Stage Unix

0. Bibliographie
1. Le système Unix

0

Bibliographie

Livres de références

- Georges Hansel, Jean-Marc Champarnaud, *Passeport pour Unix et C*, International Thomson Publishing France, 1995.
- Jean-Marie Rifflet, *La programmation sous Unix*, 3ème édition, Ediscience international, 1993.
- Jean-Marie Rifflet, *Unix : 99 exercices corrigés*, Ediscience international, 1996.

1

Système Unix

1. Système d'exploitation
2. Environnement de travail
3. Une session sous Unix
4. Le système de fichiers
5. Les éditeurs
6. Les processus
7. Le langage Shell

Système d'exploitation

Les machines que vous utilisez comprennent un seul processeur.

Un système d'exploitation est une interface entre le niveau matériel et les programmes de l'utilisateur. Il permet de gérer la mémoire, les périphériques, les processus, l'allocation au processeur, les requêtes des utilisateurs, la communication, etc ...

Il est constitué d'une gamme d'outils à la disposition de l'utilisateur dont des *interpréteurs de langage de commande.* Il comprend aussi des *appels système* (fonction dont le code est chargé en mémoire), qui sont utilisés par les programmes.

Unix

- système multi-utilisateurs
- système multi-tâches.
- interpréteurs de langages de commande (shell) : il existe plusieurs langages de commande, sh, ksh, bash, csh, tcsh, etc ...
- commandes de bases relatives à la gestion de fichiers, à la messagerie, l'archivage, etc ...
- autres outils : éditeurs de texte, compilateurs, éditeurs de liens, etc ...

Environnement de travail

Vous allez travailler sur des machines PC Dell avec Linux, distribution debian.

Utilitaires

- environnement de développement : compilateurs C, C++, Java, Caml, ...
- éditeurs : vi, vim et variantes, emacs, xemacs, nedit,...
- traitement de texte : TeX, LaTeX, Ghostview, xpdf, staroffice 5.2, ...
- mathématiques et calcul formel : scilab, octave, MuPad, Maple.

Une session Linux

Se loger

- En tant qu'utilisateur, vous allez avoir un *nom d'utilisateur* (ou encore *nom de login* et un *mot de passe* (ce sont deux chaînes de caractères avec certaines contraintes). Sur l'écran apparaît tout d'abord une fenêtre pour le nom de login, puis ensuite une autre pour le mot de passe

```
login : beal
password : xxxxxx
```

- Après avoir tapé chacune des chaînes, on appuie sur la touche `Enter` (ou retour chariot).
- Si le login et le mot de passe sont corrects, on se retrouve connecté, c'est-à-dire reconnu par le système comme un utilisateur ayant lancé une nouvelle session de travail.
- L'effet est l'ouverture d'une session dite sous X. L'affichage sur l'écran de plusieurs fenêtres est géré par un système de gestion des fenêtres ou window manager qui s'appelle Window Maker. Certaines fenêtres sont des fenêtres d'information et d'autres des fenêtres interactives dans lesquelles on va par exemple taper des commandes qui seront interprétées par le langage de commande shell.
- En général une fenêtre dite texte, ou encore un terminal alphanumérique xterm, est ouverte par défaut par le gestionnaire de fenêtre. Dans cette fenêtre apparaît le *caractère d'invite* ou *prompt*, par exemple `$`. Dans cette fenêtre tourne un shell.

Se déloger

- Attention, pour se déloger, il ne suffit pas de taper `exit` ou `logout` dans une fenêtre xterm, ce qui arrête la session attaché à cette fenêtre.
- On accède par le bouton droit de la souris à des menus déroulant. En sélectionnant `exit` ou `exit session` on est déconnecté et on retrouve l'écran d'acceuil initial.
- *Il ne faut jamais partir avant d'avoir effectué la manœuvre précédente.*
- L'environnement permet de créer plusieurs bureaux (ou écrans) par le fenêtre en haut à gauche.
- Des fenêtres en haut à droite permettre de créer une nouvelle fenêtre de type xterm et de configurer le gestionnaire de fenêtres.
- Testez les menus déroulant avec les différents boutons de la souris (en particulier le bouton de droite). Pour lancer le logiciel de navigation Netscape par exemple, sélectionner `Application`, `Net`, `Netscape`. La page d'accueil est le serveur Web des étudiants `http://etudiant.univ-mlv.fr/`

Changer le mot de passe

- Utiliser la commande `yppasswd` (Yellow Pages Password) qui permet de changer le mot de passe sur toutes les machines du réseau. Attention la modification n'est pas toujours immédiate.
- En l'absence de serveur de pages jaunes, la commande de changement de mot de passe est `passwd`.

Quelques exemples de commandes date, echo, who, finger, tty, man, wc

```
test@niznogood1:~$ date
lun sep 10 18:19:08 CEST 2001
test@niznogood1:~$ echo toto
toto
test@niznogood1:~$
```

Sur une autre machine, avec un autre prompt.

```
monge : ~ > who
bassino  pts/0    Sep  3 14:32
berstel  pts/2    Aug 27 10:49
herault  pts/5    Aug 21 14:33
ristov   pts/8    Sep 10 10:14
jyt      pts/9    Sep  5 11:12
ristov   pts/10   Sep 10 10:19
dr       pts/11   Sep  5 17:36
beal     pts/17   Sep 10 12:32
www      pts/19   Sep 10 15:49
beal     pts/20   Sep 10 16:09
sagot    pts/13   Sep 10 10:33
hivert   pts/15   Sep 10 12:19
marsan   pts/12   Sep 10 14:15
monge : ~ >
```

```
monge : ~ > finger berstel
Login: berstel                          Name: Jean Berstel
Directory: /home/institut/berstel       Shell: /bin/bash
On since Mon Aug 27 10:49 (CEST) on pts/2 from pcberstel
   3 days 5 hours idle
Mail last read Mon Sep 10 15:31 2001 (CEST)
No Plan.
monge : ~ > who am i
monge.univ-mlv.fr!beal      pts/21   Sep 10 16:14
monge : ~ > tty
/dev/pts/21
monge : ~ > man tty
--> affichage d'informations sur la commande tty
```

```
--> on en sort par q.
```

La commande **wc** permet de compter le nombre de lignes, de mots et de caractères d'un fichier.

```
monge : ~ > wc /etc/passwd
    196     413   12430 /etc/passwd
```

Clavier, caractères de contrôle

- Certains caractères ont une action spéciale, par exemple l'appui sur la touche ← ou backspace provoque l'effacement du dernier caractère entré sur la ligne de commande.
- On peut modifier ces effets en reparamétrant la liaison entre le clavier et l'ordinateur par la commande **`stty`**.
- L'option **`-a`** de la commande **`stty`** permet de connaître l'état courant de la liaison.

```
test@niznogood1:~$ stty -a
speed 38400 baud; rows 51; columns 91; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = <undef>; eol2 = <undef>; start = ^Q; stop = ^S;
susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O;
min = 1; time = 0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr
icrnl ixon -ixoff -iuclc-ixany -imaxbel
test@niznogood1:~$
```

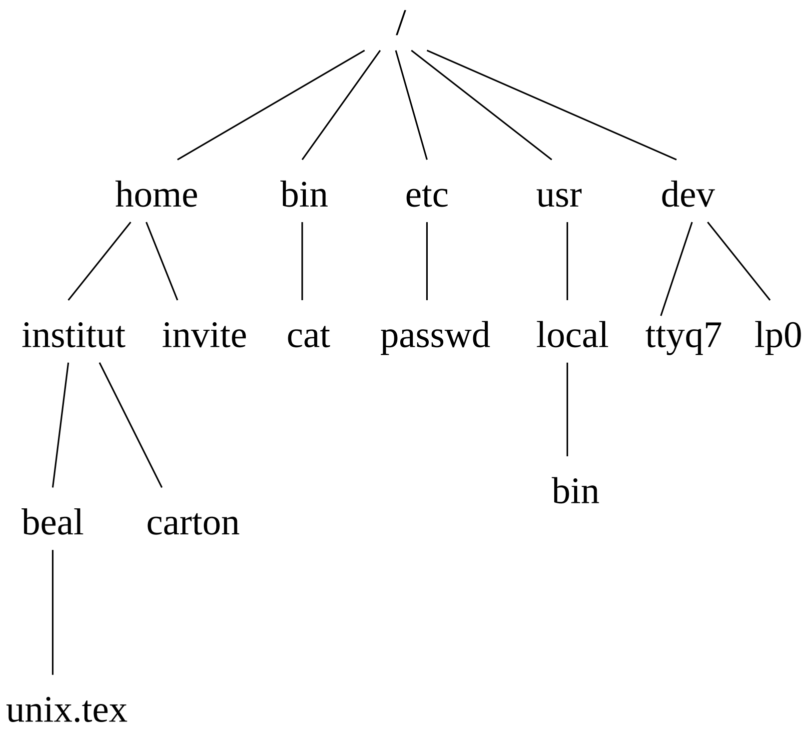
- Les principaux caractères de contrôle sont
  - **`erase`**, efface de dernier caractère, backspace;
  - **`kill`**, efface la ligne, ctrl-u
  - **`eof`**, ferme un flux d'entrée en lecture, crtl-d
  - **`newline`**, valide une commande, crtl-j
  - **`intr`**, interruption, crtl-c
  - **`quit`**, interruption avec image mémoire, crtl-\

```
test@niznogood1:~$ sleep 1000
 --> frappe de crtl-c
test@niznogood1:~$ stty intr w
test@niznogood1:~$ sleep 1000
 --> frappe de w
test@niznogood1:~$
```

Le système de fichiers

L'arbre des fichiers

- Un fichier sous Unix est non structuré.
- Une vision approchée du système de fichiers sous Unix est celle d'un arbre. On verra par la suite que ce n'est pas tout à fait vrai.
- Exemple d'arborescence.



- Tout nœud qui n'est pas une feuille de l'arbre est appelée *catalogue* (directory). Les feuilles sont soit des catalogues, soit des fichiers ordinaires, soit des fichiers spéciaux (désignant des périphériques).
- Cette structure permet de *référencer* un fichier ou un catalogue de plusieurs manières.

Références d'un fichier

- *Référence absolue* : Tout fichier ou catalogue peut être désigné par le chemin qui permet d'y accéder à partir de sa racine. Cette référence s'appelle référence absolue.

```
/home/institut/beal
/home/institut/beal/unix.tex
/dev/ttyq7
/
```

- *Référence relative* : A chaque instant, un utilisateur logé a un catalogue de travail. Il se situe virtuellement sur un nœud catalogue de l'arborescence. On connaît ce catalogue par la commande **`pwd`** (print working directory).

```
monge : ~/Recherche > pwd
monge : ~/Recherche > /home/institut/beal/Recherche
```

  Tout fichier ou catalogue peut être référencé relativement à ce catalogue de travail. Une référence relative ne commence pas par le caractère `/`.

```
Article/Determinisation/det.tex
```

- *Les références* `.` *et* `..` : La référence `.` désigne le catalogue lui-même et la référence `..` désigne le catalogue père.

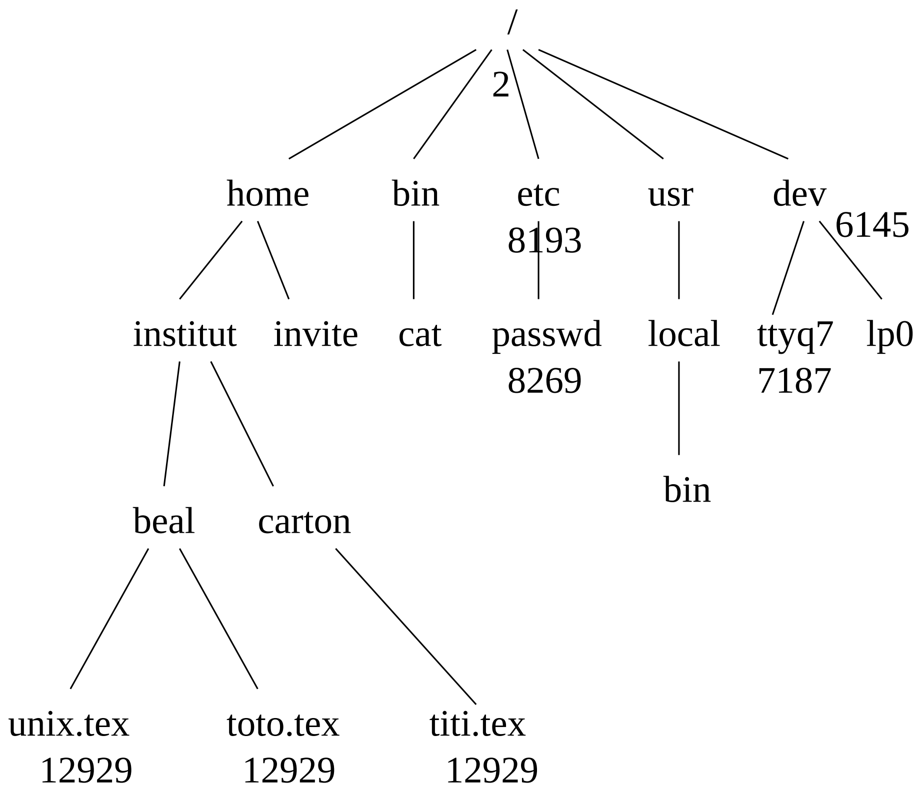
```
Article/Determinisation/det.tex
../Recherche/Article/Determinisation/det.tex
./Article/Determinisation/det.tex
../../beal/Recherche/Article/Determinisation/det.tex
```

- Chaque utilisateur se retrouve, après s'être logé, dans son catalogue privé.

```
/home/institut/beal
```

Le DAG[1] *des fichiers*

Nous allons donner une description plus précise de l'arborescence des fichiers.



A chaque fichier (ou catalogue) sur le disque est associé un *i-nœud*. C'est un bloc d'information écrit sur le disque qui contient les informations suivantes

- la taille du fichier ou catalogue,
- les adresses des blocs du disque ou est stocké le fichier ou le catalogue,
- l'identification du propriétaire et les droits d'accès,
- le type du fichier (catalogue, fichier ordinaire ou spécial),
- le nombre de liens du fichier, c'est-à-dire le nombre de fois ou il figure comme nom dans un catalogue.
- d'autres informations.

[1] Directed Acyclic Graph

Le i-nœud ne contient pas de nom ou référence quelconque du fichier ou catalogue.

On peut alors préciser le contenu d'un catalogue. C'est une liste de couples **`(numero de i-nœud,nom)`**, où **`nom`** est un nom de fichier ou catalogue. Par exemple,

- le catalogue de référence absolue **`/home/institut/beal`** contient le couple

  ```
  (12929,unix.tex)
  ```

- le catalogue de référence absolue **`/home/institut/carton`** contient le couple

  ```
  (12929,titi.tex)
  ```

- A un fichier (ou catalogue) physique est associé un seul i-nœud et un seul numéro de i-nœud. Le numéro de i-nœud s'appelle l'index. Plusieurs couples peuvent avoir le même numéro et sont donc associés au même fichier (ou catalogue) physique. Chaque couple est appelé un *lien*.

- Le i-nœud d'index 12929 a un nombre de liens égal à 3. Les deux fichiers de référence absolue

  ```
  /home/institut/beal/unix.tex
  /home/institut/carton/titi.tex
  ```

  sont donc les mêmes (un seul emplacement mémoire sur le disque).

- Combien y a t-il de liens sur le catalogue de référence absolue **`/home/institut/beal`** ?

Les fichiers spéciaux

A chaque ressource (terminal, imprimante, lecteur de disquette, disque logique, ...) est associé un fichier spécial. On le trouve dans le catalogue `/dev`. Sa taille est nulle. Les renseignements intéressants sont contenus dans le i-nœud. Il contient les informations suivantes

- propriétaire, groupe, droits d'accès,
- le *majeur* est un nombre entier qui indique la classe de ressource associée. Un terminal peut avoir pour majeur le nombre 3.
- le *mineur* est un nombre entier qui indique un exemplaire particulier de cette classe.
- le *mode* du fichier spécial, bloc ou caractère, qui indique le mode des lectures et écritures.

```
monge : ~ > tty
/dev/ttyp2
monge : ~ > ls -s  /dev/ttyp2
   0 ttyp2                 --> indique la taille
monge : ~ > ls -l ttyp2
crw--w----   1 beal     tty        3,   2 sep 12 11:58 ttyp2
monge : ~ > mesg n
 --> empêche les autres utilisateurs d'écrire sur l'écran
monge : ~ > ls -l ttyp2
crw-------   1 beal     tty        3,   2 sep 12 11:58 ttyp2
```

Manipulations de base

On va voir les manipulations de base sur les fichiers qui comprennent les déplacements dans l'arborescence.

- La commande **cd** (change directory) permet de se déplacer dans l'arbre des fichiers.

```
monge : ~ > pwd
/home/institut/beal
monge : ~ > cd ..
monge : /home/institut > pwd
/home/institut/
monge : /home/institut >  cd iznogood
bash: iznogood: Aucun fichier ou répertoire de ce type.
monge :  /home/institut >  cd
monge : ~ >              --> retour à la maison
monge : ~ > cd ./../../institut/beal/Recherche
monge : ~/Recherche >
```

- Les commandes **cat** et **more** permettent l'affichage d'un fichier texte. On sort de more par **q**.

```
monge : ~ > echo coucou coucou > toto
monge : ~ > ls -l toto
-rw-rw-r--   1 beal     beal           14 sep 12 12:28 toto
monge : ~ > cat toto
coucou coucou
monge : ~ >
```

- La commande **ls** a de nombreuses options que l'on peut voir par la commande **man** (taper **man ls**). Sans option, elle affiche la liste des fichiers et catalogues qui ne commencent pas par un point. L'option **-a** permet d'avoir la liste complète de ces fichiers qui sont en général des fichiers d'initialisation et de paramétrage. L'option **-i** permet d'avoir les index des i-nœuds.

```
monge : ~ >  ls -al
drwxr-xr-x   4 root   root     1024 avr 15  1999 ..
drwxr-xr-x   2 beal   beal     1024 sep 22  1999 .Emacs
-rw-------   1 beal   beal      115 sep 12 10:04 .Xauthority
-rw-r--r--   1 beal   beal     1147 avr 27  1999 .Xdefaults
-rw-------   1 beal   beal    13244 sep 12 10:05 .bash_history
-rw-r--r--   1 beal   beal       24 avr 12  1999 .bash_logout
-rw-r--r--   1 beal   beal      307 jun 19  1999 .bash_profile
drwxr-xr-x   7 beal   beal     1024 fév  4  2001 Enseignement
drwxr-xr-x   7 beal   beal     1024 sep 12  2000 Recherche
-rw-r-x---   1 beal   beal      658 avr 11 19:59 dessay
-rw-r-x---   1 beal   beal   111482 aoû  5 18:01 entretien_diesel.html
-rw-r-x---   1 beal   beal       14 sep 12 12:28 toto
monge : ~ >
```

- La commande Unix **find** permet de rechercher un fichier dans une sous-arborescence d'un catalogue (faire **man find**).

```
[beal@localhost Unix]$ ls Cat
toto1.c  toto2.c
[beal@localhost Unix]$ find Cat -name toto1.c -print
Cat/toto1.c
[beal@localhost Unix]$
```

- La commande Linux **locate** permet d'obtenir les références absolues de tous les fichiers contenant la chaîne indiquée. La réponse est rapide car elle consulte une table de hachage précalculée régulièrement (par **updatedb**). Sur un portable (car sinon beaucoup d'autres personnes ont des fichiers de nom toto).

```
[beal@localhost Unix]$ locate toto
locate: warning: database '/var/lib/locatedb' is more than 8 days o
/home/beal/#toto.tex#
/home/beal/.#toto.tex
/home/beal/Enseignement/Calculabilite/toto
/home/beal/Java/Programmes/essai/toto
```

```
/home/beal/Java/Programmes/fichiers/toto
/home/beal/Java/Programmes/flots/toto
/home/beal/Java/Programmes/serialisation/toto
/home/beal/Java/toto.java
[beal@localhost Unix]$
```

Manipulations de base suite

- La copie physique d'un fichier se fait avec la commande **cp**. Ceci crée deux fichiers distincts, avec des i-nœuds distincts, dont le contenu est identique.
- La création d'un lien supplémentaire sur un fichier déjà existant se fait par la commande **ln**. Il n'y a pas création d'un nouvel i-nœud mais seulement d'un nouveau lien. Le nombre de liens est incrémenté dans le i-nœud du fichier correspondant. Plusieurs références désignent alors un même fichier.

```
monge : ~ >  ls -ld
drwx------  36 beal     beal         5120 sep 12 12:28 .
monge : ~ >  ls toto
toto
monge : ~ >  ln toto tutu
monge : ~ >   cat tutu
coucou coucou
monge : ~ >  ls -il toto tutu
 211163 -rw-rw-r--   2 beal   beal   14 sep 12 12:28 toto
 211163 -rw-rw-r--   2 beal   beal   14 sep 12 12:28 tutu
monge : ~ >  cp toto titi
monge : ~ >  ls -il toto tutu titi
211191 -rw-rw-r--   1 beal   beal   14 sep 12 13:00 titi
211163 -rw-rw-r--   2 beal   beal   14 sep 12 12:28 toto
211163 -rw-rw-r--   2 beal   beal   14 sep 12 12:28 tutu
```

- La commande **rm** (remove) permet de détruire un lien sur un fichier (et non pas forcément le fichier lui même).

```
monge : ~ > rm toto
rm: détruire ‘toto’? y                     --> alias sur rm -i
monge : ~ > cat toto
cat: toto: Aucun fichier ou répertoire de ce type.
monge : ~ > cat tutu
coucou coucou
monge : ~ >  ls -il toto tutu titi
```

```
ls: toto: Aucun fichier ou répertoire de ce type.
211191 -rw-rw-r--   1 beal   beal   14 sep 12 13:00 titi
211163 -rw-rw-r--   1 beal   beal   14 sep 12 12:28 tutu
```

- Un fichier est physiquement détruit lorsque son nombre de liens devient nul.
- La commande `mv` (move) permet de changer le nom d'un fichier dans un catalogue.

```
monge : ~ >  ls -il
211163 -rw-rw-r--   1 beal   beal   14 sep 12 12:28 tutu
monge : ~ >  mv tutu toutou
monge : ~ >  ls -il
211163 -rw-rw-r--   1 beal   beal   14 sep 12 12:28 toutou
```

- La création et suppression de catalogues se font par **`mkdir`** (make directory) et **`rmdir`** (remove directory).

```
monge : ~ >  mkdir Cat
monge : ~ >  mv toutou Cat/
monge : ~ >  cd Cat
monge : ~/Cat >  ls
toutou
monge : ~/Cat > cd
monge : ~ >  ls Cat
toutou                           ---> nouvelle action de ls
```

- Des caractères spéciaux du shell permettent de désigner un ensemble de fichier (voir plus loin).

```
monge : ~ > ls t*
titi toutou
 --> affiche la liste des fichiers (ou catalogues)
 --> commençant par t
```

Droits d'accès

- Les droits s'appliquent à trois catégories d'utilisateurs.
  - Le propriétaire du fichier (lettre `u` pour user)
  - Les membres du groupe du propriétaire (lettre `g` pour group)
  - Les autres utilisateurs (lettre `o` pour others)
  - Il existe un superutilisateur (root) qui a tous les droits.
- On distingue trois types de droits qui n'ont pas le même sens lorsqu'ils s'appliquent à un fichier ou à un catalogue.
  - Le droit en lecture (lettre `r` pour read) indique pour un fichier que l'on peut le lire, l'afficher, pour un catalogue que l'on peut lire la liste des fichiers ou catalogues qu'il contient.
  - Le droit en écriture (lettre `w` pour write) indique pour un fichier que l'on peut le modifier et pour un catalogue que l'on peut lui ajouter des fichiers ou supprimer les liens de fichiers dedans.
  - Le droit en exécution (lettre `x` pour exécute) indique pour un fichier qui est un programme exécutable que l'on est autorisé à l'exécuter. Pour un catalogue, le droit d'exécution indique que l'on peut se positionner sur lui et le traverser pour avoir accès à des fichiers qu'il contient.
- On peut changer les droits des fichiers ou catalogues dont on est propriétaire.

```
monge : ~/Cat > ls -l
total 1
-rw-rw-r--   1 beal   ens   14 sep 12 12:28 tutu
monge : ~/Cat > chmod u-r tutu
monge : ~/Cat > cat tutu
cat: tutu: Permission non accordée.
```

- On utilise souvent le code octal pour changer les droits (4 pour lecture, 2 pour écriture, 1 pour exécution).

```
monge : ~/Cat > chmod 750 tutu
monge : ~/Cat >  ls -l
total 1
-rwxr-x---   1 beal  ens   14 sep 12 12:28 tutu
```

Liens symboliques

Jusqu'à présent, on a considéré qu'il n'existait qu'un seul disque avec des index d'i-nœuds distincts. En fait, un système peut comprendre plusieurs disques, chacun pouvant être partitionné en plusieurs disques logiques. Chaque disque logique a sa propre table des i-nœuds.

Un disque est *monté* lorsqu'un catalogue de l'arbre des fichiers est en réalité le catalogue racine du disque logique. Ceci se fait de façon transparente pour l'utilisateur.

```
[beal@localhost beal]$ cat /etc/fstab
/dev/hda5               /               ext2    defaults
/dev/hda6               swap            swap    defaults
/dev/fd0                /mnt/floppy     msdos    noauto,user
/dev/cdrom              /mnt/cdrom      iso9660 noauto,ro
none                    /proc           proc    defaults
[beal@localhost beal]$
```

Le premier champ indique le périphérique bloc ou système de fichier distant à monter. Le deuxième champ indique le point de montage.

```
[beal@localhost beal]$ mount /mnt/floppy     --> montage
[beal@localhost beal]$ umount /mnt/floppy    --> démontage
```

- Il n'est pas possible de créer un lien sur un fichier ou catalogue appartenant à un disque logique différent.
- Un *lien symbolique* est un fichier `f1` qui contient la référence d'un autre fichier `f2`. Toute opération sur `f1` se fait alors sur `f2`.

```
[beal@localhost beal]$ ls -l f1
-rw-rw-r--   1 beal     beal           14 sep 12 15:37 f1
[beal@localhost beal]$ ln -s f1 f2
  --> création du lien symbolique
[beal@localhost beal]$ ls -l f1 f2
-rw-rw-r--   1 beal     beal           14 sep 12 15:37 f1
lrwxrwxrwx   1 beal     beal            2 sep 12 15:37 f2 -> f1
[beal@localhost beal]$ cat f1
coucou coucou
[beal@localhost beal]$ cat f2
coucou coucou
[beal@localhost beal]$ mv f1 toto
[beal@localhost beal]$ cat f2
cat: f2: Aucun fichier ou répertoire de ce type.
[beal@localhost beal]$ ls -l f2 toto
lrwxrwxrwx   1 beal     beal            2 sep 12 15:42 f2 -> f1
-rw-rw-r--   1 beal     beal           14 sep 12 15:37 toto
[beal@localhost beal]$
```

L'éditeur vi

Pour lancer l'éditeur vi, on peut lancer

- `vi`,
- `vi toto`,
- `vi + toto`. Dans ce cas on se place à la fin du fichier `toto`.

L'éditeur vi a deux modes :

- le mode *commande*, dans lequel on est au moment de l'appel.
- le mode *insertion*, qui permet d'insérer directement dans le *tampon* ce qui est frappé au clavier.

Le passage du mode commande au mode insertion se fait par la frappe des caractères suivants

- `a`, `A` (insertion après le curseur, insertion en fin de ligne),
- `i`, `I` (insertion avant le curseur, insertion en début de ligne),
- `o` (insertion à la ligne suivante),
- `O` (insertion à la ligne précédente),

Le passage du mode commande au mode insertion se fait par la frappe du caractère `Esc`.

En mode insertion, on peut effacer ce qu'on vient d'écrire par le caractère backspace et on peut insérer un caractère spécial par `ctrl-v`. Par exemple, pour insérer le caractère `ctrl-a`, on frappe `ctrl-v ctrl-a` et on voit dans le tampon `^A`.

Dans le mode commande, on distingue deux moyens pour demander l'exécution d'une action. La plupart des commandes de recherche, remplacement, sauvegarde, se font en tapant d'abord `:`. Le curseur se positionne alors automatiquement au bas de l'écran et on peut taper une ligne de commande validée par un retour chariot. Il s'agit alors en fait de commandes de l'éditeur ligne `ed`.

- Quelques commandes de déplacement du curseur
  - `h,j,k,l`. On peut aussi utiliser les flèches.
  - `$` place le curseur en fin de ligne.
  - `G` place le curseur en fin de fichier.
  - `^F` : retour à la page précédente.
  - `^B` : avance à la page suivante.
- Quelques commandes d'effacement et de remplacement
  - `x` efface le caractère courant.
  - `dw` efface le mot courant.
  - `dd` efface la ligne courante.
  - `5dd` efface 5 lignes.
- Quelques commandes de l'édition en ligne
  - `:$` va à la fin du fichier.
  - `:r toto` ajoute le contenu du fichier `toto` après le curseur.
  - `:w` sauvegarde du tampon dans le fichier (`toto` si on a lancé `vi toto`).
  - `:w titi` sauvegarde du tampon dans le fichier `titi`.
  - `:wq` sauvegarde et sortie de vi.
  - `:q!` sortie de vi sans sauvergarde.

- Quelques commandes de recherche de motifs
  - `:/coucou` recherche la prochaine occurrence de `coucou`.
  - `:1,10s/coucou/doudou/g` remplace toutes les occurrences de `coucou` par `doudou` dans les lignes 1 à 10.
  - `:1,$s/coucou/doudou/g` remplace toutes les occurrences de `coucou` par `doudou` dans toutes les lignes.

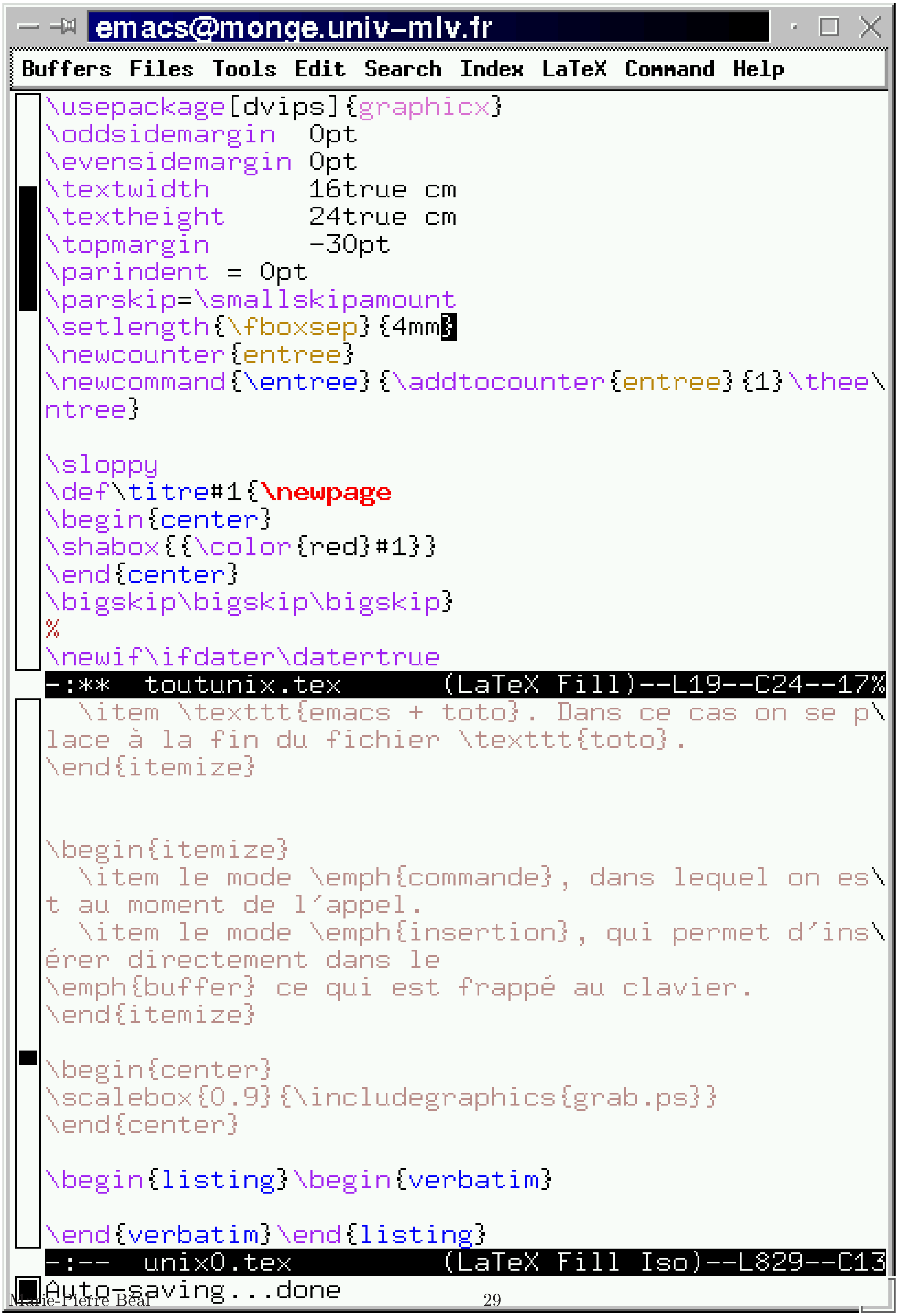
L'éditeur emacs

L'éditeur emacs est beaucoup plus puissant que vi. Il est aussi beaucoup plus gourmand en mémoire. La différence de modes (insertion, commande) disparaît. Les caractères frappées sont pour la plupart directement insérés dans le tampon. On peut aussi en particulier positionner le curseur à un endroit quelconque du texte en cliquant sur le bouton gauche de la souris.

Beaucoup de commandes sont accessibles par des menus et les menus sont configurables dans un fichier `.emacs` qui comprend des commandes en lisp.

Pour lancer l'éditeur emacs, on peut lancer

- `emacs`,
- `emacs toto`,
- `emacs + toto`. Dans ce cas on se place à la fin du fichier `toto`.


emacs@monge.univ-mlv.fr
Buffers Files Tools Edit Search Index LaTeX Command Help
\usepackage[dvips]{graphicx}
\oddsidemargin 0pt
\evensidemargin 0pt
\textwidth   16true cm
\textheight   24true cm
\topmargin   -30pt
\parindent = 0pt
\parskip=\smallskipamount
\setlength{\fboxsep}{4mm}
\newcounter{entree}
\newcommand{\entree}{\addtocounter{entree}{1}\thee\
ntree}
\sloppy
\def\titre#1{\newpage
\begin{center}
\shabox{{\color{red}#1}}
\end{center}
\bigskip\bigskip\bigskip}
%
\newif\ifdater\datertrue
-:** toutunix.tex (LaTeX Fill)--L19--C24--17%
\item \texttt{emacs + toto}. Dans ce cas on se p\
lace à la fin du fichier \texttt{toto}.
\end{itemize}
\begin{itemize}
\item le mode \emph{commande}, dans lequel on es\
t au moment de l'appel.
\item le mode \emph{insertion}, qui permet d'ins\
érer directement dans le
\emph{buffer} ce qui est frappé au clavier.
\end{itemize}
\begin{center}
\scalebox{0.9}{\includegraphics{grab.ps}}
\end{center}
\begin{listing}\begin{verbatim}
\end{verbatim}\end{listing}
-:-- unix0.tex (LaTeX Fill Iso)--L829--C13
Auto-saving...done

Pour décrire les commandes de emacs, on note

- `C-p` au lieu de `ctrl-p` (rappel : la touche `ctrl` et `p` sont frappées en même temps).
- `M-p` au lieu de `alt-p`.

Certaines commandes s'obtiennent par une combinaison de touches qui provoquent l'ouverture d'un mini tampon au bas de la page. La commande est alors complétée dans le mini tampon. Ainsi

```
C-x i
```

Affiche un mini tampon avec

```
Insert file: /mnt/monge/sda1/institut/beal/Unix/
```

On indiquera ci-dessous de façon raccourcie

```
C-x i toto
```

La frappe de `C-g` permet d'annuler une commande en cours.

Quelques commandes de base

- `C-x C-s` sauvegarde du tampon.
- `C-x C-c` sortie de emacs.
- `C-x C-f toto` ouverture du fichier toto.
- `C-h` de l'aide
- `C-h t` tutorial.
- `C-x 2` sépare le tampon en deux.
- `C-x 1` remets une seul tampon.
- `C-s coucou` recherche la prochaine occurence du mot coucou. La recherche est incrémentale. Elle commence dès la frappe des premières lettres de `coucou`.
- `C-s C-s` répétition de la recherche.

- `M-% coucou doudou` recherche-remplacement des occurrences de `coucou` par `doudou`.

Nous renvoyons à la liste des commandes distribuée pour les autres commandes.

Il est possible de sélectionner une portion de texte à la souris et d'effacer la zone par un double clic sur le bouton de droite.

Compression et archivages

- Il existe plusieurs commandes de compression et décompression possibles. Elles correspondent à des algorithmes différents.

```
gzip toto
monge : ~/Unix > gzip toto
monge : ~/Unix > ls tot*
toto.gz              --> compression effectuee
monge : ~/Unix > gunzip toto
monge : ~/Unix >     --> pour decompresser
```

  Les commandes **gzip** et **gunzip** sont les mêmes.

```
monge : ~/Unix > ls -il /bin/gunzip    /bin/gzip
30756 -rwxr-xr-x    3 root  48368  /bin/gunzip
30756 -rwxr-xr-x    3 root  48368  /bin/gzip
monge : ~/Unix >
```

- L'archivage peut se faire au moyen de la commande **tar**. L'archive est en général ensuite compressée. Ci-dessous le nom de l'archive du catalogue **Cat** est **titi.tar**.

```
monge : ~ > tar cvf titi.tar Cat
Cat/
Cat/toto
Cat/tutu
monge : ~ > tar xvf titi.tar
Cat/
Cat/toto
Cat/tutu
monge : ~ >
```

La communication

Il est possible d'envoyer des messages électroniques, des fichiers, de se connecter et de travailler sur des sites distants, etc ... par le protocole IP.

- Une adresse IP d'une machine peut avoir deux formes.
  - Une forme littérale (`monge.univ-mlv.fr`)
  - Une forme numérique (`193.55.63.80`)
- La commande **`nslookup`** permet d'obtenir ces adresses.

```
[beal@localhost Unix]$ nslookup monge.univ-mlv.fr
Server:  pixel.univ-mlv.fr
Address:  193.55.46.10

Name:    monge.univ-mlv.fr
Address:  193.55.63.80

[beal@localhost Unix]$ nslookup igm.univ-mlv.fr
Server:  pixel.univ-mlv.fr
Address:  193.55.46.10

Name:    gaspard.univ-mlv.fr
Address:  193.55.63.81
Aliases:  igm.univ-mlv.fr

[beal@localhost Unix]$ nslookup lix.polytechnique.fr
Server:  pixel.univ-mlv.fr
Address:  193.55.46.10

Non-authoritative answer:
Name:    lix.polytechnique.fr
Address:  129.104.11.2
```

- La commande **`ping`** permet de voir si une machine est en vie (ou du moins si elle répond).

```
[beal@localhost Unix]$ ping monge
```

```
PING monge (193.55.63.80): 56 data bytes
64 bytes from 193.55.63.80: icmp_seq=1 ttl=251 time=280.0 ms
64 bytes from 193.55.63.80: icmp_seq=2 ttl=251 time=230.0 ms
64 bytes from 193.55.63.80: icmp_seq=3 ttl=251 time=210.0 ms
--- monge ping statistics ---
7 packets transmitted, 6 packets received, 14% packet loss
round-trip min/avg/max = 210.0/226.6/280.0 ms
```

- Les adresses numériques se composent de quatre nombre compris entre 0 et 255 séparés par des points. Plusieurs adresses IP sont regroupées en un réseau IP. Tous les ordinateurs d'un tel réseau ont des adresses IP avec une partie commune. Cette partie commune initiale référencie le réseau alors que l'autre partie référencie la machine.
- Une adresse d'un utilisateur est le nom de login suivant de `@` et de l'adresse de la machine.
  - `beal@monge.univ-mlv.fr`
  - `Marie-Pierre.Beal@univ-mlv.fr`
- Pour envoyer un courrier électronique, la commande de base est la commande `mail`. On peut aussi utiliser d'autres commandes (`elm`). On recommande le navigateur netscape qu'il faut configurer avant la première utilisation.

```
[beal@localhost]$ cat lettre
coucou
[beal@localhost]$ mail -s ``bonjour'' carton@univ-mlv.fr < lettre
    --> envoie à carton le contenu du fichier lettre

[beal@localhost]$ mail carton@univ-mlv.fr
subject : bonjour
coucou
.
 --> on peut aussi terminer par ctrl-d
cc : beal
```

Les processus

Processus et programme

Un processus est l'exécution d'un programme.

Il s'agit donc d'une notion dynamique contrairement à celle de programme qui est statique. Un programme est une suite de caractères dans un fichier qui doit représenter une suite d'instructions. Un processus est créé à un certain moment. A un moment donné il meurt, soit lorsque le programme qu'il exécute se termine soit lorsqu'il est interrompu lors de la réception d'un signal par exemple.

Il existe des systèmes d'exploitation mono processus (MSDOS par exemple). Unix autorise l'exécution concurrente de plusieurs processus.

Le processus de numéro zéro est l'ordonnanceur (sceduler). À intervalles de temps réguliers (millièmes de seconde), il alloue le processeur à tel ou tel processus. Divers algorithmes de choix sont possibles. Le processus de numéro 1 (init) associe à chaque terminal un processus `getty` qui sera le père du shell créé lors de la connexion au terminal.

La commande ps

- Elle donne la liste des processus de l'utilisateur qui sont en cours d'exécution attaché au terminal.
  - L'option `a` donne la liste des processus de tous les utilisateurs.
  - L'option `l` donne des informations complètes avec, entre autre, le numéro des processus et le numéro de leur père.

– L'option `x` affiche les processus sans terminal d'attachement.
– L'option `t` donne les processus attaché à un terminal donné
– L'option `--user` donne les processus de l'utilisateur donné attachés à tous les terminaux.

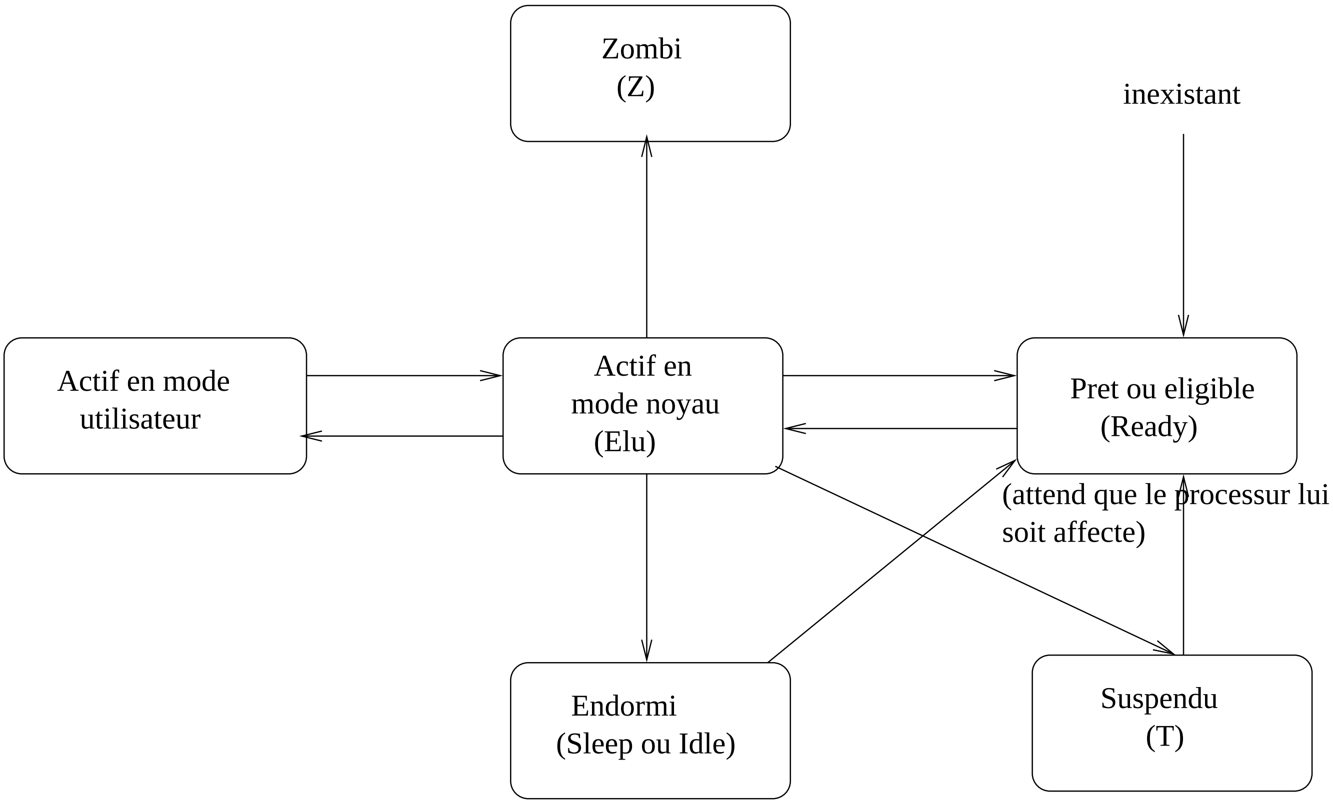
- La commande `top` donne la liste de tous les processus avec mise à jour régulière.

```
[beal@localhost beal]$ ps
  PID TTY STAT TIME COMMAND
  608  p1 S    0:00 bash
  690  p2 S    0:00 bash
  923  p3 S    0:00 bash
 2143  p3 S    0:11 emacs toutunix.tex toutunixNB.tex unix0.tex
 2181  p4 S    0:00 xdvi.bin -name xdvi toutunix.dvi
 2190  p4 S    0:00 gs -sDEVICE=x11 -dNOPAUSE -q -dDEVICEWIDTH=623 -
 2407  p2 R    0:00 ps
[beal@localhost beal]$ tty
/dev/ttyp2
[beal@localhost beal]$ ps tttyp2
  PID TTY STAT TIME COMMAND
  690  p2 S    0:00 bash
 2421  p2 R    0:00 ps tttyp2
[beal@localhost beal]$ ps l
PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
 608   605   0   0   1280   832 read_chan   S    p1  0:00 bash
 690   689  15   0   1384   924 wait4       S    p2  0:00 bash
2424   690  16   0    952   492             R    p2  0:00 ps l
 923   922   0   0   1332   900 read_chan   S    p3  0:00 bash
2143   923   1   0   7452  6236 do_select   S    p3  0:11 emacs unix0.t
2181  2143   0   0   3776  2836 do_select   S    p4  0:00 xdvi.bin
2190  2181   0   0   3776  2336 pipe_read   S    p4  0:00 gs -sDEVICE=x
[beal@localhost beal]$
```

- Un processus déjà existant peut donner naissance à un autre processus par un apple système **`fork`**.
- Tout processus, sauf le processus zéro, est créé par un appel à **`fork`** à partir de son père.

Les différents états d'un processus

A chaque instant, un processus est dans un certain état.



Commandes internes et commandes externes

Les shells ont des commandes internes et des commandes externes.

- *commande interne* : c'est une commande dont le code est une partie du code de l'interpréteur (`cd`, `pwd`). Le shell courant exécute directement cette commande.
- *commande externe* : c'est une commande dont le code est contenu dans une fichier binaire placé par exemple dans le catalogue `/bin` ou `/usr/local/bin` (`cat`, `date`, ...). Lorsqu'une telle commande est lancée, le shell crée un processus qui exécute cette commande. Il se mets en sommeil en en attendant la terminaison.
- un *script shell* est une fichier écrit en langage shell. On peut l'exécuter et le shell courant lance alors un processus shell qui interprète ce fichier.

```
[beal@localhost beal]$ cat f1
date
[beal@localhost beal]$ f1
mer sep 12 17:06:54 CEST 2001
[beal@localhost beal]$ cat f2
ps ltp2
date
[beal@localhost beal]$ f2
 PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
 690   689  10   0   1384   928 wait4       S   p2  0:00 bash
2483   690  10   0   1384   928 wait4       S   p2  0:00 bash
2484  2483  11   0    952   492             R   p2  0:00 ps ltp2
mer sep 12 17:11:28 CEST 2001
[beal@localhost beal]$  ps ltp2
 PID  PPID PRI  NI   SIZE   RSS WCHAN       STA TTY TIME COMMAND
 690   689  10   0   1384   928 wait4       S   p2  0:00 bash
2488   690  11   0    952   492             R   p2  0:00 ps ltp2
```

Redirection

Les processus communiquent avec l'extérieur par l'intermédiaire de trois fichiers dits *fichiers standards*

- le fichier *entrée standard*,
- le fichier *sortie standard*,
- le fichier *sortie erreur standard*.

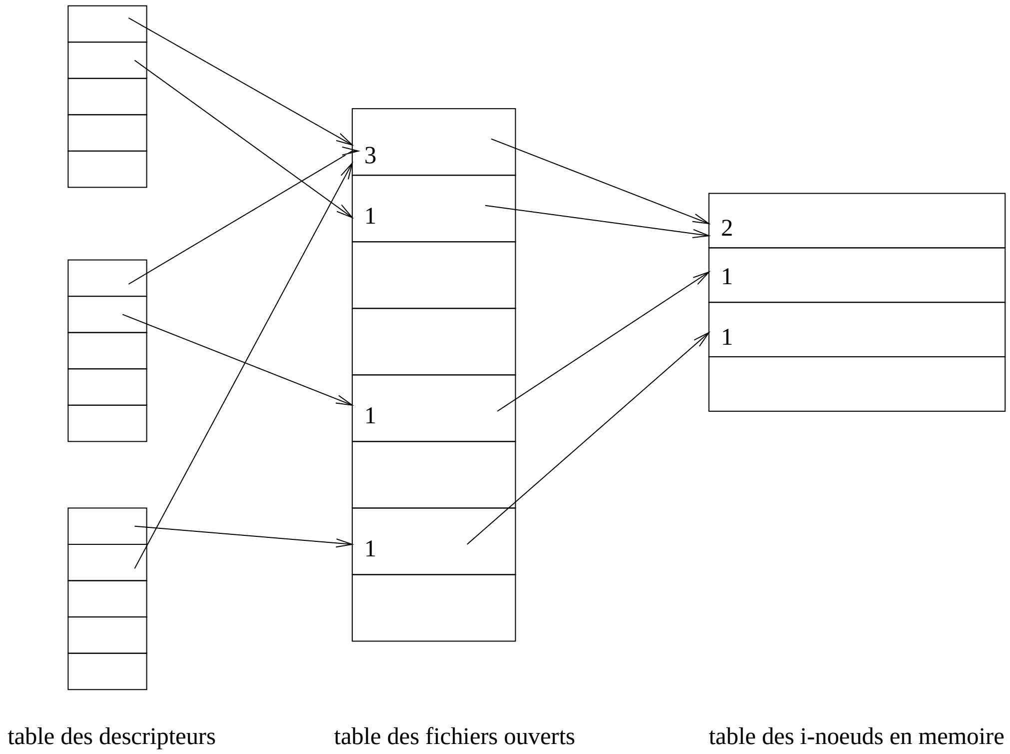
Par défaut, ces trois fichiers sont associés au clavier (pour l'entrée standard) et à l'écran (pour les sorties standards).
On peut rediriger les entrée/sorties standard d'un processus. Cette opération consiste à leur un indiquer un fichier autre que le terminal.

```
[beal@localhost beal]$ date > tutu  --> sortie standard redirigée
[beal@localhost beal]$ cat tutu
mer sep 12 17:30:06 CEST 2001
[beal@localhost beal]$ date >> tutu    --> idem sans ecrasement
[beal@localhost beal]$ cat tutu
mer sep 12 17:35:00 CEST 2001
mer sep 12 17:35:02 CEST 2001
[beal@localhost beal]$ wc -w < tutu
      12                       --> entrée standard redirigée
[beal@localhost beal]$  ls tata
ls: tata: Aucun fichier ou répertoire de ce type.
[beal@localhost beal]$ ls tata 2> tutu
                         --> sortie erreur standard redirigée
[beal@localhost beal]$ cat tutu
ls: tata: Aucun fichier ou répertoire de ce type.
[beal@localhost beal]$ ls tutu tata &> tonton
                         --> les deux sorties en même temps.
[beal@localhost beal]$ cat tonton
ls: tata: Aucun fichier ou répertoire de ce type.
tutu
```

Descripteurs de fichiers

On remarque que sur l'exemple précédent le fichier erreur standard a été désigné par un numéro appelé *descripteur*.

- De façon générale chaque fichier manipulé par un processus Unix est identifié localement dans celui-ci par un descripteur.
- Ce descripteur est un indice dans une table qui est propre à chaque processus.
- Un processus fils hérite à sa naissance d'une copie de la table du père.
- L'entrée standard a comme descripteur 0.
- La sortie standard a comme descripteur 1.
- La sortie erreur standard a comme descripteur 2.
- Une entrée dans la table des descripteurs comprend un pointeur sur une entrée de la table des fichiers ouverts. Chaque entrée de la table des fichiers ouverts comprend elle-même une entrée sur la table des i-nœuds en mémoire.



Enchaînement de processus

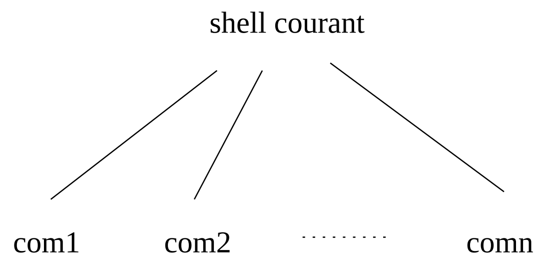
- Il est possible de lancer plusieurs commandes sur une même ligne qui vont s'exécuter séquentiellement
- Il est possible de lancer plusieurs commandes sur une même ligne qui vont s'exécuter *de façon concurrente ou simultanée* en communiquant entre elles par l'intermédiaire d'une zone mémoire appelée tube (pipe). Par le pipe, la sortie standard d'un processus devient l'entrée standard d'un autre.

```
[beal@localhost beal]$ date; who; ps tp2
mer sep 12 18:14:12 CEST 2001
beal       ttyp1    Sep 12 10:05 (:0.0)
carton     ttyp2    Sep 12 10:06 (:0.0)
beal       ttyp3    Sep 12 10:09 (:0.0)
  PID TTY STAT TIME COMMAND
  690  p2 S    0:00 bash
 2757  p2 R    0:00 ps tp2
[beal@localhost beal]$
[beal@localhost beal]$ cat tutu
mer sep 12 17:35:00 CEST 2001
mer sep 12 17:35:02 CEST 2001
[beal@localhost beal]$ cat tutu | wc
      2      12      60
[beal@localhost beal]$
```

Tubes et arbres des processus

- Sous Linux, l'arborescence des processus lors de communications par tube est la suivante.

```
com1 | com2 | ... | comn
```




```
test@niznogood1:~$ sleep 20 | sleep 30 | ps l
 PID  PPID PRI  NI   VSZ  RSS WCHAN  STAT TTY     TIME COMMAND
4763  4761  10   0  2672 1448 wait4  S    pts/1  0:00 -bash
4767  4763   9   0  1848  664 nanosl S    pts/1  0:00 sleep 20
4768  4763  10   0  1848  664 nanosl S    pts/1  0:00 sleep 30
4769  4763  13   0  3000 1124 -      R    pts/1  0:00 ps l
test@niznogood1:~$
```

- Sous Unix System V, l'arborescence est différente.

Lancement de commandes en arrière plan

- Il est possible de lancer une commande en arrière plan, sans que le shell courant en attende la terminaison.
- L'intérêt est de pouvoir continuer à travailler au clavier sans attendre la fin de l'exécution de la commande.

```
[beal@localhost beal]$ xterm &   --> ouverture d'une fenêtre
[1] 3013
[beal@localhost beal]$
```

Les jobs

- Certains shells permettent une gestion aisée des processus créés depuis ces shells. C'est le mécanisme dit de *job control*.
- Il permet aux utilisateurs de ne voir que les processus qu'ils ont lancés sous le shell interactif dans lequel ils travaillent et qui constitue les cadre d'une session de travail.
- Chaque commande correspond à un *job* ou une *tâche*. Du point de vue interne, il peut s'agir d'un groupe de processus.
- Toute tâche est identifiée de façon interne au shell par un petit nombre.
- La gestion se fait ainsi sans solliciter le noyau (`ps` est une commande gourmande). Il est plus aisé de manipuler de petit nombres que des grands (pour les numéros de processus).

Exemple en shell bash.

```
[beal@localhost Unix]$ cat boucle
while :
do :
done
[beal@localhost Unix]$ boucle
                                    --> boucle indéfiniment
```

```
                              --> interruption (voir plus loin)
[beal@localhost Unix]$ boucle &
[2] 3036
[beal@localhost Unix]$ boucle | boucle &
[3] 3039
[beal@localhost Unix]$ jobs -l
[1]   2937 Running                 emacs unix0.tex &
[2]-  3036 Running                 boucle &
[3]+  3038 Running                 boucle
      3039                       | boucle &
[beal@localhost Unix]$  fg %2
boucle                      --> envoie de ctrl-c
[beal@localhost Unix]$ jobs -l
[1]-  2937 Running                 emacs unix0.tex &
[3]+  3038 Running                 boucle
      3039                       | boucle &
[beal@localhost Unix]$ kill %3
[beal@localhost Unix]$ jobs -l
[1]-  2937 Running                 emacs unix0.tex &
[3]+  3038 Terminated              boucle
      3039                       | boucle
[beal@localhost Unix]$
```

Les signaux

- Un processus peut recevoir un signal venant d'un autre processus ou du clavier.
- Le signal reçu est enregistré dans une table contenue dans le bloc de contrôle du processus.
- L'action associée au signal se fait au moment du passage du mode actif en mode noyau au mode actif en mode utilisateur.
- Certains processus ignorent certains signaux. Un processus lancé en arrière plan, un shell, vi ignore les signaux 2 et 3.
  - signal 2, interruption, intr, ctrl-c
  - signal 3, arrêt, quit, ctrl-\
- On peut voir la liste des signaux et leur numéro dans le fichier `/usr/include/asm/signal.h`.

```
#define SIGHUP          1
#define SIGINT          2
#define SIGQUIT         3
#define SIGILL          4
#define SIGTRAP         5
#define SIGABRT         6
#define SIGIOT          6
#define SIGBUS          7
#define SIGFPE          8
#define SIGKILL         9
#define SIGUSR1        10
```

- La commande `kill` permet de lancer un signal donné à un processus donné.

```
[beal@localhost Unix]$$ kill -1 $$
       --> $$ est le numéro du shell courant
       --> tue le shell courant et la fenêtre se referme
```

```
beal@localhost Unix]$ ps
  PID TTY STAT TIME COMMAND
  608  p1 S    0:00 bash
  690  p2 S    0:00 bash
 3240  p2 S    0:02 emacs toutunix.tex unix0.tex
 3242  p2 R    0:00 ps
[beal@localhost Unix]$ kill -9 3240    --> tue de manière forte
[1]+  Killed                    emacs *unix*.tex
[beal@localhost Unix]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGIOT       7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR
[beal@localhost Unix]$
```

- La commande `trap` ordonne au processus courant d'exécuter la commande s'il reçoit le signal indiqué. Les signaux ignorés lors de l'invocation du shell ne peuvent pas être capturés ni reprendre leurs comportements par défaut. Lors de la création d'un processus fils, les signaux capturés reprennent leur comportement par défaut.

```
[beal@localhost Unix]$ ls toto
toto
[beal@localhost Unix]$ trap 'rm toto' 1
[beal@localhost Unix]$ kill -1 $$
rm: détruire ‘toto’? y
[beal@localhost Unix]$ trap 'date' 1
[beal@localhost Unix]$ kill -1 $$
ven sep 28 12:23:40 CEST 2001
[beal@localhost Unix]$ trap '' 1
[beal@localhost Unix]$ kill -1 $$
[beal@localhost Unix]$ trap 1
 --> retour du comportement par défaut.
```

Les langages Shell

Il existe différents dialectes de langages Shell.

- Bourne Shell,
- C-Shell, (csh, tcsh,...)
- Bash (Bourne Again SHell),
- Korn Shell.

Ils interprètent une ligne de commande

- en préparant les arguments (expansion des caractères d'abréviation),
- en effectuant des substitutions de variables,
- en exécutant la commande.

Quelques manipulations de variables en Shell

- Le shell permet de définir et manipuler des variables. Nous décrivons ici la syntaxe du Bourne Shell, Bash et Kornshell.

```
[beal@localhost Cat]$ ls
toto1.c  toto2.c
[beal@localhost Cat]$ ls *1.*
toto1.c
[beal@localhost Cat]$ a = coucou
bash: a: command not found  --> attention aux blancs
[beal@localhost Cat]$ a=coucou
[beal@localhost Cat]$ echo $a
coucou
```

- Certaines variables sont prédéfinies, comme les variables `HOME`, `PATH`, `PS1`, `PS2`.

```
[beal@localhost beal]$ echo $HOME
/home/beal
[beal@localhost beal]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/beal/bin:.
  --> catalogues dans lesquels sont recherchées les commandes
  --> remarquer que la liste contient le catalogue '.'
[beal@localhost beal]$ echo $PS1
[\u@\h \W]\$
[beal@localhost beal]$ echo $PS2
>
[beal@localhost beal]$ PS1=XXX
XXXecho $HOME
/home/beal
XXX
```

Propriétés des langages de commande

- Présence de commandes internes et externes. On rappelle que le code d'une commande interne fait partie du code de l'interpréteur. Par exemple **`cd, pwd, trap, echo`** sont des commandes internes. Ces commandes sont exécutées par le processus shell lui-même. Les commandes externes sont recherchées dans une liste de catalogues prédéfinis (voir la variable `PATH`). Ces commandes sont exécutées par un sous processus.

```
[beal@localhost beal]$ which cal
/usr/bin/cal
[beal@localhost beal]$
```

- On peut écrire des commandes complexes grâce à des structures de contrôle.

```
[beal@localhost beal]$ for i in 1 2 3; do echo coucou; done
coucou
coucou
coucou
[beal@localhost beal]$
```

- On peut écrire des fichiers de commandes appelés scripts shell.

```
[beal@localhost beal]$cat ecritcoucou
echo coucou
[beal@localhost beal]$ecritcoucou
bash: ./ecritcoucou: Permission non accordée.
[beal@localhost beal]$chmod 755 ecritcoucou
[beal@localhost beal]$ecritcoucou
coucou
```

```
[beal@localhost beal]$cat comps
ps f
[beal@localhost beal]$chmod 755 comps
[beal@localhost beal]$comps
  PID TTY STAT TIME COMMAND
  688  p2 S    0:00 bash
  749  p2 S    0:13  \_ emacs toutunix.tex
  800  p3 S    0:00  |   \_ xdvi.bin -name xdvi toutunix.dvi
  908  p2 S    0:00  \_ bash
  909  p2 R    0:00      \_ ps f
[beal@localhost beal]$
 ---> on remarque qu'un nouveau shell a été créé
      pour interpréter comps
[beal@localhost beal]$
```

Caractères spéciaux du shell

- `*` remplace toute chaîne de caractères permettant de compléter un nom de fichier du catalogue courant sauf une chaîne commençant par le caractère `.`.

```
[beal@localhost beal]$ ls .*
  --> indiquer ce qui se passe
[beal@localhost beal]$
```

- `?` remplace un caractère quelconque à l'exception du `.` en début de chaîne.
- `[0-9]`, `[a-z]`, `[a-zA-Z]`, désigne un caractère quelconque des intervalles entre les crochets.
- caractères spéciaux liés au lancement d'une commande `;`, `>>`, `2>`, `<`, `>`, `&`, `|`, `()`, `{`, `}`, `~`.
- caractère d'évaluation d'une variable : `$`, début d'une ligne de commentaire : `#`.
- pour déspécialiser un caractère : `\`

```
[beal@localhost beal]$ echo $0
bash
[beal@localhost beal]$ echo \$0
$0
[beal@localhost beal]$
```

- Certaines chaînes sont entourées de caractères de quotation et reçoivent un traitement particulier.
  - `'abc'` les variables sont non évaluées, les caractères spéciaux sont despécialisés.
  - `''abc''` les variables sont évaluées, les caractères spéciaux sont despécialisés.
  - `` `abc` `` donne le résultat d'une commande.

Les variables du shell

Elles permettent de programmer en shell tout comme les variables de n'importe quel langage de programmation. Certaines variables sont prédéfinies et contiennent des informations sur l'environnement du shell.

Une variable en shell a un nom et une valeur.

- Le nom est une chaîne de caractères quelconque commençant par une lettre et composée de lettres, chiffres ou du caractère `_`.
- La valeur est une chaîne de caractères quelconque.
- La définition se faut par **`nom=valeur`**.
- La valeur est donnée par **`$nom`** ou **`${nom}`**.
- Si **`nom`** n'est pas une variable, la valeur de **`$nom`** est la chaîne de caractères vide.

```
[beal@localhost beal]$ var=coucou
[beal@localhost beal]$ echo "$var > titi"
coucou > titi
[beal@localhost beal]$ echo '$var > titi'
$var > titi
[beal@localhost beal]$ echo `date`
sam sep 22 16:03:20 CEST 2001
[beal@localhost beal]$ x=beal
[beal@localhost beal]$ xy=mpb
[beal@localhost beal]$ echo $xy
mpb
[beal@localhost beal]$ echo ${x}y
bealy
[beal@localhost beal]$ unset x
[beal@localhost beal]$ echo $x

[beal@localhost beal]$
```

La commande interne read

On peut affecter à une ou plusieurs variables des valeurs lues sur l'entrée au moyen de la commande **read**. A la dernière variable est affectée toute la fin de la ligne.

```
[beal@localhost beal]$ read nom adresse
beal paris palerme santiago
[beal@localhost beal]$ echo $nom
beal
[beal@localhost beal]$ echo $adresse
paris palerme santiago
[beal@localhost beal]$
```

Variables prédéfinies

Certaines variables d'environnement sont prédéfinies. On peut les modifier ou en définir d'autres dans les fichiers `.profile` ou `.bash_profile` qui sont exécutés au login. Le fichier `.bashrc` est exécuté à chaque lancement d'un shell bash. Ces variables ont souvent un nom en majuscules.

- Exemple de fichier /etc/bashrc

```
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# For some unknown reason bash refuses to inherit
# PS1 in some circumstances that I can't figure out.
# Putting PS1 here ensures that it gets loaded every time.
PS1="[\u@\h \W]\\$ "

alias which="type -path"
```

- Exemple de fichier .bashrc

```
# .bashrc
# User specific aliases and functions
```

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

alias ls='ls --color=auto'
alias rm='rm -i'
alias mv='mv -i'
alias cp='cp -i'
```

- Exemple de fichier .bash_profile

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:.
BASH_ENV=$HOME/.bashrc
# j'ai rajoute le browser :
BROWSER=/usr/bin/netscape-communicator

export USERNAME BASH_ENV PATH BROWSER
```

- Le mécanisme d'alias (commandes **`alias, unalias`**).

```
[beal@localhost Unix]$ alias ddd=date
[beal@localhost Unix]$ ddd
sam sep 22 16:45:36 CEST 2001
[beal@localhost Unix]$ sh
bash$ ddd
sh: ddd: command not found
bash$ exit
exit        ---> on sort du shell empilé
[beal@localhost Unix]$ ddd
sam sep 22 16:47:25 CEST 2001
```

```
[beal@localhost Unix]$ unalias ddd
[beal@localhost Unix]$ ddd
bash: ddd: command not found
[beal@localhost Unix]$ alias
alias cp='cp -i'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='type -path'
[beal@localhost Unix]$
```

Les commandes set et unset

- La commande **set** permet d'obtenir la liste des variables d'environnement et leur valeur.
- La commande **unset** permet de supprimer une variable d'environnement.
- La commande **env** permet d'obtenir la liste des variables d'environnement qui sont exportables, ainsi que leurs valeurs.
- Lorsqu'un nouveau processus est créé par l'appel à fork, il hérite des variables de l'environnement exportables et de leurs valeurs. On rend une variable exportable, c'est-à-dire visible lors d'empilements de shells, par la commande **export**.

```
[beal@localhost Unix]$ nom=beal
[beal@localhost Unix]$ export nom
[beal@localhost Unix]$ env
BROWSER=/usr/bin/netscape-communicator
HISTSIZE=1000
HOSTNAME=localhost.localdomain
LOGNAME=beal
HISTFILESIZE=1000
TERM=xterm
HOSTTYPE=i386
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/beal/bin:.
```

```
HOME=/home/beal
SHELL=/bin/bash
USER=beal
nom=beal
BASH_ENV=/home/beal/.bashrc
DISPLAY=:0.0
OSTYPE=Linux
```

- Quelques exemples de manipulation de variables

```
[beal@localhost beal]$ PS1='$ '
$ date
sam sep 22 17:02:03 CEST 2001
$ bash
[beal@localhost beal]$ --> expliquer pourquoi on a ici
                           ce prompt (voir man bash)

[beal@localhost Unix]$ a=coucou
[beal@localhost Unix]$ export a
[beal@localhost Unix]$ bash
[beal@localhost Unix]$ echo $a
coucou
[beal@localhost Unix]$ a=couic
[beal@localhost Unix]$ echo $a
couic
[beal@localhost Unix]$ exit
exit    --> on sort du shell empilé
[beal@localhost Unix]$ echo $a
coucou

[beal@localhost Unix]$ b=`pwd`
[beal@localhost Unix]$ echo $b
/home/beal/Unix
[beal@localhost Unix]$
```

Exécution d'un fichier de commandes

Considérons le petit fichier de commandes suivant

```
[beal@localhost Unix]$ cat com1
a=coucou
echo $a
[beal@localhost Unix]$
```

Il existe différentes méthodes pour l'exécuter.

- `bash com1`

  ```
  [beal@localhost Unix]$ bash com1
  coucou
  [beal@localhost Unix]$
  ```

- `. com1` (ou `source com1`) redirige l'entrée standard sur `com1`.

  ```
  [beal@localhost Unix]$ . com1
  coucou
  [beal@localhost Unix]$
  ```

- `com1`, le shell crée un nouveau processus qui lit les commandes de com1.

  ```
  beal@localhost Unix]$ chmod 500 com1
  [beal@localhost Unix]$ com1
  coucou
  [beal@localhost Unix]$
  ```

- `exec com1`. Le shell courant est recouvert par un shell qui lit le fichier `com1` et meurt ensuite. Le fichier `com1` doit être accessible en lecture est exécution.

  ```
  [beal@localhost Unix]$ exec com1
  --> la fenêtre xterm se referme
  ```

Variables maintenues par le shell

Les variables d'environnement sont des variables qui sont en général fixes. Le shell maintient aussi à jour certaines variables qui sont utiles pour la programmation.

- `$` a pour valeur le numéro du processus shell en cours.
- `!` a pour valeur le numéro du dernier processus lancé en arrière plan.
- `?` a pour valeur le code de retour de la dernière commande exécutée (0 pour normal, différent de 0 sinon).

```
[beal@localhost Unix]$ sleep 100 &
[1] 1008
[beal@localhost Unix]$ ps
  PID TTY STAT TIME COMMAND
  943  p5 S    0:00 bash
 1008  p5 S    0:00 sleep 100
 1009  p5 R    0:00 ps
[beal@localhost Unix]$ echo $$
943
[beal@localhost Unix]$ echo $!
1008
[beal@localhost Unix]$ ls titi
ls: titi: Aucun fichier ou répertoire de ce type.
[beal@localhost Unix]$ echo $?
1
[beal@localhost Unix]$
```

Variables de position et paramètres

Tout processus shell maintient une liste de chaînes de caractères que l'on peut connaître par la valeur des variables `*,#,1,2,3,...`

- `#` donne le nombre de chaînes présente dans la liste.
- `i` donne la ième chaîne de la liste.
- `*` donne la liste.
- `0` donne la variable de numéro 0.
  - Si c'est un login shell ou un shell utilisateur, `0` a pour valeur le nom du shell et `*` est la chaîne vide.
  - Si c'est un shell qui exécute une commande, `0` a pour valeur le nom de cette commande et `*` est la liste des arguments de la commande.

```
[beal@localhost Unix]$ cat com2
echo "commande : $0"
echo "nombre d'arguments : $#"
echo "liste des arguments : $*"
echo "2eme argument : $2"
[beal@localhost Unix]$ com2 paris palerme santiago
commande : ./com2
nombre d'arguments : 3
liste des arguments : paris palerme santiago
2eme argument : palerme
[beal@localhost Unix]$
```

On peut affecter une valeur à la liste des variables de position par `set`.

```
[beal@localhost Unix]$ set world trade center
[beal@localhost Unix]$ echo $3
center
```

Exemple de fichier de commande

On écrit un script shell qui permet d'échanger le contenu de deux fichiers passés en argument. On utilise un fichier temporaire pris dans le catalogue /tmp.

```
#!/bin/bash
#echange le contenu de deux fichiers
ln $1 /tmp/echange$$
ln -f $2 $1
ln -f /tmp/echange$$ $2
rm  /tmp/echange$$
```

```
[beal@localhost Unix]$ echo escort > ford
[beal@localhost Unix]$ echo 307 > peugeot
[beal@localhost Unix]$ echange ford peugeot
[beal@localhost Unix]$ cat ford peugeot
307
escort
```

Structures de contrôle

En shell, on ne peut tester que le code de retour d'une commande.

- A la terminaison d'une commande externe, le code de retour du processus fils est affecté par le shell à la variable de nom `?`.
- Cette variable est également positionnée à la terminaison d'une commande interne (0 → vrai, différent de 0 → faux).
- Pour une commande composée, c'est le code de retour de la dernière commande qui est transmis.

```
[beal@localhost Unix]$ ls toto
ls: toto: Aucun fichier ou répertoire de ce type.
[beal@localhost Unix]$ echo $?
1
[beal@localhost Unix]$ if (who | grep "^beal")  > /dev/null
> then write beal < peugeot
> else mail beal < peugeot
> fi
write: beal is logged in more than once; writing to ttyp2
--> ecrit escort sur une autre fenêtre
[beal@localhost Unix]$
```

La commande test

Il s'agit d'une commande externe qui teste si une expression est vraie ou fausse. Si l'expression est vraie, le code de retour de la commande est 0, sinon le code de retour est différent de 0.

- Usage : `test expression` ou `[ expression ]`
- Tests sur les chaînes de caractères :
  - `test -z chaine`, vrai si la chaine est nulle.
  - `test -n chaine`, vrai si la chaine est non nulle.
  - `test chaine1 = chaine2`, vrai si chaine1 est égale à chaine2.
  - `test chaine1 >= chaine2`, vrai si chaine1 est supérieure ou égale à chaine2 pour l'ordre lexicographique.

```
[beal@localhost Unix]$ if test ‘pwd‘ = $HOME
> then echo a la maison
> else echo pas a la maison
> fi
pas a la maison
[beal@localhost Unix]$ if [ ‘pwd‘ = $HOME ]
> then echo a la maison
> else echo pas a la maison
> fi
>
pas a la maison
[beal@localhost Unix]$
```

- Tests sur les chaînes numériques entières : une chaîne numérique entière est une suite de chiffres précédée éventuellement de − ou +.
  - `test chaine1 -eq chaine2`, vrai si la valeur numérique correspondant à la chaîne1 est égale à la valeur numérique

correspondant à la chaîne2. Autres opérateurs **-eq, -ne, -lt -le, -gt, -ge**.

```
[beal@localhost Unix]$ ls toto
ls: toto: Aucun fichier ou répertoire de ce type.
[beal@localhost Unix]$ if test $? -eq 0
> then echo toto existe
> else echo toto n\'existe pas
> fi
toto n'existe pas
[beal@localhost Unix]$
```

- Tests sur les fichiers
  - **test -f reference**, vrai si la référence est un fichier ordinaire. Autres tests **-p, -d, -c, -b**.
- Tests sur les droits d'accès
  - **test -r reference**, teste le droit en lecture. Autres tests **-x, -w**.
- Tests sur la taille d'un fichier
  - **test -s reference**, vrai si la référence est un fichier de taille non nulle.
- Tests de rattachement d'un descripteur à un terminal
  - **test -t n**, vrai si le descripteur n est attaché à un terminal.
- Opération sur les expressions
  - **test -f toto -a -r toto** ou **test \(-f toto -a -r toto \)** pour le et.
  - Autres opérations (ou , et, non) **-o, -a, !**.

```
beal@localhost Unix]$ if [ ! -d /tmp/tmp1/tmp2 ]
> then mkdir -p /tmp/tmp1/tmp2
> fi
```

Conditionnelle if

Usage :

```
if commande1
then commande2
else commande3
fi
```

Si **`if`**, **`then`**, **`else`** ne sont pas écrits en début de ligne, ils doivent être précédés de `;`.

- Dans l'exemple ci-dessous, on écrit un script shell qui prend en argument une référence, l'affiche par **`cat`** si c'est un fichier ordinaire et par **`ls`** si c'est un catalogue.

```
[beal@localhost Unix]$ cat affiche
if [ $# -ne 1 ]
then echo syntaxe : $0 nom de fichier ; exit 1
fi
if test -f $1
then cat $1
else if test -d $1
     then ls $1
     else echo $1 est un fichier special
     echo ou n\'existe pas
     fi
fi
[beal@localhost Unix]$ affiche affiche
--> Que se passe-t-il ici ?
```

Conditionnelle case

Usage :

```
case chaine in
motif1) commande1;;
motif2) commande2;;
...
esac
```

- Exemple d'écriture d'une commande de compilation qui appelle "le bon" compilateur.

```
case $# in
0) echo -n fichier à compléter
   read reference
   set $reference
   ;;
esac
case $1 in
*.java) javac $1 ;;
*.c)    gcc $1 ;;
*)      echo compilation de $1 impossible ;;
esac
```

Itération bornée

Usage :

```
for variable in chaine1 chaine2 ...
do commande
done
```

```
[beal@localhost Cat]$ for i in * ; do echo $i ; done
toto1.c
toto2.c
[beal@localhost Cat]$ for i in * ; do wc -c $i; done
      7 toto1.c
      7 toto2.c
[beal@localhost Cat]$
```

Autres itérations

```
[beal@localhost Unix]$ cat com2
question='repondre oui ou non'
echo $question
read reponse
while [ "$reponse" != oui  -a  "$reponse" != non ]
do echo $question
read reponse
done
[beal@localhost Unix]$ com2
repondre oui ou non
yes
repondre oui ou non
oui
[beal@localhost Unix]$ while :; do echo coucou; done
coucou
coucou
coucou
coucou
--> boucle infinie interrompue par ctrl-c.
```

Les filtres

Un filtre est une commande qui lit l'entrée standard et écrit les données lues et éventuellement modifiées sur la sortie standard.

- Les commandes **head** et **tail** permettent de selectionner une portion de texte.

```
[beal@localhost Unix]$ cat wtc
world
trade
tour1
tour2
center
[beal@localhost Unix]$ head -4 < wtc | tail +3
tour1
tour2
[beal@localhost Unix]$
```

- La commande **tr** permet de remplacer un caractère par un autre.

```
[beal@localhost Unix]$ tr [a-z] [A-Z] < wtc
WORLD
TRADE
TOUR1
TOUR2
CENTER
[beal@localhost Unix]$
```

- Le filtre **sed** reçoit en entrée chaque ligne d'un fichier, lui fait subir un traitement (des substitutions par exemple) et envoie la ligne modifiée sur la sortie standard.

- Exemple simple d'utilisation de **sed**.

```
[beal@localhost Cat]$ cat toto
coucou coucou
couic couic
[beal@localhost Cat]$ sed s/co/do/ < toto > titi
[beal@localhost Cat]$ cat titi
```

```
doucou coucou
douic couic
[beal@localhost Cat]$ sed s/co/do/g < toto > titi
[beal@localhost Cat]$ cat titi
doudou doudou
douic douic
[beal@localhost Cat]$ sed '2,$s/co/do/g' < toto > titi
[beal@localhost Cat]$ cat titi
coucou coucou
douic douic
--> les transformations sont effectuées à partir
--> de la deuxième ligne. Remarquer les quotations
--> car $ est un caractère spécial du shell.
[beal@localhost Cat]$
```

- Pour une utilisation plus élaborée, on peut former des expressions régulières de sed.
  - `.` désigne tout caractère excepté newline.
  - `*` signifie une répétition du caractère précédent un nombre arbitraire (éventuellement nul) de fois.
  - `[abc]` désigne un caractère quelconque `a,b` ou `c`.
  - `^` placé en début d'une expression indique que celle-ci doit être recherchée en début de ligne.
  - `$` placé en fin d'expression indique que celle-ci doit être recherchée en fin de ligne.
  - \ déspécialise le caractère qui suit (comme en shell).
- Un exemple plus élaboré (et quelque peu illisible). On désire écrire une commande filtre `com` qui ne conserve que les deuxièmes colonnes de l'entrée standard.

```
beal@localhost Unix]$ who
beal     ttyp1    Sep 23 15:31 (:0.0)
beal     ttyp3    Sep 23 15:46 (:0.0)
```

```
beal     ttyp4    Sep 23 17:01 (:0.0)
[beal@localhost Unix]$ beal@localhost Unix]$ cat com
#!/bin/bash
# filtre qui garde les deuxièmes colonnes du texte

sed 's/^\([^ ]*[ ]*\)\([^ ]*\)\(.*\)/\2/'
         |    |
          ----
           |
   (tout sauf un blanc)*

[beal@localhost Unix]$ who | com
ttyp1
ttyp3
ttyp4
[beal@localhost Unix]$
```

Exercice Écrire une commande **`com.sh`** qui contient un script shell prenant deux arguments et transforme tous les fichiers du répertoire courant suffixés par le premier argument en des fichiers suffixés par le deuxième argument.

```
[beal@localhost Unix]$ ls
toto tutu.h titi.h
[beal@localhost Unix]$ com.sh h c
[beal@localhost Unix]$ ls
toto tutu.c titi.c
```