
Eléments de base

Pascal Picard

Corto E.T.F., K&M

Sainte-Clotilde, Ile de la Réunion

Ivan Kurzweg

Fremens Institute.

La Possession, Ile de la Réunion

\$Id: C-ElemBase.xml,v 1.4 2006/02/12 09:43:02 pascal Exp \$

Copyright © 2003-2005, 2006 Pascal PICARD,*pascal@seth.homeunix.net*, 2006, Ivan KURZWEG *ik-r@wanadoo.fr*, 2006

Permission to use, copy, modify, and distribute this documentation for any purpose with or without fee is here by granted, provided that the above copyright notice and this permission notice appear in all copies.

THE DOCUMENTATION IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS DOCUMENTATION INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS DOCUMENTATION.

1. Commandes et notions de bases
 - 1.1. chemins, chemins absolus et chemins relatifs
 - 1.2. **pwd**
 - 1.3. **cd**
 - 1.4. Gérer les répertoires : **mkdir**, **rmdir**
 - 1.5. .. et .
 - 1.6. **ls (1)**
 - 1.7. Notion d'inode
 - 1.8. Liens durs (**ln (1)**)
 - 1.9. Liens symboliques (**ln -s**)
 - 1.10. Manipuler les fichiers : **file**, **cp**, **mv**, **rm**
 - 1.11. **who**, **logname**, **whoami**, **hostname**
 - 1.12. **id**, **tty**, **printenv**
 - 1.13. **date**, **cal**
 - 1.14. Exercices
2. Utilitaires de *textprocessing*
 - 2.1. **wc**, **tail**, **head**, **cut**, **nl**, **pr**
 - 2.1.1. **wc (1)**
 - 2.1.2. **tail (1)**
 - 2.1.3. **head (1)**
 - 2.1.4. **cut (1)**
 - 2.1.5. **nl (1)**
 - 2.1.6. **pr (1)**
 - 2.2. Utilitaires **cat**, **tac**, **more**, **less**, **sort**, **uniq**
 - 2.2.1. **cat (1)**
 - 2.2.2. **tac**
 - 2.2.3. **more (1)**
 - 2.2.4. **less (1)**
 - 2.2.5. **sort (1)**
 - 2.2.6. **uniq (1)**
 - 2.3. **expand**, **unexpand**, **tr**
 - 2.3.1. **expand (1)**
 - 2.3.2. **unexpand (1)**
 - 2.3.3. **tr (1)**
 - 2.4. **od**, **split**, **fmt**, **fold**
 - 2.4.1. **od (1)**
 - 2.4.2. **split (1)**
 - 2.4.3. **fmt (1)**
 - 2.4.4. **fold (1)**
 - 2.5. **paste**, **join**, **tee**
 - 2.5.1. **paste (1)**
 - 2.5.2. **join (1)**
 - 2.5.3. **tee (1)**
 - 2.6. Exercices
3. Eléments d'initiation à **sed**
 - 3.1. Principes généraux
 - 3.2. Commandes Sed
 - 3.3. Quelques exemples
 - 3.4. Exercices
4. Eléments d'initiation à **awk**
 - 4.1. Premier exemple

Résumé

Nous aborderons dans une première partie, quelques commandes de bases de l'univers *Unix*. Puis nous mènerons dans la deuxième partie une étude, non exhaustive, des principaux utilitaires de gestion de texte.

Nous proposons dans les deux parties suivantes quelques éléments d'initiation à **awk** et **sed**.

Enfin, pour nous joindre : <pascal@seth.homeunix.net><ik-r@wanadoo.fr>

1. Commandes et notions de bases

Cette partie présente de manière très générale quelques commandes des systèmes *Unix* sans rentrer dans le détail de toutes les options. La plupart seront revues de manière plus approfondies dans des chapitres ultérieurs et plus spécifiques. La consultation du *man* est impérative (rappel : `man <commande>`).

Note

Dans les extraits qui illustrent le comportement des commandes, j'utilise parfois le caractère #. Celui-ci est considéré comme un début de commentaire qui s'étend jusqu'à la fin de la ligne, le système ignore tout ce qui fait suite à ce caractère.

Dans tous les exemples, le symbole \$ désigne un utilisateur standard, sans privilège particulier.

Enfin, dans les paragraphes qui suivent les termes référence et désignation sont des synonymes, ils sont donc interchangeables.

1.1. chemins, chemins absolus et chemins relatifs

Le système de fichiers dans l'environnement *Unix* est organisé sous forme arborescente^[1]. Les noeuds internes de l'arbre sont généralement des répertoires (i.e. des fichiers au sens *Unix* contenant eux mêmes des fichiers simples ou des répertoires), tandis que les feuilles sont les fichiers. La désignation qui permet de nommer une ressource est un chemin.

La racine de cette arborescence est dénotée par le symbole /, dit *root*. Tout chemin qui commence par ce symbole est dit absolu [*absolute pathname*] puisqu'il se réfère à l'origine de la structure arborescente. Ainsi, les désignations suivantes sont des chemins absolus :

```
/sbin,  
  
/usr/local/bin,  
  
/usr/X11R6/bin,
```

Un chemin absolu n'est autre qu'un chemin relatif à la racine (/) de l'arborescence. Si maintenant on veut désigner une ressource relativement à l'endroit où l'on se trouve, on obtient un chemin relatif. Si, par exemple, je me trouve dans le répertoire /usr, alors les désignations suivantes sont des chemins relatifs [*relative pathnames*] :

```
local/bin  
X11R6/bin
```

1.2. pwd

L'utilitaire **pwd (1)** [*print working directory*, situé dans /bin, section (1) du man], indique le répertoire courant dans lequel on se trouve.

```
EXEMPLE :  
  
$ pwd  
/home/pascal/lecture
```

1.3. cd

La commande interne **cd** [*change directory*], permet de naviguer dans l'arborescence. Son argument optionnel qui est un chemin relatif ou absolu permet de changer de répertoire courant.

```
EXEMPLES :  
  
$ pwd  
/home/pascal/lecture  
  
$ cd /usr  
$ pwd  
/usr  
  
$ cd local/bin  
$ pwd  
/usr/local/bin
```

Sans argument, la commande `cd` nous ramène dans notre répertoire de domiciliation (ce qui est équivalent à la séquence `cd ~`, c.f. shell).

```
EXEMPLE :  
  
$ pwd  
/usr/local/bin  
  
$ cd  
$ pwd  
/home/pascal
```

1.4. Gérer les répertoires : `mkdir`, `rmdir`

mkdir. Cette commande permet de créer de nouveau(x) répertoire(s)

```
EXEMPLE :  
  
$ ls -F  
bin/ mytmp@ test/ tmp/  
$ mkdir rep1 rep2 rep3 rep4  
$ ls -F  
bin/ mytmp@ rep1/ rep2/ rep3/ rep4/ test/ tmp/
```

L'option `-p` de la commande `mkdir` permet de créer une hiérarchie en une seule opération, dans l'exemple ci-après, on crée un répertoire (`rep5`) contenant un répertoire (`facile`) contenant ... :

```
EXEMPLE :  
  
$ mkdir -p rep5/facile/a/faire  
$ ls -R rep5  
ls -FR rep5  
rep5:  
facile/  
  
rep5/facile:  
a/  
  
rep5/facile/a:  
faire/  
  
rep5/facile/a/faire:
```

rmdir. Cette commande permet de supprimer un répertoire vide. Supprimer un répertoire revient explicitement à supprimer les objets qu'il contient. C'est logique, mais inefficace si la hiérarchie à supprimer contient beaucoup d'objets, c.f. la section consacrée à la commande `rm` pour une méthode radicalement efficace.

Illustration - Répertoires

- i. Créer dans votre répertoire de domiciliation le répertoire `~/elemntBase/`
- ii. Créer dans votre répertoire de domiciliation le répertoire `~/elemntBase/part1/` en utilisant l'option `-p`
- iii. À quelle condition est-ce que la ligne de commande `mkdir ~/elementBase/part1/exo1/com` peut fonctionner et créer effectivement ce sous-répertoire ?
- iv. Vous voulez créer à la fois un répertoire `~/elemntBase/part1/exo1/dir1/` et un sous répertoire `~/elemntBase/part1/exo1/dir2`. Quelle option faut-il utiliser pour que : 1. Si `~/elemntBase/part1/exo1` n'existe pas, le répertoire et son sous-répertoire soient créés; 2. Si `~/elemntBase/part1/exo1` existe déjà, le sous-répertoire `dir1/` soit créé, sans message d'erreur.

1.5. .. et .

La désignation « .. » désigne le répertoire père du répertoire courant.

```
EXEMPLES :  
  
$ pwd  
/usr/local/bin  
  
$ cd ..  
$ pwd  
/usr/local  
  
$ cd ../sbin  
$ pwd  
/usr/sbin
```

Important

Sous *Unix*, une commande est séparée de ces arguments par les caractères d'espace. Ainsi `cd .` n'a pas de sens ! Il faut écrire `cd .`, noter bien l'espace séparateur.

La désignation « `.` » désigne le répertoire courant, c'est donc une auto-référence. Ainsi l'opération suivante, ne modifie nullement le répertoire courant :

```
EXEMPLE :  
  
$ pwd  
/usr/sbin  
$ cd .  
$ pwd  
/usr/sbin
```

Cette désignation est souvent utilisée pour exécuter un programme situé dans le répertoire courant, i.e. hors des localisations traditionnelles.

```
EXEMPLE :  
  
$ pwd  
/usr/sbin  
$ cd  
$ pwd  
/home/pascal  
$ cd mybin  
$ pwd  
/home/pascal/mybin  
$ ./myprog  
...  
exécution de myprog  
...
```

1.6. ls (1)

Cette commande (situé traditionnellement dans `/bin`), permet de lister le contenu d'un répertoire. Elle contient nombre d'options permettant d'avoir beaucoup de détails.

```
EXEMPLE :  
  
$ cd /usr/local  
$ ls  
Acrobat5          bin          lib          ports        work  
GNUstep           distfiles   libdata      proj         www  
OpenOffice.org1.0 etc         libexec      sbin  
OpenOffice.org1.0.3 include     linux-sun-jdk1.3.1 share  
ant              info        man          src  
apps            jdk1.3.1   pgsql       sup
```

L'option `-a` permet de lister en plus les *dotfiles*, i.e. les fichiers cachés qui commencent toujours par un point (dans leur nom), ici nous obtenons en plus les entrées spéciales « `.` » et « `..` » :

```
EXEMPLE :  
  
$ pwd  
/usr/local  
$ ls -a  
.          ant          info         man          src  
..         apps        jdk1.3.1    pgsql       sup  
Acrobat5   bin          lib          ports        work  
GNUstep    distfiles   libdata     proj         www  
OpenOffice.org1.0 etc         libexec     sbin  
OpenOffice.org1.0.3 include     linux-sun-jdk1.3.1 share
```

L'option `-l` permet de lister au format long le répertoire ou le fichier qui lui est éventuellement passé en argument.

```
EXEMPLES :  
  
$ ls -l /usr  
total 41  
drwxr-xr-x 13 root  wheel   512 May 25 13:21 X11R6  
drwxr-xr-x  2 root  wheel  7168 May 24 10:19 bin  
drwxr-xr-x  3 root  wheel   512 Jan  8 2003 compat  
drwxr-xr-x  3 root  wheel   512 Jan  6 2003 games  
drwxr-xr-x 41 root  wheel  3584 May 24 10:18 include  
drwxr-xr-x  4 root  wheel  6144 May 24 10:19 lib  
drwxr-xr-x  9 root  wheel   512 Oct  9 2002 libdata  
drwxr-xr-x  8 root  wheel  1536 May 24 10:19 libexec
```

```

drwxr-xr-x 28 root wheel 512 Jul 10 08:23 local
drwxrwxrwt 44 root wheel 9216 May 24 14:48 lost+found
drwxr-xr-x 3 root wheel 512 May 24 09:23 obj
drwxr-xr-x 2 root wheel 4096 May 24 10:19 sbin
drwxr-xr-x 27 root wheel 512 May 24 10:17 share
drwxr-xr-x 21 root wheel 512 May 12 22:02 src
drwxr-xr-x 3 root wheel 512 Jan 8 2003 sup

$ ls -l /etc/passwd
-rw-r--r-- 1 root wheel 1527 Aug 7 01:50 /etc/passwd

```

Sans trop rentrer dans les détails, notons que les listings précédents, sont structurés en sept colonnes. De la gauche vers la droite, on trouve en première colonne les permissions sur l'objet, en seconde colonne le nombre de liens sur l'objet, puis en troisième et quatrième le propriétaire et le groupe de l'objet, en cinquième la taille en octet, en sixième la date de dernière modification (*mtime*) de l'objet. Enfin, la dernière colonne donne le nom de l'objet.

Nous examinerons plus tard la première colonne en détail, celle qui présente en détail les permissions sur l'objet, mais nous pouvons déjà décomposer ces informations en quatre groupes – *rw* *rx* *rx*. Le premier sous-groupe décrit le type de fichier : un fichier ordinaire est représenté par un « - », et les répertoires par « d ». La liste suivante présente les différents types de fichiers exprimés dans le premier sous-groupe :

1. « - » : Fichier ordinaire
2. « b » : Fichier spécial en mode bloc
3. « c » : Fichier spécial en mode caractère
4. « d » : Répertoire
5. « l » : lien symbolique
6. « p » : tube nommé (pipe)

Il est possible de lister le/les répertoires sans leur(s) contenu(s) (option `-d`). Il est évidemment possible de combiner les options :

```

EXEMPLE :

$ ls -ld /usr /usr/bin /usr/sbin
drwxr-xr-x 17 root wheel 512 Jan 8 2003 /usr
drwxr-xr-x 2 root wheel 7168 May 24 10:19 /usr/bin
drwxr-xr-x 2 root wheel 4096 May 24 10:19 /usr/sbin

```

De manière symétrique, on peut vouloir voir l'ensemble des informations d'un répertoire donné, i.e. l'ensemble des informations sur les répertoires et sous répertoires qu'il contient (parcours récursif) avec l'option (`-R`). Attention cela peut être volumineux !

Essayez-le dans votre `HOME`, puis dans `/home`.

Illustration - ls

- i. Listez le répertoire au format long : `~/elemntBase/part1/` et interprétez le résultat
- ii. Créez le fichier `~/elemntBase/part1/fic1` contenant une phrase de votre choix.
- iii. Lister de nouveau le répertoire - Interprétez le résultat. Que représente `total` ?
- iv. Créez un deuxième fichier, puis un sous répertoire et notez les changements.

1.7. Notion d'inode

Tout est fichier sous *Unix* (« credo *Unixien* », un répertoire est donc un fichier) et chaque fichier se voit attribuer un index unique appelé *inode* [*index node*]. Informellement, un *inode* est une structure de données qui décrit le contenu d'un fichier à l'exception notable de son nom (sa désignation). Un fichier peut ainsi avoir plusieurs désignations différentes, lesquelles sont stockées dans les répertoires^[2]. La liste des *inodes* d'un répertoire donné est obtenue avec la commande `ls -li <repertoire>`. On peut évidemment combiner les options :

```

EXEMPLE :

$ ls -ila /usr/X11R6/libexec
total 137
3984118 drwxr-xr-x 4 root wheel 512 Jun 22 19:11 .
3928896 drwxr-xr-x 13 root wheel 512 May 25 13:21 ..
3983621 -r-xr-xr-x 1 root wheel 8296 Jun 22 16:40 gconf-sanity-check-2
3983619 -r-xr-xr-x 1 root wheel 45752 Jun 22 16:40 gconfd-2
4005734 drwxr-xr-x 3 root wheel 512 May 25 15:48 gkrellm
3983624 -r-xr-xr-x 1 root wheel 5112 Jun 22 19:04 gnome2-db2html
3983625 -r-xr-xr-x 1 root wheel 21884 Jun 22 19:04 gnome2-info2html
3983623 -r-xr-xr-x 1 root wheel 43848 Jun 22 19:04 gnome2-man2html
3983626 -r-xr-xr-x 1 root wheel 8564 Jun 22 19:11 gnome_seg2
4017439 drwxr-xr-x 2 root wheel 512 May 29 15:36 sawfish

```

Observons que le répertoire `/usr/X11R6/libexec` possède l'*inode* 3984118 et qu'il est référencé 4 fois (i.e. bien qu'existant en un seul

endroit cet *inode* possède plusieurs références). Quelles sont-elles (c.f. ci-après) ?

EXEMPLE :

```
$ ls -idl /usr/X11R6/libexec /usr/X11R6/libexec/. /usr/X11R6/libexec/gkrellm/.. /usr/X11R6/libexec/sawfish/..
3984118 drwxr-xr-x  4 root  wheel  512 Jun 22 19:11 /usr/X11R6/libexec
3984118 drwxr-xr-x  4 root  wheel  512 Jun 22 19:11 /usr/X11R6/libexec/.
3984118 drwxr-xr-x  4 root  wheel  512 Jun 22 19:11 /usr/X11R6/libexec/gkrellm/..
3984118 drwxr-xr-x  4 root  wheel  512 Jun 22 19:11 /usr/X11R6/libexec/sawfish/..
```

Puisque « .. » est une référence au répertoire père du répertoire courant, /usr/X11R6/libexec/gkrellm/.. et /usr/X11R6/libexec/sawfish/.. sont bien deux désignations différentes du même objet (à savoir /usr/X11R6/libexec, qui est aussi désigné par /usr/X11R6/libexec/).

1.8. Liens durs (ln (1))

La désignation d'un *inode* est encore appelée : *lien* [*link*]. Un *inode* peut posséder plusieurs désignations qu'on appelle *liens durs* [*hard links*]. L'*inode* ainsi multi-référencé existe au sein du *filesystem* tant qu'il reste un lien dur le référençant. Pour créer un *lien dur*, on utilise la commande **ln** :

EXEMPLE :

```
$ touch toto
$ ls -i
 96770 toto
$ ln toto titi          # ln <src> <dst>
$ ls -i
 96770 titi    96770 toto
```

Limitations classiques des *liens durs* :

- On ne peut créer des liens durs que sur des fichiers (réguliers, caractère, bloc, socket, *named pipe*...) et non sur des (fichiers) répertoires^[3].
- Les liens ne traversent pas les limites d'un *filesystem*^[4].

Enfin, puisque chaque désignation « pointe » (ou référence) le même *inode* (objet) et que les droits d'accès sont stockés dans l'*inode*, les différentes désignations (liens durs) ont les mêmes droits d'accès^[5].

Illustration - liens durs

- Créez le répertoire ~/elemntBase/part1/links/.
- Créez le lien ~/elemntBase/part1/links/link1 référençant le fichier ~/elemntBase/part1/fic1 créé précédemment.
- Comparez les fichiers ~/elemntBase/part1/links/link1 et ~/elemntBase/part1/fic1 (*inode*, taille, résultat à l'affichage)
- Supprimer ~/elemntBase/part1/fic1 Que constatez vous ?

1.9. Liens symboliques (ln -s)

Les limitations sur les *liens durs* [*symbolic links*] ont conduit à l'élaboration d'un nouveau type de fichier dit *lien symbolique*. Un tel (type de) fichier se réfère à un autre par le nom plutôt que par l'*inode*, en fait le contenu de ce fichier est un chemin absolu ou relatif qui conduit au fichier référencé. Pour créer un *lien symbolique*, on utilise la commande **ln -s** :

EXEMPLE :

```
$ ln -s ~/tmp mytmp          # création d'un lien symb. sur le répertoire ~/tmp
$ ll                          # utilisation de l'alias
total 8
lrwxrwxrwx  1 pascal  gnu          16 Aug 14 13:14 mytmp -> /home/pascal/tmp
drwx-----  2 pascal  gnu         4096 Aug 14 09:45 test
drwx-----  2 pascal  gnu         4096 Aug 14 13:14 tmp
$ cd mytmp ; pwd              # vérification
/home/pascal/mytmp

$ ll -i . ~/tmp              # contenu ~/mytmp qui est le contenu de ~/tmp
.:
total 1
 96770 -rw-----  1 pascal  gnu          24 Aug 14 13:19 toto
 96771 lrwxrwxrwx  1 pascal  gnu           4 Aug 14 11:14 tutu -> toto

/home/pascal/tmp:
total 1
 96770 -rw-----  1 pascal  gnu          24 Aug 14 13:19 toto
 96771 lrwxrwxrwx  1 pascal  gnu           4 Aug 14 11:14 tutu -> toto
```

Remarques :

- un lien symbolique est dénoté par un **l** dans le listage au format long (**ls -l** ou son alias **ll**),
- la taille d'un lien symbolique (exprimée en octet) est la taille nécessaire au stockage du chemin d'accès (relatif ou absolu) à l'objet référencé (dans l'exemple précédent 4 octets pour la désignation relative « toto »),
- un lien symbolique peut référencer un fichier régulier, un répertoire, un fichier en mode bloc ou caractère...
- puisqu'un lien symbolique s'appuie sur la notion de chemin (et non d'*inode*) il est possible d'en créer sur un objet d'un *filesystem* différent.
- Les droits d'accès au lien symbolique sont, logiquement, les droits d'accès à l'objet référencé, ainsi sous *GNU/Linux* le « lrwxrwxrwx » qui apparaît lors d'un listage au format long n'a pas de signification, tandis que sous *xBSD* les droits qui s'affichent sont hérités de la valeur de l'*umask* de l'utilisateur (uid) qui crée le lien, cette valeur n'est pas mise à jour si les droits de l'objet référencé sont modifiés !

Illustration - liens symboliques

- Créez le lien symbolique `~/elemntBase/part1/links/link2` référençant le fichier `~/elemntBase/part1/fic2` créé précédemment.
- Comparez les fichiers `~/elemntBase/part1/links/link2` et `~/elemntBase/part1/fic2` . (inode, taille, résultat à l'affichage)
- Supprimer `~/elemntBase/part1/fic2` Que constatez vous ?

1.10. Manipuler les fichiers : file, cp, mv, rm

file (1). Cette commande tente de déterminer le type du fichier qui lui est passé en argument.

EXEMPLES :

```
$ file ~/tmp/toto
/home/pascal/tmp/toto: ASCII text

$ file ~/test/C/bin/_crypt
_crypt: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux
2.0.0, dynamically linked (uses shared libs), not stripped

$ file ~/test/C/src/_crypt.c
/home/pascal/test/C/src/_crypt.c: ISO-8859 C program text

$ file ~/bin/_backup
/home/pascal/bin/_backup: C shell script text executable

$ file /bin/rm
/bin/rm: ELF 32-bit LSB executable, Intel 80386, version 1 (FreeBSD), for FreeBS
D 4.8, statically linked, stripped

$ file /usr/local/bin/xemacs
/usr/local/bin/xemacs: symbolic link to xemacs-21.1.14
```

cp (1). Cette commande permet de copier un fichier existant.

EXEMPLE :

```
$ pwd                                # où suis-je ?
/home/pascal/tmp
$ ls -l                               # quels sont les "objets" ?
 96771 bookmarks.html                # 1 "objet"
$ cp bookmarks.html bookmarks.html.SAV # en faire une copie de sauvegarde
$ ls -l                               # "objets" physiquement différents ?
 96771 bookmarks.html      96770 bookmarks.html.SAV
```

Note

- La commande **cp** crée bien un objet différent puisque les deux fichiers ont des *inodes* différents.
- Noter le format : **cp <src> <dst>**, si **<dst>** existe il est écrasé par défaut.

mv (1). Cette commande permet de déplacer/renommer un fichier existant dans l'arborescence. Un déplacement ne change pas l'*inode* tant que l'on reste dans le même *filesystem*, mais sa localisation dans l'arborescence i.e. sa désignation (sa référence ou encore son nom).

EXEMPLES :

```
$ pwd                                # où suis-je ?
/home/pascal/tmp
$ ls -l                               # inodes des "objets"
 96771 bookmarks.html      96770 bookmarks.html.SAV
```

```

$ mv bookmarks.html ~/test # déplacement de bookmarks.html dans le rép. ~/test
$ ls -i ~/test # l'inode de bookmarks.html est toujours 96771
.:
 96770 bookmarks.html.SAV

/home/pascal/test:
 96771 bookmarks.html

$ mv ~/test/bookmarks.html /tmp # changement de filesystem
$ ls -i /tmp/bookmarks.html
 14 /tmp/bookmarks.html

```

Rappel : /tmp réside sur une partition séparée, or l'unicité des numéros d'*inodes* est assuré uniquement dans un *filesystem* donné.

rm. Cette commande permet d'enlever les objets d'un *filesystem*.

```

EXEMPLE :

$ pwd
/home/pascal
$ ls -F
bin/ file.txt file.txt~ test/ titi titi.o titi~ tmp/ tootoo toto toto.
o toto~
$ rm titi
/bin/rm: remove regular file `titi'? y
$ ls -F # plus de "titi"
bin/ file.txt file.txt~ test/ titi.o titi~ tmp/ tootoo toto toto.o tot
o~
$ rm * # effacer tous les fichiers !
/bin/rm: remove directory `bin'? y
/bin/rm: cannot remove directory `bin': Is a directory
/bin/rm: remove regular file `file.txt'? y
/bin/rm: remove regular file `file.txt~'? y
/bin/rm: remove directory `test'? y
/bin/rm: cannot remove directory `test': Is a directory
/bin/rm: remove regular empty file `titi.o'? y
/bin/rm: remove regular file `titi~'? y
/bin/rm: remove directory `tmp'? ^C
$

```

Note

- Dans l'exemple, il y a demande de confirmation avant d'enlever l'objet. Cela est dû au fait que nous avons défini un alias sur **rm**, qui active l'option **-i** (pour interactif). Le comportement par défaut de **rm** est le comportement classique des commandes *Unix* : être silencieux quand il n'y a rien de spécial à signaler^[6].
- Le métacaractère « * » désigne tous les objets (i.e. tous les fichiers).
- Sans argument(s) spécifique(s) la commande **rm** tente d'effacer les fichiers qui ne sont pas des répertoires.

Effacer un répertoire : un répertoire n'est autre qu'un fichier contenant une liste de couple <désignation, *inode*>. Effacer un répertoire revient à effacer son contenu (récursivement) puis le répertoire lui-même. Pour effectuer une telle action on utilise l'option **-r** ou l'option synonyme **-R** de la commande **rm**. Compte tenu de l'alias existant, la séquence **rm -r ~/tmp** (dont la sémantique est : « effacer l'arborescence désignée par ~/tmp ») va demander confirmation de l'effacement de chacun des objets. Si on est sûr de ce que l'on fait, on utilisera plutôt la commande suivante **rm -rf ~/tmp** (dont la sémantique est : « effacer l'arborescence désignée par ~/tmp, sans me poser de questions »), *attention c'est radical !*

```

EXEMPLE :

$ pwd # où suis-je ?
/home/pascal

$ ls -F # quel est le contenu ?
bin/ test/ tmp/ tootoo toto toto.o toto~

$ ls -lR ~/tmp # quel est le contenu de ~/tmp ?
total 20 # 2 fichiers réguliers
-rw----- 1 pascal gnu 20341 Aug 14 09:18 bookmarks.html.SAV
-rw----- 1 pascal gnu 0 Aug 14 09:56 toto

$ rm -rf ~/tmp # "supprimer" le répertoire ~/tmp, sans confirmation
$ ls -F # ~/tmp n'existe plus
bin/ test/ tootoo toto toto.o toto~

```

Illustration - Manipulation de fichiers

- Copiez ~/elemntBase/part1/links/link2 vers le fichier ~/elemntBase/part1/fic2 créé précédemment.
- Comparez les fichiers ~/elemntBase/part1/links/link2 et ~/elemntBase/part1/fic2 . (inode, taille, résultat à l'affichage)

- iii. Créez en une commande les fichiers `~/elemntBase/part1/fic3` et `~/elemntBase/part1/fic4` . Déplacez tous les fichiers commençant par `fic` vers le répertoire en une seule commande (utilisation des méta - caractères)

1.11. who, logname, whoami, hostname

who (1). Permet de connaître qui travaille actuellement sur cette machine.

```
EXEMPLE :  
  
$ who  
guest          ttyv1      Aug 13 11:39  
pascal        ttyv0      Aug 10 11:26 (:0.0)  
pascal        ttyv1      Aug 13 11:39 (:0.0)  
pascal        ttyv2      Aug 11 12:01 (:0.0)  
pascal        ttyv3      Aug 7 08:15 (:0.0)  
foo           ttyv4      Aug 13 11:39 (Euler)  
pascal        ttyv5      Aug 11 12:00 (:0.0)  
pascal        ttyv6      Aug 9 08:32 (:0.0)
```

Indique ici :

- que l'utilisateur *guest* est connecté sur le premier terminal non graphique (ttyv1).
- que l'utilisateur *pascal* est connecté sur 6 terminaux (en interface graphique, la notation :0.0 de la dernière colonne indiquant le *display* graphique).
- enfin, que l'utilisateur *foo* est connecté sur le terminal ttyv4, depuis la machine Euler (session distante ssh, par exemple).

logname (1) et whoami (1). Sans argument ces deux commandes donnent le nom de connexion de l'utilisateur courant.

```
EXEMPLE :  
  
$ whoami  
pascal  
$ logname  
pascal
```

hostname (1). Affiche le nom de la machine courante.

```
EXEMPLE :  
  
$ hostname -s # nom court : xBSD  
Godel  
  
$ hostname # GNU/Linux  
tux
```

1.12. id, tty, printenv

id (1). Affiche l'identifiant de l'utilisateur (uid), son identifiant de groupe primaire (gid), les identifiants des groupes secondaires, ainsi que les nom associés.

```
EXEMPLE :  
  
$ id  
uid=666(pascal) gid=666(gnu) groups=666(gnu), 0(wheel), 5(operator), 88(mysql),  
668(cvs), 70(pgsq1)
```

tty (1). Affiche le nom du terminal courant

```
EXEMPLE :  
  
$ tty  
/dev/ttyv7
```

printenv (1). Affiche toute ou partie de l'environnement courant.

```
EXEMPLE :  
  
$ printenv # FreeBSD  
SSH_AGENT=/usr/bin/ssh-agent  
USER=pascal  
MAIL=/var/mail/pascal  
HOME=/home/pascal  
SSH_ASKPASS=/usr/X11R6/bin/ssh-askpass  
LOGNAME=pascal  
BLOCKSIZE=K
```

```

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin
:/home/pascal/bin
DISPLAY=:0.0
SHELL=/bin/tcsh
FTP_PASSIVE_MODE=YES
SSH_AUTH_SOCK=/tmp/ssh-isrTtwab/agent.463
SSH_AGENT_PID=478
WMAKER_BIN_NAME=/usr/X11R6/bin/wmaker
WRASTER_COLOR_RESOLUTION=4
IFS=

ETERM_THEME_ROOT=/usr/X11R6/share/Eterm/themes/Eterm
ETERM_USER_ROOT=/home/pascal/.Eterm
COLORFGBG=default
WINDOWID=54526015
TERM=xterm-color
COLORTERM=rxvt
COLORTERM_BCE=Eterm
ETERM_VERSION=0.9.2
HOSTTYPE=FreeBSD
VENDOR=amd
OSTYPE=FreeBSD
MACHTYPE=i386
SHLVL=1
PWD=/home/pascal/bin
GROUP=gnu
HOST=Godel.corto.home
color=1
colorcat=1
LC_CTYPE=fr_FR.ISO8859-15
PAGER=less
MOZILLA_HOME=/usr/X11R6/bin/mozilla
EDITOR=emacs
CPUTYPE=k7
TEXEDIT=emacs
LSCOLORS=EhGxbxcxBxdxHBhbfhCad
CLICOLOR=1
CLICOLOR_FORCE=1
RSYNC_RSH=/usr/bin/ssh
CVSROOT=/home/cvsroot

$ printenv # GNU/Linux
LC_PAPER=a4
TERM=xterm-color
SHELL=/bin/bash
SSH_CLIENT=192.168.200.2 3878 22
SSH_TTY=/dev/pts/0
USER=pascal
LS_COLORS=no=00;fi=00;di=01;34;ln=01;36;pi=40;33;so=01;35;do=01;35;bd=40;33;01:c
d=40;33;01;or=40;31;01;ex=01;32;*.tar=01;31;*.tgz=01;31;*.arj=01;31;*.taz=01;31:
*.lzh=01;31;*.zip=01;31;*.z=01;31;*.Z=01;31;*.gz=01;31;*.bz2=01;31;*.deb=01;31:*
.rpm=01;31;*.jar=01;31;*.jpg=01;35;*.jpeg=01;35;*.gif=01;35;*.bmp=01;35;*.pbm=01
;35;*.pgm=01;35;*.ppm=01;35;*.tga=01;35;*.xbm=01;35;*.xpm=01;35;*.tif=01;35;*.ti
ff=01;35;*.png=01;35;*.mpg=01;35;*.mpeg=01;35;*.avi=01;35;*.fli=01;35;*.gl=01;35
;*.dl=01;35;*.xcf=01;35;*.xwd=01;35;*.ogg=01;35;*.mp3=01;35;*.wav=01;35:
PAGER=less
MAIL=/var/mail/pascal
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
PWD=/home/pascal
EDITOR=emacs
LANG=fr_FR@euro
SHLVL=1
HOME=/home/pascal
LOGNAME=pascal
LC_CTYPE=fr_FR.ISO8859-15
SSH_CONNECTION=192.168.200.2 3878 192.168.200.5 22
_=/usr/bin/printenv

```

1.13. date, cal

date (1). Evoquée sans argument cette commande donne la date du jour.

```

EXEMPLE :

$ date
Wed Aug 13 11:31:51 RET 2003

```

cal (1). Evoquée sans argument cette commande affiche le calendrier du mois en cours.

```

EXEMPLE :

$ cal
    August 2003
Su Mo Tu We Th Fr Sa

```

```

          1  2
    3  4  5  6  7  8  9
  10 11 12 13 14 15 16
  17 18 19 20 21 22 23
  24 25 26 27 28 29 30
  31

```

1.14. Exercices

Exercice - 1 - Liens durs (ou physiques)

- Créer un fichier texte `foo`, contenant le seul mot `foo`, dans le répertoire `~/elemntBase/part1/synth1/`.
- Créer `~/elemntBase/part1/synth1/dir1/foo2` comme lien dur à `~/foo`. Comment vérifiez que ces deux désignations différentes référencent le même objet.
- Est-il possible de savoir si un fichier possède plusieurs désignations ?
- Modifier le contenu de `~/elemntBase/part1/synth1/dir1/foo2` en lui ajoutant du texte (`echo ... >>~/elemntBase/part1/synth1/dir1/foo2`). Contenu prévisible de `~/elemntBase/part1/synth1/foo` ? Le vérifier.
- Supprimer la désignation `~/elemntBase/part1/synth1/dir1/foo2`. Que devient `~/elemntBase/part1/synth1/dir1/foo2` ?
- Peut-on créer un lien dur `~/elemntBase/part1/synth1/dir2` à `~/elemntBase/part1/synth1/dir1` ? Pourquoi ?
- Pourquoi les liens durs ne peuvent pas appartenir à des *filesystem* différents ?

Exercice - 2 - Liens symboliques

- Créer un fichier texte `~/elemntBase/part1/synth1/dir2/bar`, contenant le seul mot `bar`, puis un lien symbolique `~/elemntBase/part1/synth1/barls` vers celui-ci. Contrôler avec la commande `ls -liR`, puis avec `cat ~/elemntBase/part1/synth1/barls`. Renommer `~/elemntBase/part1/synth1/dir2/bar` en `~/elemntBase/part1/synth1/dir3/bar`. Contrôler le résultat et conclure.
- Tester la transitivité de la notion de lien symbolique (i.e. lien vers un lien vers une désignation d'un objet du *filesystem*).
- Tester l'acyclicité du SGF en créant une circularité de liens symboliques..

2. Utilitaires de *textprocessing*

Nous proposons dans cette partie un bref panorama des commandes de gestion des fichiers au format texte, format retenu par *Unix* car universel^[7]. Un détour par la documentation (`man`) permettra de compléter utilement ces connaissances.

2.1. `wc`, `tail`, `head`, `cut`, `nl`, `pr`

2.1.1. `wc` (1)

Cette commande [*word count*] permet de comptabiliser le nombre d'octets, de mots et de lignes du fichier qui lui est passé en argument. Dans l'exemple ci-dessous, je donne le nombre de lignes du fichier `/etc/group`.

```

EXEMPLE :

$ wc -l /etc/group      # nombre de ligne uniquement, ici
  45 /etc/group

```

Exercice - `wc`

- En utilisant la commande `ps -axcw -o ppid,pid,user,vsz,time,command.`, créez le fichier `~/elemntBase/part2/process` contenant des informations sur les processus en cours.
- En utilisant la commande `ps -aux.`, créez le fichier `~/elemntBase/part2/ps-aux` contenant des informations sur les processus en cours.
- Combien de lignes contiennent les fichiers ?

2.1.2. `tail` (1)

Cette commande affiche la partie finale (par défaut, les 10 dernières lignes) du fichier qui lui est passé en argument. Dans l'exemple qui suit, on utilise une propriété intéressante, l'option `-f` qui permet d'afficher en permanence la fin d'un fichier donné. On peut donc *logger* en temps réel ce qui arrive dans le fichier `/var/log/message` (si l'on possède les droits de lecture sur ce fichier). Pour en sortir faire un `<CTRL>-C`.

```

EXEMPLE :

$ tail -f /var/log/messages
Aug  5 16:01:44 Godel /kernel: Connection attempt to UDP 127.0.0.1:512 from 127.
Aug  5 16:17:10 Godel /kernel: Connection attempt to UDP 192.168.200.2:2428 from

```

```

Aug  5 16:17:10 Godel su: BAD SU pascal to toor on /dev/tty5
Aug  5 16:32:35 Godel su: pascal to toor on /dev/tty5
Aug  5 16:48:01 /kernel: pid 33004 (sysinstall), uid 666: exited on signal 11 (c
Aug  5 17:03:28 Godel /kernel: Connection attempt to UDP 192.168.200.2:2438 from
Aug  5 17:18:53 Godel /kernel: Connection attempt to UDP 192.168.200.2:2441 from
Aug  5 17:34:18 Godel /kernel: Connection attempt to UDP 192.168.200.2:2444 from
Aug  5 17:34:18 Godel /kernel: Connection attempt to UDP 192.168.200.2:2445 from
Aug  5 17:49:44 Godel /kernel: Connection attempt to UDP 192.168.200.2:2448 from
^C
$

```

Exercice - tail

- i. Créez le fichier `~/elemntBase/part2/procList` en supprimant la première ligne du fichier `~/elemntBase/part2/process`.

2.1.3. head (1)

Cette commande affiche la partie initiale (par défaut, les 10 premières lignes) du fichier qui lui est passé en argument.

2.1.4. cut (1)

Cette commande permet d'extraire des champs (délimités par des caractères), des lignes du fichier ou du flux qui lui est passé en argument. Dans l'exemple qui suit, j'extrais les champs 1 (login), 3 (uid) et 4 (gid) délimités par les caractères « : » du fichier `/etc/passwd` :

```

EXEMPLE :

$ cut -f1,3,4 -d: /etc/passwd
root:0:0
toor:0:11
daemon:1:1
bin:2:2
sys:3:3
sync:4:65534
man:6:12
mail:8:8
news:9:9
uucp:10:10
proxy:13:13
postgres:31:32
backup:34:34
operator:37:37
nobody:65534:65534
pascal:666:666
corto:667:666
sshd:101:65534

```

2.1.5. nl (1)

Cette commande ajoute un numéro de ligne à chaque ligne du fichier qui lui est passé en argument.

```

EXEMPLE :

$ nl ~/bin/mylock.sh
 1  #!/bin/sh

 2  stty -echo
 3  trap '' 1 2 3 15 # ignorer les signaux
 4  echo -n "Key:"
 5  read key
 6  echo
 7  echo -n "Again:"
 8  read again
 9  echo
10  if [ "$key" = "$again" ]; then
11      while [ 1 ]; do
12          read again
13          if [ "$key" != "$again" ]; then
14              echo "What a fair foot !"
15          else
16              echo "unlock terminal"
17              break
18          fi
19      done
20  else
21      echo "Mismatch keys"
22  fi
23  # restaurer
24  stty echo
25  trap 1 2 3 15

```

2.1.6. pr (1)

Cette commande permet de morceler un fichier en plusieurs pages, souvent dans le but de l'imprimer. L'exemple montre quelques lignes de la deuxième page d'un script Perl :

```
EXEMPLE :

$ pr ~/bin/logfile-1.2.pl | less
...
Jan  7 14:23 2003 /home/pascal/bin/logfile-1.2.pl Page 2

open(TEMP, "temp")
  || die "problème pour ouvrir temp : $!\n";
@T = split(/\s+/, <TEMP>);
($owner, $group) = @T[2,3];
close(TEMP) || warn " problème de fermeture du fichier temp : $!";

$owner = "$owner" . ":";
$group = "$group" . ":";

system("cat /etc/passwd | grep $owner > temp");
open(TEMP, "temp")
...
```

2.2. Utilitaires cat, tac, more, less, sort, uniq

2.2.1. cat (1)

Cette commande permet d'imprimer tel quel, sur la sortie standard, les contenus des fichiers qui lui sont passés en argument, i.e. elle lit le contenu de ces fichiers. Elle permet aussi de réaliser la concaténation de plusieurs fichiers dans un seul au moyen d'une redirection de la sortie standard.

```
EXEMPLE :

$ cat file1.txt file2.txt file3.txt > file123.txt
```

Exercice - cat

- i. Utiliser **cat** pour écrire deux lignes de texte et les mettre dans le fichier `~/elemntBase/part2/exo1/notes`.
- ii. Depuis le fichier `~/elemntBase/part2/exo1/notes`, écrire la commande permettant d'obtenir le fichier `~/elemntBase/part2/exo1/notesNum`, dont les lignes sont numérotées.

2.2.2. tac

Cette commande *GNU/Linux* fonctionne comme la **cat**, mais imprime dans l'ordre inverse, i.e. commence par la dernière ligne.

```
EXEMPLES :

$ cat file.txt
ligne 1
une autre ligne, la 2 !
puis la ligne 3

$ tac file.txt
puis la ligne 3
une autre ligne, la 2 !
ligne 1
```

2.2.3. more (1)

Cette commande permet l'affichage du contenu du fichier qui lui est passé en argument, en tenant compte des caractéristiques de l'écran. L'affichage se fait donc page par page, mais dans un mode unidirectionnel (du début vers la fin du fichier). Il est possible de faire des recherches sur critères en avant ou en arrière, d'avancer dans le fichier ligne par ligne ou page par page, c.f. commande **help** de **more**.

En pratique, on préfère utiliser la commande **less** (c.f. section suivante), qui propose d'avantages de fonctionnalités.

2.2.4. less (1)

Cette commande généralise la commande précédente en reprenant ses caractéristiques d'affichage et y ajoutant la possibilité d'aller en arrière i.e. de remonter dans le fichier.

Exercice - Less

- i. Comment dire à **less** d'ouvrir le fichier `~/elemntBase/part2/procList` en plaçant en haut de l'écran la ligne 15 ?

- ii. Comment dire à **less** d'ouvrir le fichier `~/elemntBase/part2/procList` en plaçant en haut de l'écran la première ligne où apparaît le motif "tty" ?
- iii. Quelles sont les deux commandes de **less** qui permettent de chercher un motif dans un fichier, respectivement après et avant la position courante ?
- iv. Quelles sont les commandes qui permettent d'aller à la prochaine (resp. précédente) occurrence du motif recherché à travers tous les fichiers édités ?
- v. Comment chercher dans un fichier le caractère / ?

2.2.5. sort (1)

Cette commande permet de trier le contenu du fichier qui lui est passé en argument dans l'ordre lexicographique. Consulter le **man sort** pour avoir tous les détails. Dans l'exemple qui suit, je trie le fichier `/etc/passwd` selon l'ordre numérique sur le champ 3 (uid) puis sur le champ 4 (gid). Afin de limiter l'affichage, certaines lignes ont été volontairement enlevées du résultat.

EXEMPLE :

```
$ sort -n -k 3,4 -t ":" /etc/passwd
root:x:0:0:root:/root:/bin/bash
toor:x:0:11:toor's account:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/false
bin:x:2:2:bin:/bin:/bin/false
sys:x:3:3:sys:/dev:/bin/false
sync:x:4:65534:sync:/bin:/bin/sync
man:x:6:12:man:/var/cache/man:/bin/false
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/false
uucp:x:10:10:uucp:/var/spool/uucp:/bin/false
proxy:x:13:13:proxy:/bin:/bin/false
postgres:x:31:32:postgres:/var/lib/postgres:/bin/nologin
backup:x:34:34:backup:/var/backups:/bin/false
operator:x:37:37:Operator:/var:/bin/false
sshd:x:101:65534:./var/run/sshd:/bin/false
postfix:x:102:101:./var/spool/postfix:/bin/false
pascal:x:666:666:Pascal P.,.,.,:/home/pascal:/bin/bash
corto:x:667:666:testing account:/bin/false
nobody:x:65534:65534:nobody:/nonexistent:/bin/false
```

Exercice - Sort

- i. Triez le fichier `~/elemntBase/part2/procList` sur le champ PPID puis sur le champ PID.
- ii. Triez les processus le fichier `~/elemntBase/part2/procList` sur la taille de leur mémoire virtuelle.

2.2.6. uniq (1)

Cette commande permet de supprimer les doublons dans un fichier préalablement trié. On s'inspire de l'exemple précédent, avec cette fois ci un tri uniquement effectué sur le champ gid (dans l'ordre numérique croissant), puis nous récupérons ces gids (uniquement, avec **awk**, que nous verrons dans Section 4, « Éléments d'initiation à awk ») et supprimons les doublons :

EXEMPLE :

```
$ sort -n -k 4 -t ":" /etc/passwd | awk -F ":" '{print $4}' | uniq
0
1
2
3
8
9
10
11
12
13
32
34
37
101
666
65534
```

Exercice

- i. En utilisant `~/elemntBase/part2/procList`, donnez la liste ordonnée des processus qui ont au moins un fils.

2.3. expand, unexpand, tr

2.3.1. expand (1)

Cette commande permet de convertir les caractères de tabulations, d'un fichier passé en argument, en caractères d'espace.

```
EXEMPLE :

$ cat in
#Societe      Contact      Telephone
TOTOR  Rivière 0262 994545
HADJEE Ingar 0262 983399
HOULAN Sautron 0262 999978

$ cat in | grep -v ^# | expand -20
TOTOR      Rivière      0262 994545
HADJEE     Ingar        0262 983399
HOULAN     Sautron      0262 999978
```

2.3.2. unexpand (1)

Cette commande (symétrique de la précédente) permet de convertir les caractères d'espace, d'un fichier passé en argument en caractères de tabulations.

2.3.3. tr (1)

Cette commande permet de transcoder les caractères d'un flux d'entrée selon un modèle. Dans l'exemple, je montre comment transformer un flux en majuscules, puis un décalage (a remplacé par e, b par f, modulo les 26 lettres de l'alphabet).

```
EXEMPLE :

$ cat file.txt
Voici la 1ere ligne...
Une autre ligne, la      2 !
Puis la ligne      3

$ tr a-z A-Z < file.txt
VOICI LA 1ERE LIGNE...
UNE AUTRE LIGNE, LA      2 !
PUIS LA LIGNE      3

$ tr a-z efghijklmnopqrstuvwxyzabcd < file.txt
Vsmgm pe livi pmkri...
Uri eyxvi pmkri, pe      2 !
Pymw pe pmkri      3
```

2.4. od, split, fmt, fold

2.4.1. od (1)

Cette commande transforme le flux d'entrée (fichier ou redirection) au format octal ou hexadécimal. Dans l'exemple suivant, j'affiche le contenu du fichier file.txt sous forme de caractères et pour chacun d'eux son code hexadécimal associé :

```
EXEMPLE :

$ cat file.txt
ligne 1
une autre ligne, la      2 !
puis la ligne      3

$ od -ax file.txt # sortie ascii et hexadécimale
0000000  l  i  g  n  e  s  p  l  n  l  u  n  e  s  p  a  u  t  r
        696c 6e67 2065 0a31 6e75 2065 7561 7274
0000020  e  s  p  l  i  g  n  e  ,  s  p  l  a  h  t  s  p  2  s  p  !
        2065 696c 6e67 2c65 6c20 0961 3220 2120
0000040  n  l  p  u  i  s  s  p  l  a  s  p  l  i  g  n  e  h  t  s  p
        700a 6975 2073 616c 6c20 6769 656e 2009
0000060  3  n  l
        0a33
0000062
```

Remarque : le fichier fait 50 (= 62 en octal) octets. Le code 69h désigne la lettre i, tandis que 6ch désigne le l.

2.4.2. split (1)

Cette commande permet de découper un gros fichier en plusieurs morceaux plus petit. Dans l'exemple, je dispose d'un fichier de 3436583 octets, que je voudrais transférer sur disquettes, chacune disposant d'un mégaoctet (= 1048576 octets) de libre au plus, il faudra donc 4 disquettes. Noter, l'utilisation du préfixe pour simplifier le travail, le reste du nom est généré automatiquement :

```
EXEMPLE :

$ ls -l foo.pdf
-rw----- 1 pascal gnu 3436583 Aug 6 09:03 foo.pdf
```

```

$ split -b 1048576 foo.pdf foo-
$ ls -l foo*
-rw----- 1 pascal gnu 1048576 Aug 6 09:10 foo-aa
-rw----- 1 pascal gnu 1048576 Aug 6 09:10 foo-ab
-rw----- 1 pascal gnu 1048576 Aug 6 09:10 foo-ac
-rw----- 1 pascal gnu 290855 Aug 6 09:10 foo-ad
-rw----- 1 pascal gnu 3436583 Aug 6 09:03 foo.pdf

```

Pour reconstituer le tout il suffit de concaténer les morceaux ensemble, comme suit :

```

EXEMPLE :

$ cat foo-aa foo-ab foo-ac foo-ad > nfoo.pdf
$ ls -l nfoo.pdf
-rw----- 1 pascal gnu 3436583 Aug 6 09:14 nfoo.pdf

```

2.4.3. fmt (1)

Cette commande reformate les paragraphes du fichier qui lui est passé en argument. Dans l'exemple qui suit, je limite la longueur de la ligne à 60 caractères.

```

EXEMPLE :

$ cat file.txt
Paragraphe numéro un : ceci est un exemple de paragraphe tapé au kilomètre sans
souci de mise en page histoire de voir un peu ce qu'il se passe. Deuxième phrase
du premier paragraphe.
Une autre ligne, la 2 !
Puis la ligne 3

$ fmt -uw 60 file.txt
Paragraphe numéro un : ceci est un exemple de paragraphe
tapé au kilomètre sans souci de mise en page histoire de
voir un peu ce qu'il se passe. Deuxième phrase du premier
paragraphe. Une autre ligne, la 2 ! Puis la ligne 3

```

2.4.4. fold (1)

Cette commande contraint à une longueur donnée, chaque ligne du fichier qui lui est passé en argument. Il n'y a pas de reformatage.

```

EXEMPLE :

$ cat file.txt
Paragraphe numéro un : ceci est un exemple de paragraphe tapé au kilomètre sans
souci de mise en page histoire de voir un peu ce qu'il se passe. Deuxième phrase
du premier paragraphe.
Une autre ligne, la 2 !
Puis la ligne 3

$ fold -w 60 file.txt
Paragraphe numéro un : ceci est un exemple de paragraphe tap
é au kilomètre sans souci de mise en page histoire de voir u
n peu ce qu'il se passe. Deuxième phrase du premier paragra
ph
e.
Une autre ligne, la 2 !
Puis la ligne 3

```

2.5. paste, join, tee

2.5.1. paste (1)

Cette commande fusionne les lignes des différents fichiers (un ou plus) qui lui sont passés en argument.

```

EXEMPLE :

$ cat in1
ligne 1
line 2
ligne 3 ...
line 444

$ cat in2
line aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbbaaaaa
ligne bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

$ paste in1 in2

```

```

ligne 1 line aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbaaaaa
line 2 ligne bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ligne 3 ...
line 444

$ paste in2 in1
line aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbaaaaa ligne 1
ligne bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb line 2
      ligne 3 ...
      line 444

```

2.5.2. join (1)

Cette commande similaire à **paste**, utilise un champ (par défaut le premier) commun à chaque ligne des deux fichiers donnés en entrée, pour les fusionner en une ligne. Le champ commun n'est pas répété.

```

EXEMPLE :

$ cat in1
ligne 1
line 2
ligne 3 ...
line 444

$ cat in2
ooo line aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbaaaaa
aaa ligne bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

$ join -1 1 -2 2 in1 in2
line 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbaaaaa
ligne 3 ... bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

$ join -1 2 -2 1 in2 in1
line ooo aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbaaaaa 2
ligne aaa bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 3 ...

```

Précision : tout ce passe comme suit, il y a « alignement » sur :

```

EXEMPLE :

# in1                               in2
ligne 1
line 2                               ooo line aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa bbbbbaaaaa
ligne 3 ...                           aaa ligne bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
line 444

```

Exercice

- i. En utilisant les fichiers `~/elemntBase/part2/procList`, et `~/elemntBase/part2/ps-aux`, créez le fichier `~/elemntBase/part2/procList2` contenant les champs PID, PPID, USER, %CPU et COMMAND, en séparant les champs par des tabulations.

2.5.3. tee (1)

Cette commande copie sur la sortie standard et dans les fichiers qui lui sont passés en argument le flux en provenance de l'entrée standard. Particulièrement utile quand on veut *logger* des événements dans un fichier et en même temps les voir sur un écran.

```

EXEMPLE :

$ tee out1 out2 < file.txt
Voici la 1ere ligne...
Une autre ligne, la      2 !
Puis la ligne      3

$ cat out1
Voici la 1ere ligne...
Une autre ligne, la      2 !
Puis la ligne      3

$ cat out2
Voici la 1ere ligne...
Une autre ligne, la      2 !
Puis la ligne      3

```

2.6. Exercices

Exercice - Textprocessing (1)

- i. Trier le fichier des mots de passe en majeur sur le champ *gid* (4^e) et en mineur sur le champ *uid* (3^e).

Proposition de correction.

```
CODE :

$ sort -t ":" -n -k 4 -k 3/etc/passwd
#
root:*:0:0:Charlie &:/root:/bin/csh
toor:*:0:0:Bourne-again Superuser:/root:/bin/tcsh
daemon:*:1:1:Owner of many system processes:/root:/sbin/nologin
operator:*:2:5:System &:/sbin/nologin
pop:*:68:6:Post Office Owner:/nonexistent:/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/sbin/nologin
news:*:8:8:News Subsystem:/sbin/nologin
man:*:9:9:Mister Man Pages:/usr/share/man:/sbin/nologin
games:*:7:13:Games pseudo-user:/usr/games:/sbin/nologin
# $FreeBSD: src/etc/master.passwd,v 1.25.2.6 2002/06/30 17:57:17 des Exp $
sshd:*:22:22:Secure Shell Daemon:/var/empty:/sbin/nologin
smmsp:*:25:25:Sendmail Submission User:/var/spool/clientmqueue:/sbin/nologin
mailnull:*:26:26:Sendmail Default User:/var/spool/mqueue:/sbin/nologin
bind:*:53:53:Bind Sandbox:/sbin/nologin
cyrus:*:60:60:the cyrus mail server:/nonexistent:/sbin/nologin
uucp:*:66:66:UUCP pseudo-user:/var/spool/uucppublic:/usr/libexec/uucp/uucico
xten:*:67:67:X-10 daemon:/usr/local/xten:/sbin/nologin
pgsql:*:70:70:PostgreSQL Daemon:/usr/local/pgsql:/bin/sh
www:*:80:80:World Wide Web Owner:/nonexistent:/sbin/nologin
mysql:*:88:88:MySQL Daemon:/var/db/mysql:/sbin/nologin
pascal:*:666:666:Pascal P;:/home/pascal:/bin/tcsh
miji:*:667:666:Miji M.P.:/home/miji:/bin/tcsh
postfix:*:1001:1001:Postfix Mail System:/var/spool/postfix:/sbin/nologin
jabber:*:1002:1003:Jabber Daemon:/nonexistent:/nonexistent
stunnel:*:1003:1004:stunnel Daemon:/nonexistent:/sbin/nologin
tty:*:4:65533:Tty Sandbox:/sbin/nologin
kmem:*:5:65533:KMem Sandbox:/sbin/nologin
nobody:*:65534:65534:Unprivileged user:/nonexistent:/sbin/nologin
```

- ii. Sous *FreeBSD* le fichier */etc/passwd* contient des commentaires, lignes commençant par le symbole *#*. Reprendre le travail précédent en éliminant les lignes de commentaires.

Proposition de correction.

```
$ clear && grep -v '^#' /etc/passwd | sort -t ":" -n -k 4 -k 3
root:*:0:0:Charlie &:/root:/bin/csh
toor:*:0:0:Bourne-again Superuser:/root:/bin/tcsh
daemon:*:1:1:Owner of many system processes:/root:/sbin/nologin
operator:*:2:5:System &:/sbin/nologin
pop:*:68:6:Post Office Owner:/nonexistent:/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/sbin/nologin
news:*:8:8:News Subsystem:/sbin/nologin
man:*:9:9:Mister Man Pages:/usr/share/man:/sbin/nologin
games:*:7:13:Games pseudo-user:/usr/games:/sbin/nologin
sshd:*:22:22:Secure Shell Daemon:/var/empty:/sbin/nologin
smmsp:*:25:25:Sendmail Submission User:/var/spool/clientmqueue:/sbin/nologin
mailnull:*:26:26:Sendmail Default User:/var/spool/mqueue:/sbin/nologin
bind:*:53:53:Bind Sandbox:/sbin/nologin
cyrus:*:60:60:the cyrus mail server:/nonexistent:/sbin/nologin
uucp:*:66:66:UUCP pseudo-user:/var/spool/uucppublic:/usr/libexec/uucp/uucico
xten:*:67:67:X-10 daemon:/usr/local/xten:/sbin/nologin
pgsql:*:70:70:PostgreSQL Daemon:/usr/local/pgsql:/bin/sh
www:*:80:80:World Wide Web Owner:/nonexistent:/sbin/nologin
mysql:*:88:88:MySQL Daemon:/var/db/mysql:/sbin/nologin
pascal:*:666:666:Pascal P;:/home/pascal:/bin/tcsh
miji:*:667:666:Miji M.P.:/home/miji:/bin/tcsh
postfix:*:1001:1001:Postfix Mail System:/var/spool/postfix:/sbin/nologin
jabber:*:1002:1003:Jabber Daemon:/nonexistent:/nonexistent
stunnel:*:1003:1004:stunnel Daemon:/nonexistent:/sbin/nologin
tty:*:4:65533:Tty Sandbox:/sbin/nologin
kmem:*:5:65533:KMem Sandbox:/sbin/nologin
nobody:*:65534:65534:Unprivileged user:/nonexistent:/sbin/nologin
```

Exercice - Textprocessing (2)

- i. Remplacer le séparateur de champ (*:*) du fichier */etc/passwd* par le séparateur de tabulation horizontale *\t*. Mettre le tout dans un fichier intitulé *~/mypasswd*. Vérifier le résultat.

Proposition de correction.

```
$ cat /etc/passwd | tr ":" "\t" > ~/mypasswd
```

- ii. Vous avez sans doute remarqué un défaut d'alignement des colonnes. Proposer une solution permettant un alignement de chaque colonne, sans décalage à l'affichage. Même fichier de départ, même fichier d'arrivée

Proposition de correction.

```
$ cat /etc/passwd | tr ":" "\t" | expand -32 > ~/mypasswd
```

Exercice - Textprocessing (3)

1. « Projeter » le fichier `/etc/passwd` sur les champs `<nom, uid, gid, shell>`, le séparateur final sera le `\t`.

Proposition de correction.

```
$ cat /etc/passwd | cut -d: -f 1,3,4,7 | tr ":" "\t" > ~/mypasswd
```

3. Éléments d'initiation à sed

3.1. Principes généraux

Sed et Awk sont deux outils de *textprocessing*, invoqués de la même manière :

command [options] script filename

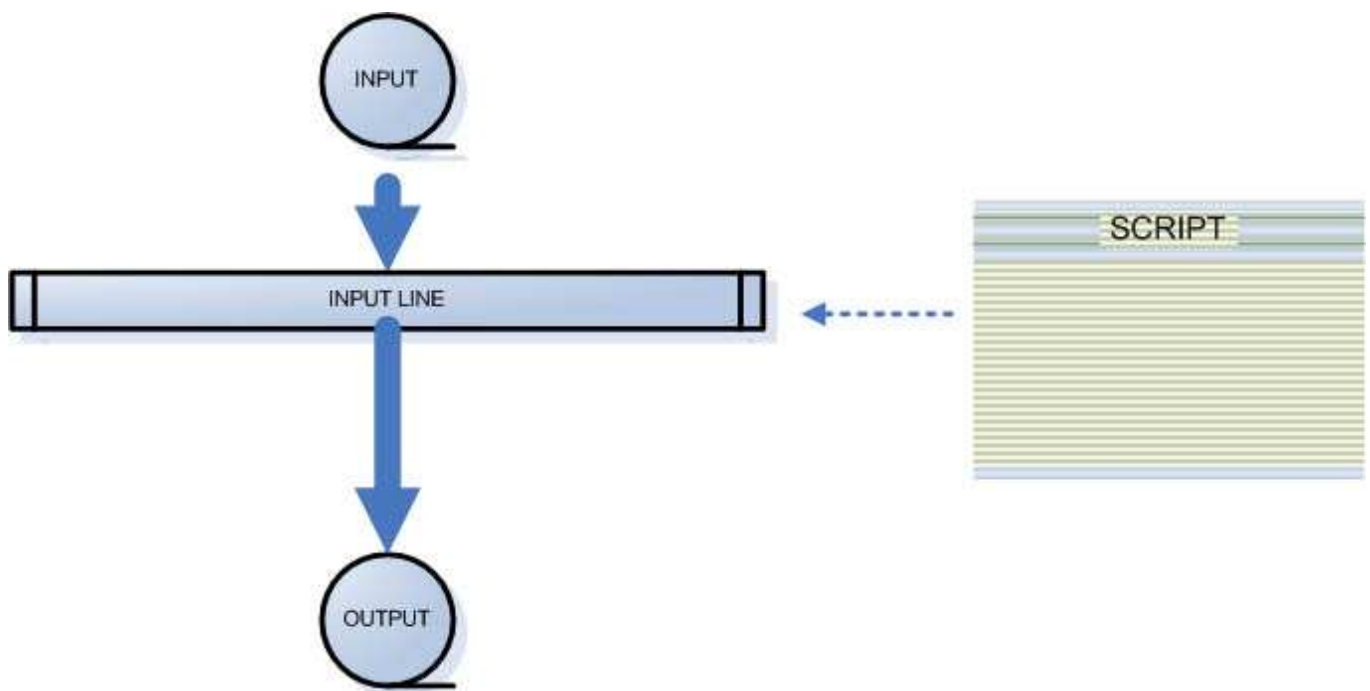
Comme la plupart des programmes UNIX, sed et awk peuvent lire sur l'entrée standard et écrire sur la sortie standard, mais aussi recevoir un flux de données depuis un fichier, et bien sûr rediriger les sorties vers un fichier.

Le script spécifie les instructions à exécuter, et doit être quoté (encadré d'apostrophes) si il est directement codé sur la ligne de commande. Une option est commune à **awk** et **sed**, `-f`, qui permet de spécifier un script externe.

sed -f scriptfile inputfile

La figure ci-dessous montre le fonctionnement de base de sed et awk. Chaque programme lit ligne par ligne le fichier d'entrée, en fait une copie, et exécute les instructions prévues dans le script sur cette copie. Les changements n'affectent pas le fichier source.

Figure 1. Fonctionnement de Sed et Awk



3.2. Commandes Sed

1. *Commande de substitution*

- La commande `s` permet d'effectuer des substitutions suivant la syntaxe : **sed -e 's/expr-régulière/remplacement/options'**
- Options
 - Sans précision, la commande ne s'applique qu'à la 1ère occurrence de chaque ligne
 - `0..9` : indique que la substitution ne s'applique qu'à la nième occurrence

o `g` : effectue les modifications sur toutes les occurrences trouvées

- Exemple : `sed -e 's/moi/toi/g' fich.moi > fich.toi` le fichier `fich.moi` est parcouru, à chaque occurrence de "moi", ce mot est remplacé par "toi" et le nouveau fichier est sauvegardé sous le nom `fich.toi`

2. Destruction ou sélection

- Cette option permet de filtrer les lignes qui satisfont une expression régulière. Ces lignes ne sont pas détruites dans le fichier d'origine, mais ne sont pas transmises en sortie.
- Par exemple, pour détruire toutes les lignes vides d'un fichier : `sed -e '/^$/d'`

3. Ajout, insertion et modification

- Pour utiliser ces commandes, il est nécessaire de les saisir sur plusieurs lignes

sed [adresse] commande\ expression

- La commande peut être :
 - o `a` pour ajout ;
 - o `i` pour insertion ;
 - o `c` pour modification
- Exemple :

```
</Neotech 3>/\ CPEN GREATTA - NEOTECH 3\ Cité scolaire du butor
```

3.3. Quelques exemples

Exemple : supprimer les lignes commençant par un commentaire dans le fichier suivant : `/etc/hosts`

```
EXEMPLE :  
$ sed -e '/^#/d' /etc/hosts > ~/myhosts
```

Le fichier résultant est nommé `~/myhosts`. L'expression utilisée est vraie pour toute chaîne commençant par le caractère `#` (marque de commentaire dans un fichier).

Remplacer toutes les occurrences de l'interface `rl0` du fichier (*FreeBSD*) `/etc/ipf.conf`, par l'interface `xl1` :

```
EXEMPLE :  
# sed -e 's/rl0/xl1/g' /etc/ipf.conf > /etc/ipf.conf.new
```

Le `#` dénote ici une opération effectuée en tant que *super*-utilisateur `root`. L'expression commence par un `s`, lequel indique une opération de substitution. Le `g`, quant à lui indique que tous les remplacements sur une ligne doivent être fait, en son absence seule la première occurrence trouvée sur la ligne serait remplacée.

Quelques expressions régulières. Ces exemples montrent l'utilisation d'expressions régulières, nous rappelons dans ce qui suit quelques méta-caractères importants :

Tableau 1. Expressions régulières

Caractère	Description	Exemple
<code>^</code>	Correspondance avec le début de ligne	<code>/^#/</code> ligne commençant par un un <code>#</code> .
<code>\$</code>	Correspondance avec la fin de ligne	<code>/^\$/</code> ligne blanche.
<code>.</code>	Correspondance avec un caractère quelconque	<code>/../</code> ligne contenant au moins deux caractères quelconques.
<code>*</code>	Correspondance avec 0 ou plusieurs caractères quelconques.	<code>/^a*/</code> ligne commençant par <code>a</code> suivit par n'importe quoi.
<code>[]</code>	Correspondance avec tout caractères de l'intervalle	<code>/[abc]\$/</code> ligne se terminant par le motif <code>abc</code> .

Donnons un autre exemple de combinaison d'expressions régulières. On cherche à imprimer (si elle existe) la fonction `main` d'un script Perl :

```
EXEMPLE :  
$ cd bin ; sed -n -e '/sub[[:space:]]*main[[:space:]]*{/ , /^}/p' *.pl | less
```

La première partie de cette expression régulière `sub[[:space:]]*main[[:space:]]*{` permet de trouver le début de la définition de la fonction `main`, tandis que la seconde `/^}/` trouve la fin du bloc de cette fonction. La classe spéciale `[[:space:]]` permet de dénoter un caractère espace ou un caractère TAB. Enfin, la commande `p` imprime les lignes qui vérifient la condition.

Quelques classes.

Tableau 2. Expressions régulières

Classe	Sémantique
[[:alnum:]]	Caractères alphanumérique : [a-zA-Z0-9].
[[:alpha:]]	Caractères alphabétiques : [a-zA-Z].
[[:blank:]]	Caractère espace ou tabulation.
[[:digit:]]	Chiffre [0-9].
[[:lower:]]	Caractères minuscules [a-z].
[[:space:]]	Espace.
[[:upper:]]	Caractères majuscules [A-Z].
[[:cntrl:]]	Caractère de contrôle.
[[:graph:]]	Caractère visibles (donc sans l'espace).
[[:print:]]	Caractère autres que les caractères de contrôle.
[[:punct:]]	caractères de ponctuation.
[[:xdigit:]]	Chiffres hexadécimaux [0-9a-fA-F].

Donnons maintenant un autre exemple faisant intervenir la mémorisation d'expression. On veut cette fois ci remplacer les occurrences d'adresses IP du fichier `/etc/hosts`, par leur adresse dans le domaine `in-addr.arpa` :

```

EXEMPLE :

$ awk -F" " '{print $1}' /etc/hosts | sed -e 's/\([[[:digit:]]*\)\.\([[[:digit:]]*\)\.\([[[:digit:]]*\)\.\([[[:digit:]]*\)/arpa.in-addr.\3.\2.\1/' -e '/^#/d'

arpa.in-addr.200.168.192
arpa.in-addr.200.168.192

arpa.in-addr.1.168.192
arpa.in-addr.1.168.192
    
```

On utilise maintenant les parenthèses de mémorisation, autour de classes. L'opérateur `*` permet d'attendre plusieurs chiffres. On reconnaît l'expression qui permet de filtrer les adresses IPv4. L'expression `\3` dénote la troisième expression mémorisée i.e. le troisième octet, tandis que `\2` désigne la deuxième expression...

On peut combiner les instructions, il suffit pour cela de rajouter autant d'option `-e` que nécessaire. Si cela devient trop complexe, on peut tout placer dans un fichier et l'appeler au besoin avec l'option `-f`.

Il est possible d'effectuer plusieurs transformations sur la même région (zone dénotée par un début et une fin : étiquettes ou numéro de lignes). Dans l'exemple qui suit nous proposons de transformer certains termes dans une région dont le début est marquée par une ligne commençant par le mot `BEGIN` et se terminant par le mot `END` (ou la fin de fichier si ce mot n'est pas trouvé).

```

EXEMPLE :

#
# ~/mycommand.sed
#

/^BEGIN/, /^END/ {
    s/[Ll]inux/GNU/Linux/g
    s/[Oo]penbsd/OpenBSD/g
    s/[Ff]reebsd/FreeBSD/g
    s/[Nn]etBSD/NetBSD/g
}

USAGE :

$ sed -n -f ~/mycommand.sed myfile.txt > res.txt
    
```

3.4. Exercices

Exercice 1

Créez un nouveau répertoire `~/elementBase/part3/`. Copiez y le fichier `ps-aux` que nous avons utilisé lors de la dernière séance.

Travail à faire.

1. Supprimer les processus dont le propriétaire est **root**, et placez le résultat dans `my_ps-aux`
2. Modifiez le nom du propriétaire **sysop**, en le remplaçant par le votre. Le résultat est placé dans `my_ps-aux`
3. Supprimez les éventuels chemins des exécutable. (`/bin/tcsh` deviendra `tcsh`)

Exercice 2

On dispose d'un répertoire web contenant des fichiers html, pour lesquels on voudrait changer toutes les occurrences des images au format gif par des images identiques au format png.

Travail à faire. Proposer un script qui effectue ces changements dans tous les fichiers. On sauvegardera de plus tous les anciens fichiers dans un sous-répertoire nommé BAK.

Proposition de correction. On se place dans le repertoire cible.

```
SCRIPT SHELL (sh) :

if [ ! -d BAK ]; then mkdir BAK; fi && \
for f in `find . -name "*.htm[l]" -print`; do
  cp $f BAK
  sed -e 's/.gif/.png/g' < $f > $f.tmp
  mv $f.tmp $f
done
```

4. Eléments d'initiation à awk

awk est un langage de script particulièrement intéressant dans l'analyse et le traitement de fichier texte ainsi que dans la génération de rapports. Nous présentons dans cette partie ces principales caractéristiques sur des exemples.

4.1. Premier exemple

On suppose vouloir extraire du fichier `/etc/passwd` les *uids*, *gids* et nom d'utilisateur, le tout trié en majeur sur l'*uid* et en mineur sur le *gid*. Bien entendu, les lignes de commentaires ne doivent pas apparaître. Voilà typiquement un travail pour **awk**.

```
EXEMPLE :

$ awk -F":" ' $0 !~ /^#/ { print $3"\t"$4"\t"$1 }' /etc/passwd | sort -n -k 1,2
0      0      root
0      0      toor
1      1      daemon
2      5      operator
3      7      bin
4      65533  tty
5      65533  kmem
7      13     games
8      8      news
9      9      man
22     22     sshd
25     25     smmsp
26     26     mailnull
53     53     bind
66     66     uucp
67     67     xten
68     6      pop
70     70     pgsq
80     80     ww
88     88     mysql
666    666    pascal
65534  65534    nobody
```

L'option `-F` de l'interpréteur **awk** permet de spécifier le séparateur de champ, ici le caractère « : ». Le reste dit que si la ligne dénotée par `$0` ne commence pas par le caractère `#` (c'est la syntaxe de *cs*, i.e. du langage *C*) alors on imprime sur la sortie standard les trois champs dénotés `$3`, `$4` puis `$1` (désignant respectivement l'*uid*, le *gid* et le nom d'utilisateur), lesquels sont repris en entrée de la commande **sort** qui trie numériquement (option `-n`) respectivement sur les champs positionnels 1 et 2 (i.e. l'*uid* et le *gid*). Noter enfin, l'utilisation des quotes qui permet d'empêcher l'évaluation des variables `$0`, `$1`, `$3` et `$4` par le *shell*.

L'utilisation de **awk** ne se limite pas aux scripts à une ligne (les fameux *one-liners*), c'est un véritable langage de programmation doté de structures de contrôles et manipulant des variables.

4.2. Scripts externes

Si le code du script **awk** devient complexe et/ou s'étend sur plusieurs lignes, il est possible de le mettre dans un fichier indépendant. Pour l'invoquer ensuite il suffit de passer l'option `-f` à l'interpréteur, comme suit :

```
EXEMPLE :

$ awk -f myscript.awk <fichier_à_traiter>
```

Blocs d'instructions

La structure typique d'un script **awk** est composé de trois blocs :

```
STRUCTURE D'UN SCRIPT awk :

BEGIN {
```

```

# bloc spécial d'initialisation, facultatif, exécuté avant que awk ne commence
# à traiter le fichier d'entrée <fichier_à_traiter>
}
{
# bloc de code exécuté à chaque ligne du fichier <fichier_à_traiter>
}
END {
# bloc spécial de terminaison, facultatif, exécuté après que awk ait terminé
# le traitement du fichier d'entrée <fichier_à_traiter>
}

```

Tout ce qui suit le caractère # est considéré comme un commentaire qui est ignoré par l'interprète **awk**.

Si nous reprenons sous forme de script externe, le *one-liners* précédent (c.f. Section 4.1, « Premier exemple »), cela peut donner :

```

CODE :

# ~/bin/myscript1.awk

BEGIN {
# block d'initialisation
FS=":" # Field Separator : positionné à ":"
OFS="\t" # Output Field Separator : positionné à "\t"
}
{
# block d'exécution
if ( $0 !~ /^#/ ) {
print $3,$4,$1
}
}
END {
# block final
}

UTILISATION :

$ awk -f ~/bin/myscript1.awk /etc/passwd | sort -n -k 1,2

```

Exemple 2. Autre exemple, trouver l'*uid* maximum dans le fichier */etc/passwd* et l'afficher ainsi que le nom de login et le *gid* du groupe correspondant.

```

CODE :

# ~/bin/myscript2.awk

BEGIN {
# block d'initialisation
FS=":" # Field Separator : positionné à ":"
OFS="\t" # Output Field Separator : positionné à "\t"
uid=0
gid=0
name=""
}
{
# block d'exécution - recherche de l'uid et du gid maximum
if ( $0 !~ /^#/ ) {
if ( $3 > uid ) {
uid=$3
gid=$4
name=$1
}
}
}
END {
# block final
print "nom = ", name, "uid max = ", uid, "gid = ", gid
}

UTILISATION :

$ awk -f ~/bin/myscript2.awk /etc/passwd
nom = nobody uid max = 65534 gid = 65534

```

Exemple 3. La commande **ps aux** permet sur un système *BSD* d'obtenir par processus, entre autre, la taille de la mémoire virtuelle utilisée (champs *vsz*) ainsi que la taille de la mémoire réelle utilisée (champs *rss*), l'unité étant le kilo-octet. On désire avoir le total pour ces deux paramètres tous processus confondus.

```

CODE :

# ~/bin/mypstot.awk

BEGIN {

```

```

# block d'initialisation
FS=" "
totvsz = 0
totrss = 0
}
{
# block d'execution
if ( $0 !~ /^USER/ ) {
    totvsz += $5
    totrss += $6
}
}
END {
# block final
printf("Total VSZ = %d\t\ttotal RSS = %d\n", totvsz, totrss)
}

UTILISATION :

$ ps aux | awk -f ~/bin/mypstot.awk
Total VSZ = 876712          total RSS = 701040

```

Expressions régulières, conditionnelles, structure conditionnelle et blocs exécutables

awk permet l'exécution conditionnelle d'un bloc. Il suffit pour cela de préfixer le bloc à exécuter par une expression à valeur booléenne, i.e. la chaîne vide et l'entier 0 sont interprétés comme la valeur booléenne « faux », le reste est « vrai ».

Les expressions régulières (ou rationnelles).

```

EXEMPLES :

/^root/ { print }      # n'imprime que les lignes commençant par le motif root

/csh$/ { print }      # n'imprime que les lignes finissant par le motif csh

/[0-9]+:[A-Za-z]*/ { print }
                    # n'imprime que les lignes contenant un motif structuré
                    # ainsi entier puis ":" puis chaîne de caractères

! /root/ { print }    # n'imprime que les lignes ne contenant pas le motif root

```

Les expressions conditionnelles.

```

EXEMPLES :

$1 == "pascal" { print } # (1) imprime la ligne si le premier champ est égale à
                        # "pascal" (égalité de chaîne de caractères)

$5 !~ /root/ { print }  # (2) imprime la ligne si le 5e champ ne contient pas 1
                        # motif root

( $1 == "foo" ) && ( $3 > 666 ) { print }
                        # (3) imprime la ligne corresp. si le 1er champ est une
                        # chaîne égale à "foo" et le 3e champ un entier plus
                        # grand strictement à 666

```

La structure conditionnelles. **awk** offre l'instruction **if**. Tous les exemples précédents peuvent ainsi être réécrit.

```

EXEMPLES :

{
    if ( $1 == "pascal" ) {
        print
    }
}

{
    if ( $5 !~ /root/ ) {
        print
    }
}

{
    if ( ( $1 == "foo" ) && ( $3 > 666 ) ) {
        print
    }
}

```

Boucles

boucle while.

```
EXEMPLE :

#
# ~/bin/test.awk
#
BEGIN {
  FS=":"
  ORS="\t"
  nbfield=0
}
{
  x = 1
  while ( x < NF ) { # NF var. spéciale donnant le nombre de champs (Number of
    print $x        # Fields) de la ligne courante
    x++
  }
  print NF, " champs lus \n"
  nbfield += NF
}
END {
  print "\ttotal : ", nbfield, " champs lus \n"
}
```

UTILISATION :

```
$ awk -f ~/bin/test.awk /etc/group
# $FreeBSD      src/etc/group,v 1.19.2.3 2002/06/30 17 57      4  champs lus
  1  champs lus
  wheel *          0          4  champs lus
  daemon *        1          4  champs lus
  kmem *          2          4  champs lus
  sys *           3          4  champs lus
  tty *           4          4  champs lus
  operator *      5          4  champs lus
  mail *          6          4  champs lus
  bin *           7          4  champs lus
  news *          8          4  champs lus
  man *           9          4  champs lus
  staff *        20          4  champs lus
  sshd *         22          4  champs lus
  smmsp *        25          4  champs lus
...
  total : 137  champs lus
```

boucle do ... while. Ici, l'évaluation de la condition se fait après la fin de la boucle.

```
EXEMPLE :

#
# ~/bin/test2.awk
#
BEGIN {
  FS=":"
  ORS="\t"
  nbfield=0
}
{
  x = 1
  do {
    print $x
    x++
  } while ( x < NF )
  print NF, " champs lus \n"
  nbfield += NF
}
END {
  print "\ttotal : ", nbfield, " champs lus \n"
}
```

UTILISATION :

```
$ awk -f ~/bin/test2.awk /etc/group
# $FreeBSD      src/etc/group,v 1.19.2.3 2002/06/30 17 57      4  champs lus
# 1  champs lus
  wheel *          0          4  champs lus
  daemon *        1          4  champs lus
  kmem *          2          4  champs lus
  sys *           3          4  champs lus
  tty *           4          4  champs lus
  operator *      5          4  champs lus
  mail *          6          4  champs lus
  bin *           7          4  champs lus
```

```

news      *      8      4 champs lus
man       *      9      4 champs lus
...
total : 117 champs lus

```

boucle for.

EXEMPLE :

```

#
# ~/bin/test3.awk
#
BEGIN {
  FS=":"
  ORS="\t"
  nbfield=0
}
{
  for ( x = 1; x < NF; x++ ) {
    print $x
  }
  print NF, " champs lus \n"
  nbfield += NF
}
END {
  print "\ttotal : ", nbfield, " champs lus \n"
}

```

UTILISATION :

```

$ awk -f ~/bin/test3.awk /etc/group
# $FreeBSD      src/etc/group,v 1.19.2.3 2002/06/30 17 57      4 champs lus
#      1 champs lus
wheel *      0      4 champs lus
daemon *     1      4 champs lus
kmem *      2      4 champs lus
sys *       3      4 champs lus
tty *       4      4 champs lus
operator *   5      4 champs lus
mail *      6      4 champs lus
bin *       7      4 champs lus
news *      8      4 champs lus
man *       9      4 champs lus
...
total : 117 champs lus

```

instructions break et continue. Ces deux intructions permettent de « rompre » le déroulement classique d'une boucle. La première permet de sortir (inconditionnellement quand elle est rencontrée) du corps de la boucle la plus interne. La seconde, quant à elle, relance une nouvelle itération en « shuntant » littéralement le reste du code de la boucle.

EXEMPLE :

```

#
# ~/bin/test4.awk
#
{
  i = 1
  while (1) { # boucle infinie
    # 'continue' conduit ici
    if ( ( i % 4 ) == 0 ) {
      i++
      continue
    }
    print "iteration " i
    if ( i > 20 ) {
      break
    }
    i++
  }
  # 'break' conduit là
  print "termine : " i
}

```

UTILISATION :

```
$ awk -f ~/bin/test4.awk
```

```

<ENTREE>
iteration 1
iteration 2
iteration 3
iteration 5

```

```
iteration 6
iteration 7
iteration 9
iteration 10
iteration 11
iteration 13
iteration 14
iteration 15
iteration 17
iteration 18
iteration 19
iteration 21
termine : 21
^C
```

Type de variables

Il existe deux types scalaires : les entiers et les chaînes de caractères (vus dans les exemples précédents) et un type structuré : tableau.

Attention, contrairement au langage *C* une chaîne de caractères n'est pas un tableau de caractères.

Opérateurs

Il s'agit des opérateurs du langage *C*.

Fonctions

AWK met à la disposition du programmeur des fonctions standard prédéfinies, dont voici quelques unes :

- **length(chaîne)** : donne la longueur de son argument, par défaut \$0
- **substr(chaîne_1,int_1,int_2)** : donne la sous chaîne de chaîne_1 commençant à la position int_1 et de longueur inférieure ou égale à int_2
- **split(chaîne, tab, car)** : éclate la chaîne, les sous chaînes sont récupérées en tab[1], tab[2]... Le caractère car sert à délimiter les différents morceaux.
- **index(chaîne_1, chaîne_2)** : donne la position de la première occurrence de chaîne_2 dans chaîne_1.

Variables spéciales

Il existe des variables prédéfinies dans awk, les voici :

- **FS** : séparateur de champ dans le fichier (par défaut espace ou tabulation), un nombre quelconque de séparateur étant équivalent à un seul. L'option -F permet de modifier à l'appel de la commande la valeur de ce séparateur.
- **RS** : séparateur d'enregistrement en entrée dans le fichier (par défaut le caractère de fin de ligne, un enregistrement étant alors une ligne)
- **OFS** : séparateur de champ en sortie (par défaut espace)
- **ORS** : séparateur d'enregistrement en sortie (par défaut le caractère fin de ligne)
- **NF** : nombre de champ de l'enregistrement courant
- **NR** : numéro de l'enregistrement en cours de traitement
- **FILENAME** : nom du fichier en cours de traitement

Variables de type tableaux

Ce type structuré permet de stocker différents éléments (de type différents dans une structure « naturellement » indexée par des chaînes de caractères. Tous les tableaux **awk** sont donc des tableaux associatifs, y compris ceux indexés par des entiers, puisqu'en **awk** un entier est représenté par sa chaîne.

La simple déclaration suivante, permet de définir un tableau :

```
EXEMPLE :

myarr[0] = "World"
myarr[1] = 72
myarr[2] = "Hello"
```

L'origine de l'indexation à 0 ou à 1 est laissée au libre choix du programmeur, **awk** ne pose aucun problème. La construction suivante permet d'itérer sur le contenu du tableau :

```
EXEMPLE :

for ( i in myarr ) {
    print myarr[i]
```

```
}
```

Toutefois, il n'y a aucune garantie sur l'ordre d'impression.

Suppression [**delete**] d'éléments dans un tableau :

EXEMPLE :

```
delete myarr[1] # suppression de l'éléments indexé par l'entier 1
```

appartenance d'un élément à un tableau [**in**] :

EXEMPLE :

```
{
  elem=72
  if ( elem in myarr ) {
    ...
  }
  else {
    ...
  }
  ...
}
```

4.3. Exercices

Exercice 1

On dispose de différents fichiers à la structure identique : une première ligne non pertinente qui pourra être ignorée, suivi d'un ensemble de lignes structurées en 13 champs séparés soit par des caractères espaces, soit par des caractères de tabulation. Voici un exemple de ligne :

```
str1 int2 int3 int4 int5 int6 int7 int8 int9 int10 int11 int12 int13
```

Travail à faire. On souhaiterait faire la somme des colonnes de chacun des fichiers et écrire les résultats obtenus dans un fichier résultat, au format suivant : file_i tot2 tot3 tot4 tot5 tot6 tot7 tot8 tot9 tot10 tot11 tot12 tot13. On pourra supposer que chaque fichier dispose d'une marque explicite de fin (ici <<EOF>>).

Réponse :

CODE :

```
BEGIN {
  # Proposition de Correction - Les fichiers possèdent une marque spéciale de
  # fin : <<EOF>>

  FS= " "      # Field Separator

  OFS = "\t"
  ORS = ""
  RT = "^D"

  MAX=12
  for ( i=0; i < MAX; i++ ) {
    mysum[i] = 0
    mytot[i] = 0
  }
}

{
  if ( $0 ~ /^<<EOF>>$/ ) {
    # Afficher les totaux des stats du fichier
    print FILENAME, " "
    for ( i=0; i < MAX; i++ ) {
      printf("%5s", mysum[i])
    }
    print "\n";

    # (re)initialiser le tableau
    for ( i=0; i < MAX; i++ ) {
      mysum[i] = 0
    }
  }
  else {
    # totaliser
    for ( i=0; i < MAX; i++ ) {
      mysum[i] += $(i+2)
      mytot[i] += $(i+2)
    }
  }
}
```

```

}
END {
  print "TOTAL", " "
  for ( i=0; i < MAX; i++ ) {
    printf("%5s", mytot[i])
  }
  print "\n";
}

UTILISATION :

$ awk -f sumstat.awk stat1 stat2 stat3
stat1      5   6   5   6   5   6   6   7   6   5   4   6
stat2      5   6   5   6   5   6   6   7   6   5   4   6
stat3     14  15   5   6   5  15   6   7   6   5  13   6
TOTAL     24  27  15  18  15  27  18  21  18  15  21  18

```

Exercice 2

Travail à faire. Récupérer l'adresse IP et MAC des interfaces réseaux de votre machine en utilisant les valeurs données par la commande **ifconfig**

Exercice 3

Travail à faire. Simuler (en partie) avec un script **awk** la commande **wc** (*word count*), qui sans option supplémentaire donne respectivement le nombre de ligne(s), de mot(s) (relativement à un séparateur prédéfini), d'octet(s) du fichier qui lui est passé en argument.

Réponse : Simuler wc.

```

CODE :

# ~/bin/wc.awk

BEGIN {
# block d'initialisation
  RS="\n"      # Record Separator
  OFS="\t"     # Output Field Separator

  line=0      # var. utilisateur
  word=0
  byte=0
}
{
# block d'execution - comptabiliser le nombre de ligne, mot, octet
  line++
  word += NF
  byte += length($0) + 1 # donne la longueur de la ligne au complet
}
END {
# block final
  print "", line, word, byte, FILENAME
}

UTILISATION :

$ wc /etc/passwd      # la commande originale
  29      79      1589 /etc/passwd

$ awk -f ~/bin/wc.awk /etc/passwd
  29      79      1589 /etc/passwd

$ wc /etc/motd        # la commande originale
  24      154     1074 /etc/motd

$ awk -f ~/bin/wc.awk /etc/motd
  24      154     1074 /etc/motd

```

Exercice 4

Travail à faire. Reprenons le fichier `my_ps` obtenu dans le chapitre précédent.

- Affichez le total de l'occupation de la mémoire de vos processus (somme des VSZ)
- Donnez le PID du programme le plus gourmand en mémoire (%MEM)
- Remplacez les heures "anglaises" en leur équivalent "français"

Références

[1] Robbins Daniel. *LPI Certification 101 exam prep, part 1*. <http://www-106.ibm.com/developerworks/edu/l-dw-linux-lpir21-i.html>.

[2] Robbins Daniel, Houser Chris, et Griffis Aron. *LPI Certification 101 exam prep, part 2*. <http://www-106.ibm.com/developerworks/edu/l-dw-linux-lpir22-i.html>.

[3] Dougherty Dale et Robbins Arnold. *sed & awk, 2nd Edition*. 1-56592-225-5. March 1997. O'Reilly. <http://www.oreilly.com/catalog/sed2/>.

[4] Robbins Daniel. *Common threads : Awk by example, Part 1 - an intro to the great language with the strange name*. <http://www-106.ibm.com/developerworks/linux/library/l-awk1.html>.

[5] Robbins Daniel. *Common threads : Awk by example, Part 2 - Records, loops and arrays*. <http://www-106.ibm.com/developerworks/linux/library/l-awk2.html>.

[6] Robbins Daniel. *Common threads : Awk by example, Part 3 - String functions and* <http://www-106.ibm.com/developerworks/linux/library/l-awk3.html>.

[1] Plus exactement, il s'agit d'un graphe acyclique, plusieurs désignations différentes pouvant aboutir à la même ressource. L'acyclicité permettant de conserver des algorithmes « simples » pour la gestion de la structure de données.

[2] Tout aussi informellement, un répertoire est une liste de couple {*inode* de l'objet, nom de l'objet}.

[3] `.` et `..` sont des liens durs sur des répertoires, mais aucun utilisateur, pas même le *super*-utilisateur *root* ne peut créer de tels liens.

[4] Au sens *Unix* un *filesystem* est une collection de fichiers. Il est organisé hiérarchiquement à l'aide de répertoires et se restreint en général à un type de support physique, typiquement le disque dur. Un *filesystem* est un aussi un espace de nommage et de contrôle d'accès aux fichiers qu'il contient.

[5] C'est quasi incorrect, en fait, la désignation ne possède aucun droit !

[6] *Rule of silence : when a program has nothing surprising to say, it should say nothing.*

[7] *Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*