un "tube").
- Affichez les lignes ne commençant pas par un chiffre.
- Affichez les lignes ne contenant que des chiffres.
- Affichez les lignes commençant par une voyelle ou un chiffre.
- Affichez les lignes ne se terminant pas par un chiffre.
- Affichez les lignes faisant moins de 10 caractères.

Voici maintenant un autre fichier nommé `test2.txt` :

```
riri:rouge:1:caneton
fifi:vert:2:caneton
loulou:bleu:3:caneton
donald:jaune:4:canard
trouvetou:violet:5:canard
daisy:rose:6:canard
picsou:or:7:canard
```

Exercices - Partie 2

- Affichez à l'écran par `cat` le contenu du fichier.
- Quel est le nombre de mots, de lignes et de caractères de `ctest2.txt` ?
- Combien de lignes contient ce fichier ?
- Affichez à l'écran le deuxième champ de toutes les lignes.
- Quel est le nombre de canards ?
- Envoyez le fichier `test2.txt` avec uniquement le premier champs de toutes les lignes dans un fichier `noms.txt`.
- Triez le fichier `noms.txt` par ordre alphabétique.

- Coupez le fichier `test2.txt` en **deux fichiers** `test2aa` et `test2ab` (voir l'option nécessaire dans le manuel en ligne).
- Réunissez-les dans un fichier `disney.txt`.

4. Montage et démontage manuel d'un système de fichiers

Pour un serveur, le montage d'un système de fichiers ne se réalise pas de façon automatique. La base du procédé se situe dans le fichier `/etc/fstab` déjà vu lors du montage d'un disque dur supplémentaire (chapitre Administration des ressources).

Ubuntu suit la tendance générale qui consiste à abandonner le répertoire `/mnt` pour le répertoire `/media`. Afin de faciliter les multiplications de périphériques, le répertoire `/media/cdrom` est en fait un lien vers `/media/cdrom0`.

a. Principes de montage

Le répertoire qui rattache le système de fichiers externe au système général s'appelle un point de montage et **doit exister au préalable**. Le montage passe par la commande `mount`. Assez complexe, elle mérite un passage vers les pages de manuels :

```
mount [options] [-t type] périphérique répertoire
```

La commande seule affiche les systèmes de fichiers et les attributs de montage :

```
root@serveur:~# mount
/dev/sda3 on / type ext3 (rw,relatime,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
/sys on /sys type sysfs (rw,noexec,nosuid,nodev)
varrun on /var/run type tmpfs (rw,noexec,nosuid,nodev,mode=0755)
varlock on /var/lock type tmpfs (rw,noexec,nosuid,nodev,mode=1777)
udev on /dev type tmpfs (rw,mode=0755)
devshm on /dev/shm type tmpfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda2 on /home type ext3 (rw,relatime,usrquota)
securityfs on /sys/kernel/security type securityfs (rw)
root@serveur:~# _
```

On voit bien les répertoires physiques sur les partitions et les montages systèmes comme par exemple `/dev/shm` espace mémoire partagé pour les échanges entre les processus.

Montage du lecteur de disquette

Ce type de périphérique, bientôt obsolète, se monte avec la commande (le type est détecté) :

```
mount /dev/fd0 /media/floppy0
```

Si la ligne du lecteur de disquettes se trouve présente dans le fichier `/etc/fstab`, un simple `mount /media/floppy0` suffit. Ne jamais enlever la disquette avant de la démonter sous peine de perte de données (le lecteur de CD-Rom, lui, ne peut pas s'ouvrir).

Montage du lecteur de CD-Rom

La commande complète nécessite le type :

```
mount -t iso9660 /dev/cd0 /media/cdrom0
```

Le montage d'autres partitions, notamment celles d'un système différent ne pose plus de problème : Ubuntu reconnaît la plupart des systèmes de fichiers avec maintenant le système NTFS en lecture **et écriture**.

b. Démontage

Le contraire de la commande `mount` est `umount` (et non pas `unmount`) :

```
umount point_de_montage
```

Attention : pour démonter un périphérique, vous ne devez être dedans.

5. Stratégies et outils de sauvegarde

La sauvegarde des données fait partie intégrante du travail de l'administrateur système. La question n'est pas de savoir pourquoi sauvegarder mais quand, comment et quoi sauvegarder. Corollaire de la pratique des sauvegardes : la manipulation d'archives.

a. Principes de la sauvegarde de données

Il faut distinguer les différentes méthodes : la méthode dite incrémentielle s'applique à sauvegarder les fichiers modifiés (ou susceptible de l'avoir été) depuis la dernière sauvegarde. Cette pratique suffit dans de nombreux cas, mais ce n'est pas la seule : on peut envisager une sauvegarde complète à intervalles réguliers, jour, semaine, mois...

En fait, pour une meilleure efficacité on utilise trois méthodes en même temps :

- la sauvegarde **complète**, qui comme son nom l'indique sauvegarde toutes les données à un instant T ;
- la sauvegarde **incrémentielle**, qui ne sauvegarde que les données modifiées depuis la dernière sauvegarde complète ;
- la sauvegarde **différentielle**, qui ne sauvegarde que les données modifiées depuis la dernière sauvegarde (qu'importe son type).

Les unités de temps généralement utilisées sont le jour, la semaine et le mois. Voici un exemple de cycle de sauvegarde :

	Dim	**Lun**	**Mar**	**Mer**	**Jeu**	**Ven**	**Sam**
Semaine 1	C	D	D	I	D	D	D
Semaine 2	C	D	D	I	D	D	D
Semaine 3	C	D	D	I	D	D	D
Semaine 4	C	D	D	I	D	D	D

Et en utilisant, bien entendu, des supports de sauvegarde différents.

b. Commandes et outils de sauvegarde

Note : les exemples portent sur la sauvegarde du répertoire des utilisateurs : /home.

La commande `tar` (`tape archiver`)

Cet utilitaire lit les fichiers et les enregistre de façon séquentielle dans le fichier de sortie. Il utilise pour compresser en ligne les données un utilitaire externe : `gzip`. Citons quelques problèmes épineux : la restauration des données (les archives corrompues ne sont pas récupérables), la sauvegarde de système entier (pour `/proc` ou les répertoires `nfs`) et l'impossibilité de sauvegarde sur plusieurs bandes.

Syntaxe (simplifiée, vu la complexité de la commande) :

```
tar [-options] archive répertoire/fichier
```

Exemple d'utilisation classique :

```
tar -cvzpf home.tar.gz /home
```

Où :

`c (--create)` : indique le mode création.

`v (--verbose)` : indique le mode "verbeux". Dans l'exemple, la liste complète des fichiers.

`z (--gzip)` : indique l'utilisation de `gzip`.

`p (--same-permissions)` : garde les mêmes permissions et droits.

`f (--file)` : utilise le nom du fichier archive qui suit.

La restauration se réalise par :

```
tar -zxvf home.tar.gz
```

Où :

`z (--gunzip)` : indique l'utilisation de `gunzip`.

`x (--extract)` : indique l'extraction des fichiers de l'archive.

Les deux dernières options sont déjà vues, l'absence de répertoire de sortie indique le répertoire courant. Pour une sauvegarde incrémentielle ou différentielle, on utilise l'option :

```
-g fichier (--listed-incremental)
```

Ce qui donnerait :

```
tar -cvzp -g /root/incrs.lst -f home.tar.gz /home
```

Où `incrs.lst` est le fichier stockant les informations de la sauvegarde incrémentielle. Si on ne change pas le fichier `incrs.lst`, `tar` effectuera une sauvegarde différentielle la fois suivante. Si l'on se base par rapport au premier `incrs.lst` (copié pour sauvegarde), on effectue une sauvegarde incrémentielle. Utiliser un autre nom (ou effacement) pour le fichier de sauvegarde incrémentielle revient à faire une sauvegarde complète.

La commande `dd (device to device)`

Cette commande copie tout simplement son fichier spécifié en entrée sur celui en sortie. Créé initialement pour le support bande magnétique, on l'utilise maintenant soit pour dupliquer une bande (entre deux lecteurs), soit pour écrire une image amorçable sur une disquette. On peut aussi l'utiliser pour effectuer des conversions de types de bandes, mais cela ne sera pas vu ici.

Syntaxe :

```
dd [options] if=fichier_entrée of=fichier_sortie [bs=taille_de_bloc]
```

Les commandes `dump` et `restore`

Ces commandes s'utilisent pour les sauvegardes incrémentielles du système Linux. Attention, `dump` opère sur un niveau plus bas que les autres commandes de sauvegarde : celui des inodes ; il prend en charge explicitement le système de fichiers utilisé.

Syntaxe principale :

```
dump [n]uf periphérique_sauvegarde système_à_sauvegarder
```

Où :

`n` : représente le niveau incrémentiel de 0 à 9, le niveau 0 définissant une sauvegarde complète alors qu'un niveau n définit une sauvegarde différentielle par rapport au niveau n-1.

`u` : stocke l'historique des sauvegardes dans `/etc/dumpdates`.

`f` : ce paramètre signifie que le nom de fichier (au sens Linux, c'est-à-dire un fichier de périphérique) qui suit est celui de la sauvegarde.

Exemples d'utilisation :

```
dump -0uf /dev/st0 /home
 # Sauvegarde complète dans un lecteur de bandes de /home
dump -0f /opt/sauv /home
 # Sauvegarde du répertoire /home dans un fichier sauv (pas de u)
```

Un `dump` supplémentaire sur un même niveau donnera une sauvegarde incrémentielle. La sauvegarde à distance s'effectue en rajoutant simplement le nom du serveur (et en utilisant `ssh`) devant le périphérique comme :

```
dump 0uf serveur:/dev/st0 /home
```

La commande `restore` extrait la sauvegarde effectuée avec `dump`.

Syntaxe :

```
restore -t -f /dev/st0
```

Cette commande affiche la liste des fichiers (l'option `t`) et restaure de façon non interactive l'ensemble des fichiers présents sur une bande. Il existe la possibilité d'une restauration interactive (option `i`) :

```
restore -i -f /dev/st0
```

Une invite apparaît : `restore>` ; la commande `ls` permet de voir l'ensemble des fichiers. Si l'on restaure uniquement le répertoire de l'utilisateur `donald`, on tape :

```
restore> add donald
restore> extract
```

Note : voir l'option `-s` si une bande renferme plusieurs archives.