Système

Polytech Paris-Sud

Cycle ingénieur de la filière étudiant

Louis Mandel

Université Paris-Sud 11

Louis.Mandel@lri.fr

année 2012/2013

Références

Livres :

- *Systèmes d'exploitation*
  Andrew Tanenbaum. Pearson Education.
- *Unix, programmation et communication*
  Jean-Marie Rifflet et Jean-Baptiste Yunès. Dunod.
- *Programmation Linux 2.0*
  Rémy Card, Eric Dumas et Franck Mével. Eyrolles.
- *Advanced Programming in the UNIX Environment*
  W. Richard Stevens et Stephen A. Rago. Addison-Wesley.

Cours :

- *Principes et Programmation des Systèmes d'exploitation*
  Didier Rémy. `http://pauillac.inria.fr/~remy/poly/system/`
- *Programmation des systèmes*
  Philippe Marquet. `http://www2.lifl.fr/~marquet/cnl/pds/`

Organisation du cours

- Page web du cours :

  `http://www.lri.fr/~mandel/systeme/`
- Ce qui est vu en cours est supposé connu en TD et TP
  - On ne refait pas le cours en TD pour les absents
- Les questions sont bienvenues
  - pendant / après le cours, mail ...
  - commentaires sur le cours
- Implication personnelle des étudiants
  - Programmez !

Système

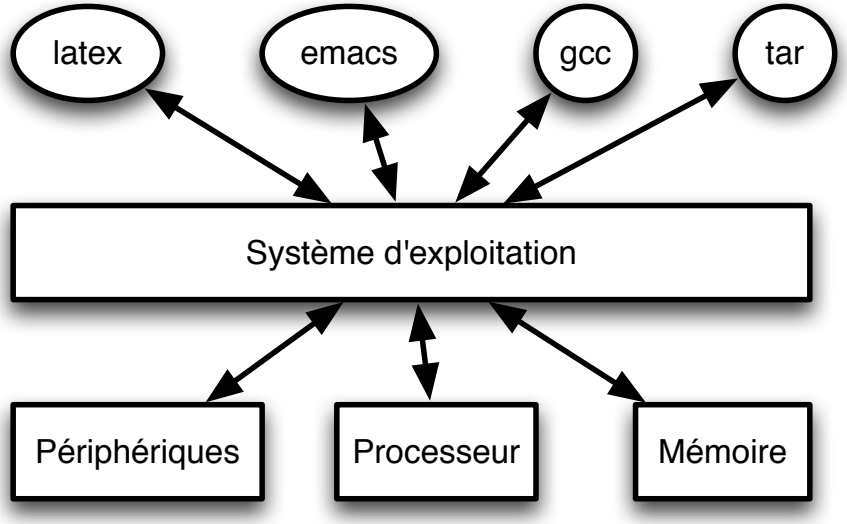
Généralités

Historique

Il était une fois . . .

- `http://www.computerhistory.org/timeline`
- `http://en.wikipedia.org/wiki/Operating_system`
- `...`

Rôles du système d'exploitation : Machine Virtuelle



- Abstraction du matériel
  - Protection : les tâches matérielles sont déléguées au système.
  - Portabilité : un même système peut être disponible sur plusieurs architectures.
  - Simplicité : pour le programmeur, c'est la même chose de lire un fichier sur le disque dur ou sur un CD-ROM.

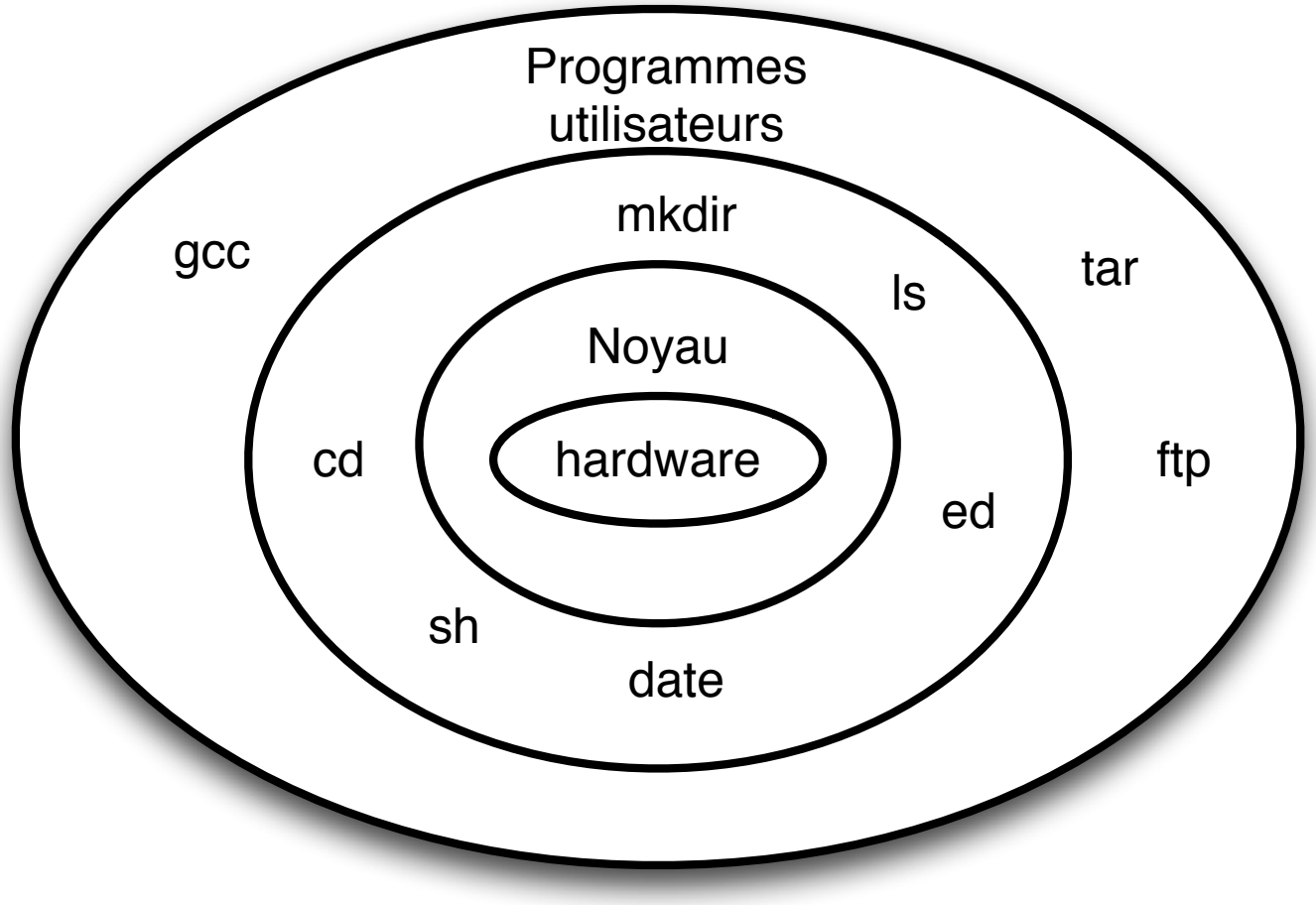
Rôles du système d'exploitation : Gestion des ressources

- Partage : donner l'illusion que
  - chaque programme tourne seul sur la machine
  - chaque utilisateur est seul sur la machine
- Équité :
  - accès au processeur
  - gestion des quotas
- Protection : garantir que
  - un utilisateur ne peut pas modifier les données d'un autre
  - un processus ne peut pas modifier la mémoire d'un autre

Rôles du système d'exploitation : Centre de communication

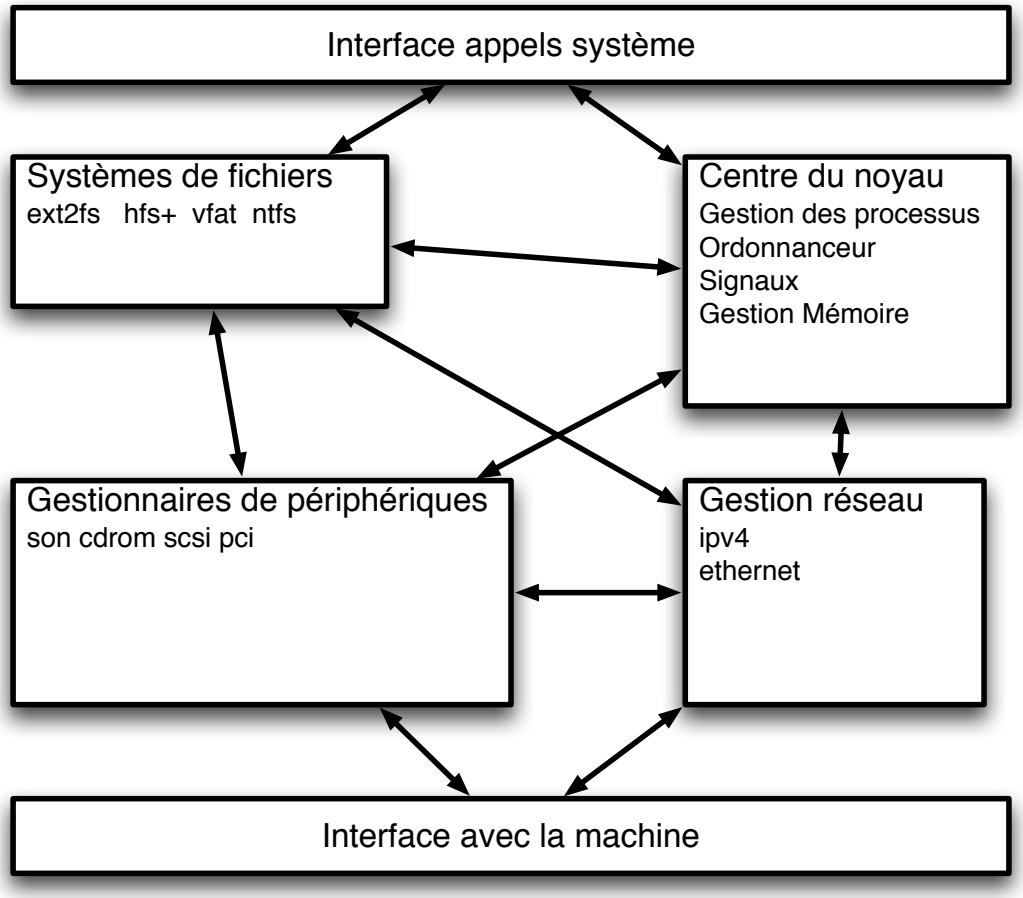
- Communication entre le matériel et le logiciel
  - interruptions
- Communication entre les processus
  - signaux
  - tubes (*pipes*)
  - *sockets*
  - IPC

Structure du système



- Deux modes d'exécution : noyau et utilisateur
  - changement de mode : appel système

Structure du système



Système

Bases d'UNIX

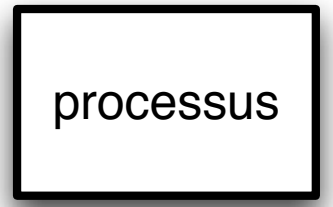
Références

- *ABC d'Unix*
  Christian Queinnec.
  `http://pagesperso-systeme.lip6.fr/Christian.Queinnec/Books/ABCdUNIX/uunix-toc.html`

Les objets d'UNIX

- Les fichiers (*file*)
  - ▷ Suites ordonnées d'octets.

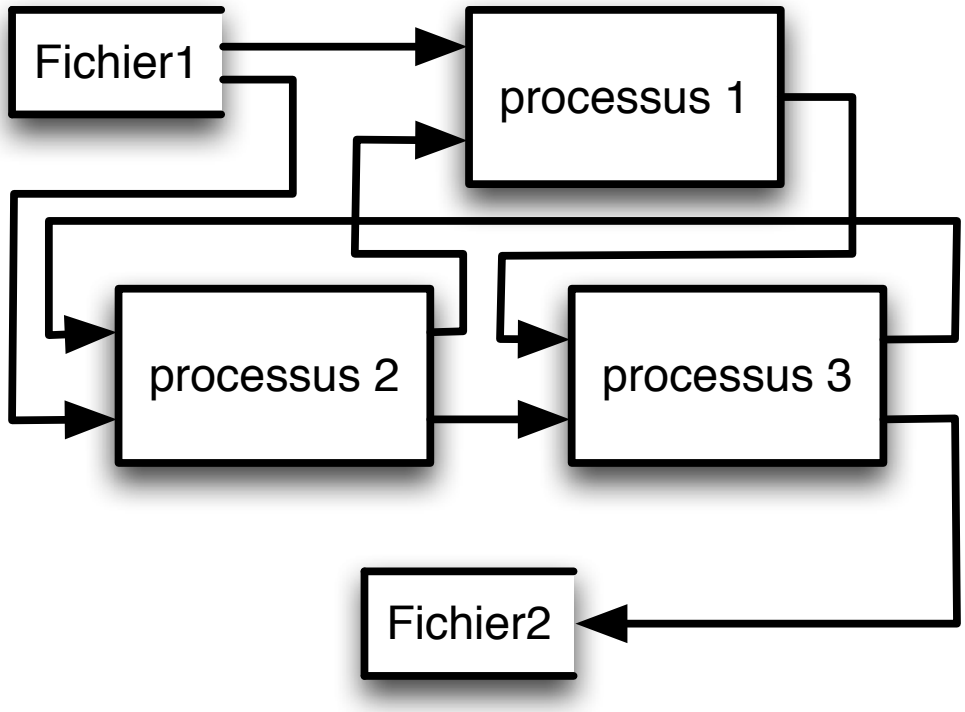
- Les processus (*processes*)
  - ▷ Programmes en cours d'exécution.



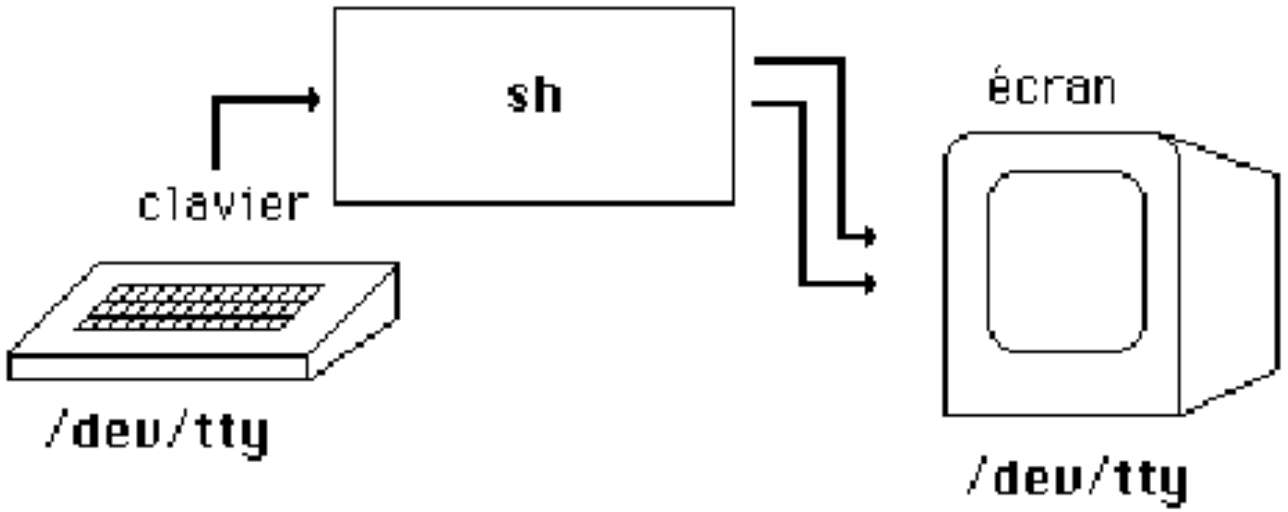
- Les flux (*streams*)
  - ▷ Suites d'octets produites par un fichier ou une tâche en destination d'un fichier ou d'une tâche.

Tout agencement particulier de tâches, flux ou fichiers peut être mis en oeuvre par UNIX.

Exemple :



Terminal



Interprète de commandes ou shell

- Interface entre le système et l'utilisateur
- Le shell permet de :
  - naviguer dans le système de fichiers
  - lancer des processus (et de les combiner)
  - interagir avec le système
  - automatiser des tâches
- Il existe plusieurs shells :
  - `sh` (Bourne shell) : shell historique
  - `ksh` (Korn shell) : shell normalisé POSIX
  - `bash` (GNU Bourne again shell) : extension de `ksh`
  - `csh` (C-shell) : shell BSD
  - `tcsh` (Toronto C-shell) : extension de `csh`

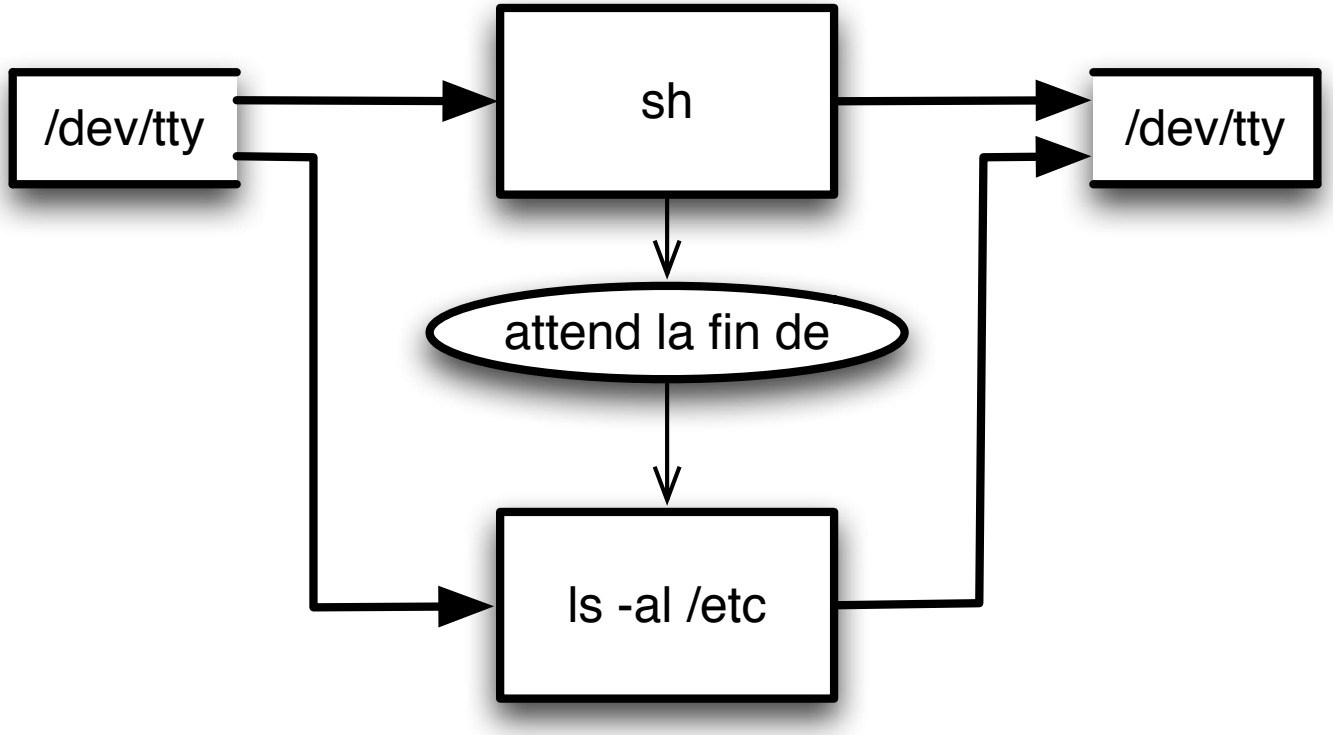
Naviguer dans le système de fichiers

Le terminal peut servir d'alternative à Finder, Explorer ...

- Le shell est toujours associé à un répertoire courant : `pwd`
- Il est possible d'afficher le contenu d'un répertoire : `ls`
- Il est possible de changer de répertoire courant : `cd`

Exécution d'une commande

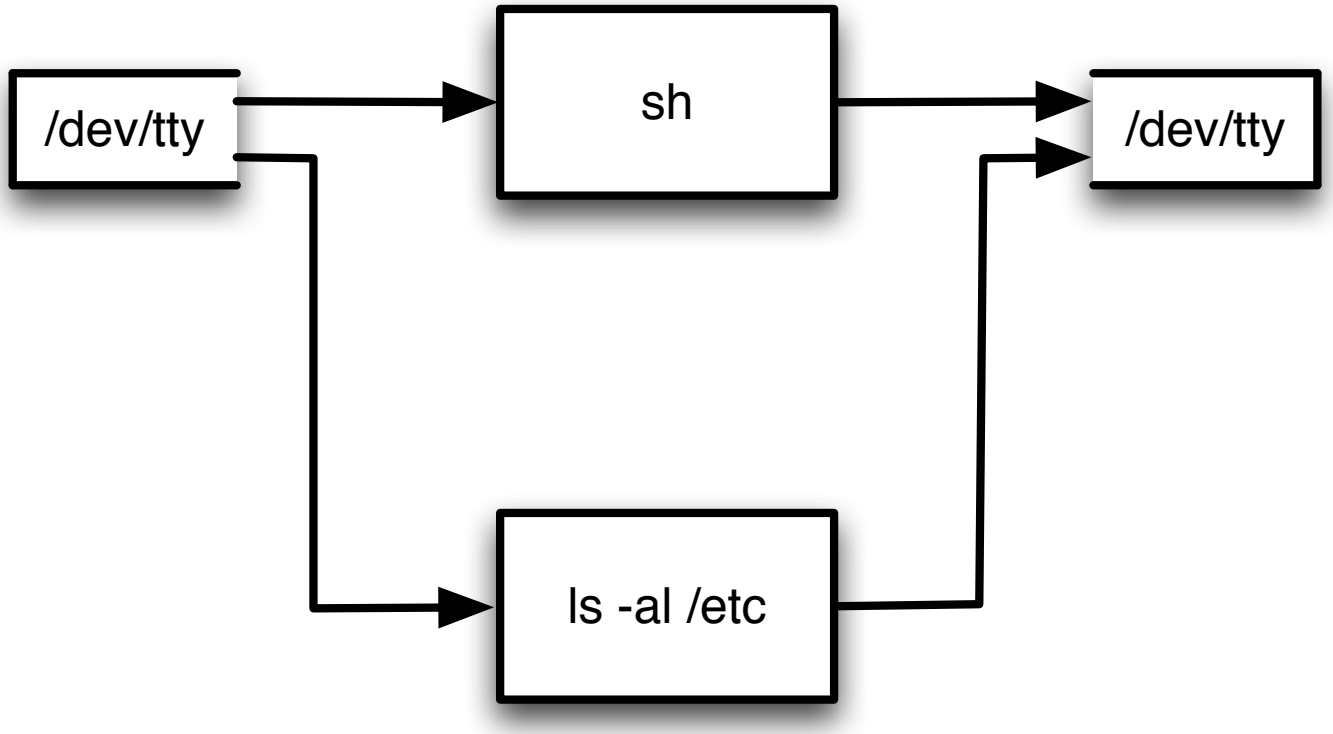
```
--> ls -al /etc
```



- L'exécution de chaque commande retourne un statut : (0 pour succès, et un entier différent de 0 sinon)
- Ce statut peut être affiché par : `echo $?`

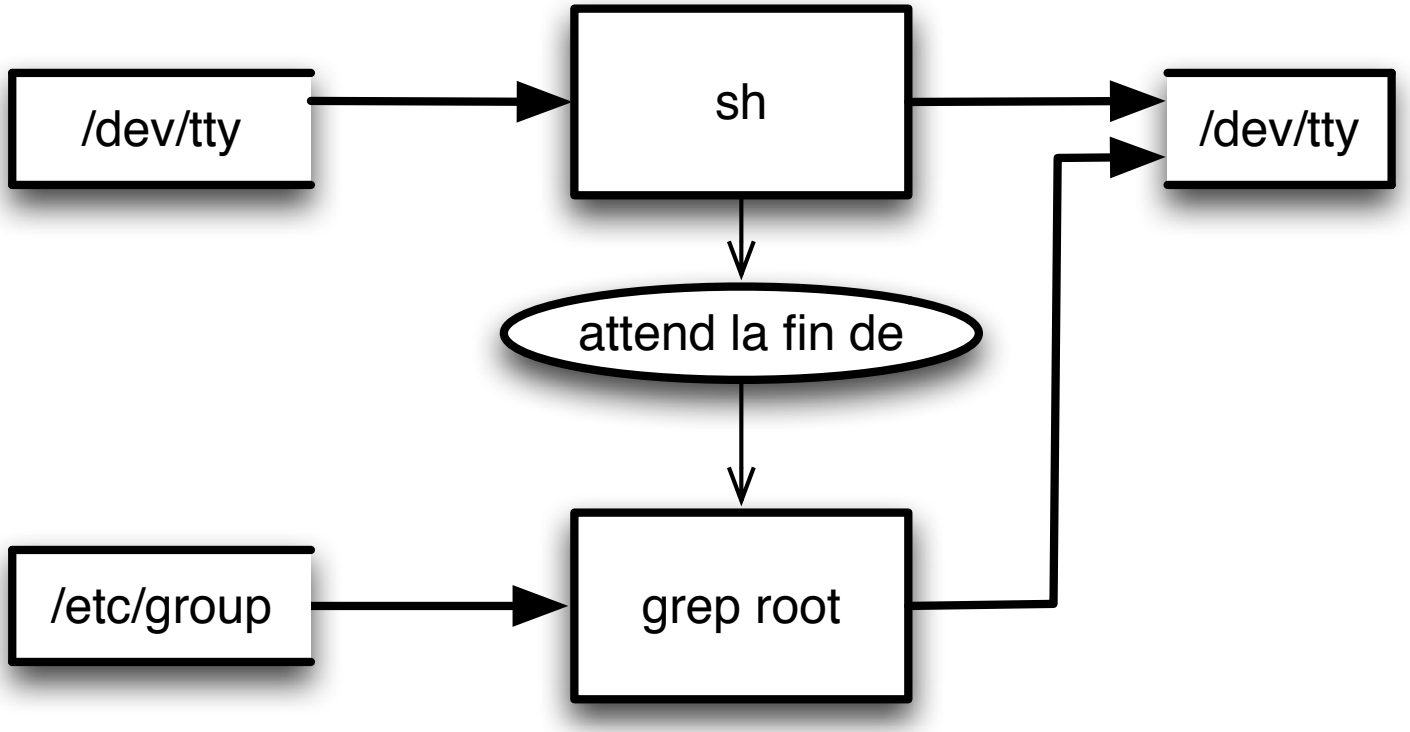
Exécution d'une commande en tâche de fond

```
--> ls -al /etc &
```



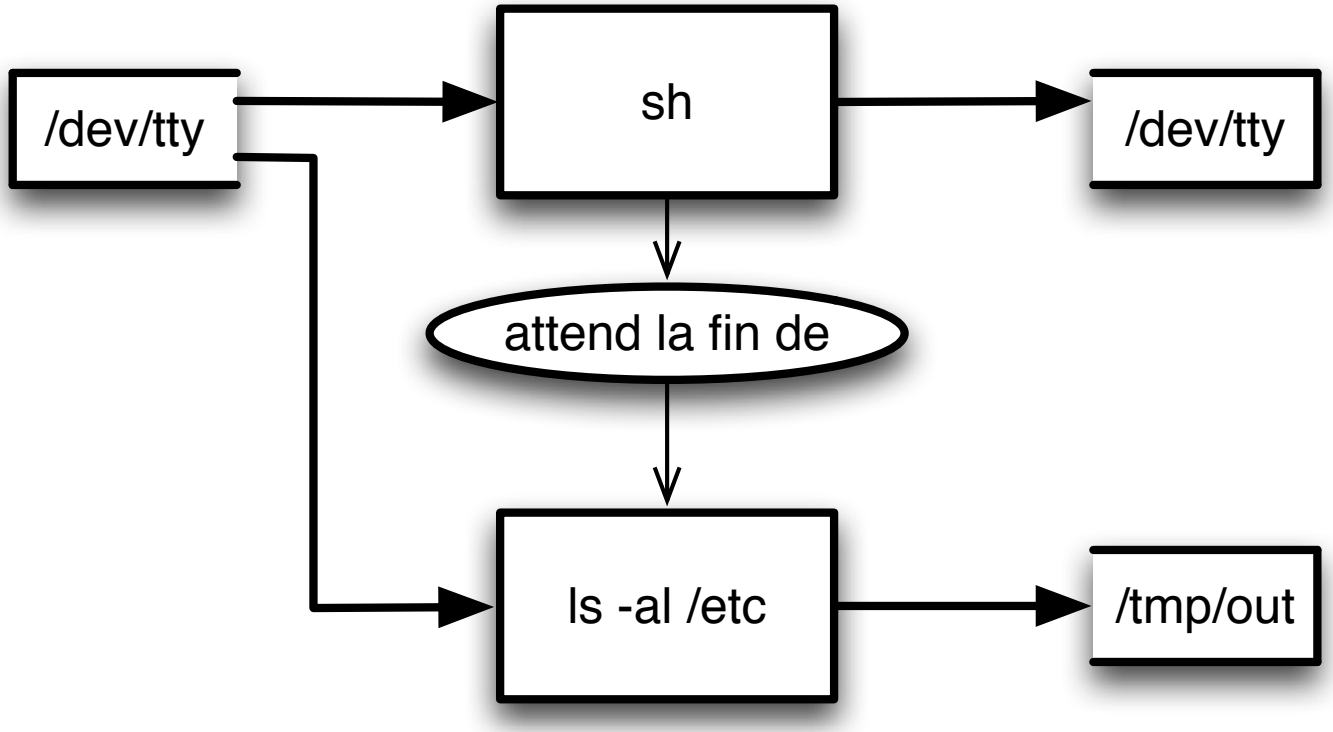
Redirection de l'entrée

```
--> grep root < /etc/group
```



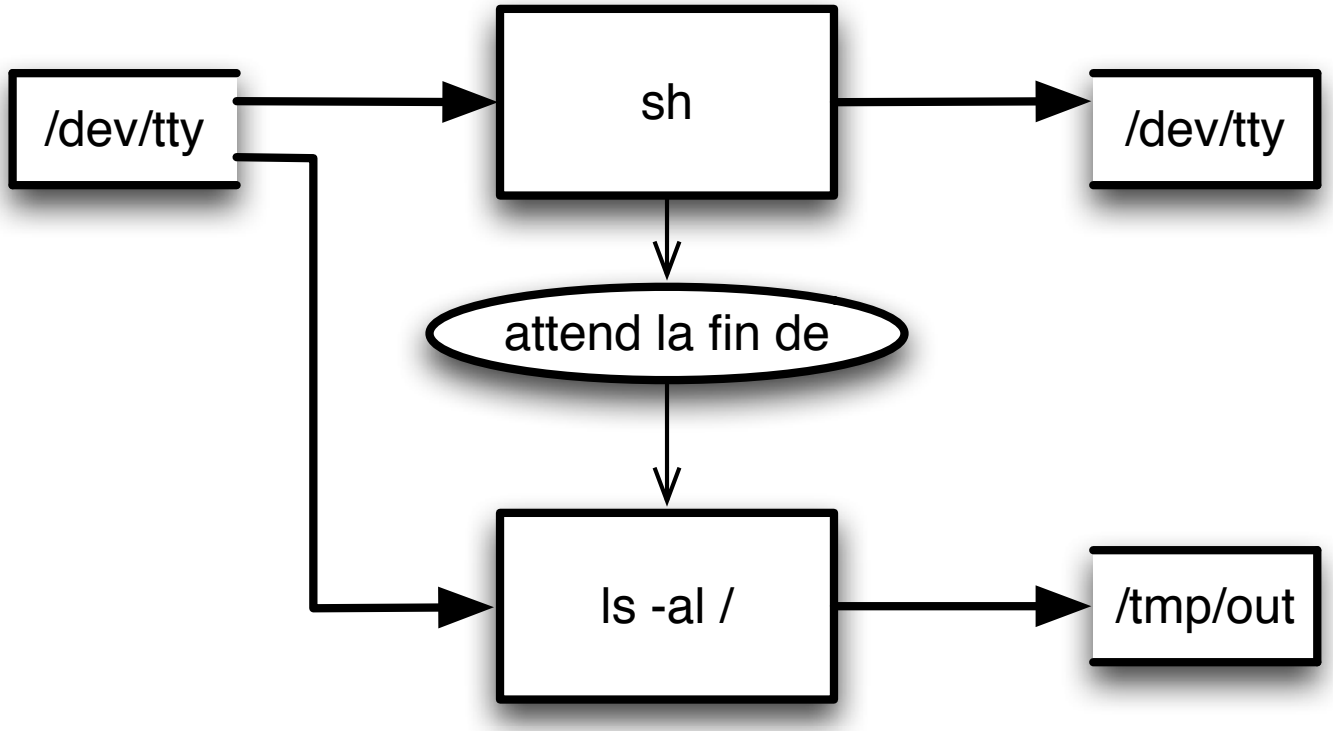
Redirection de la sortie

```
--> ls -al /etc > /tmp/out
```



Redirection de la sortie

```
--> ls -al / >> /tmp/out
```



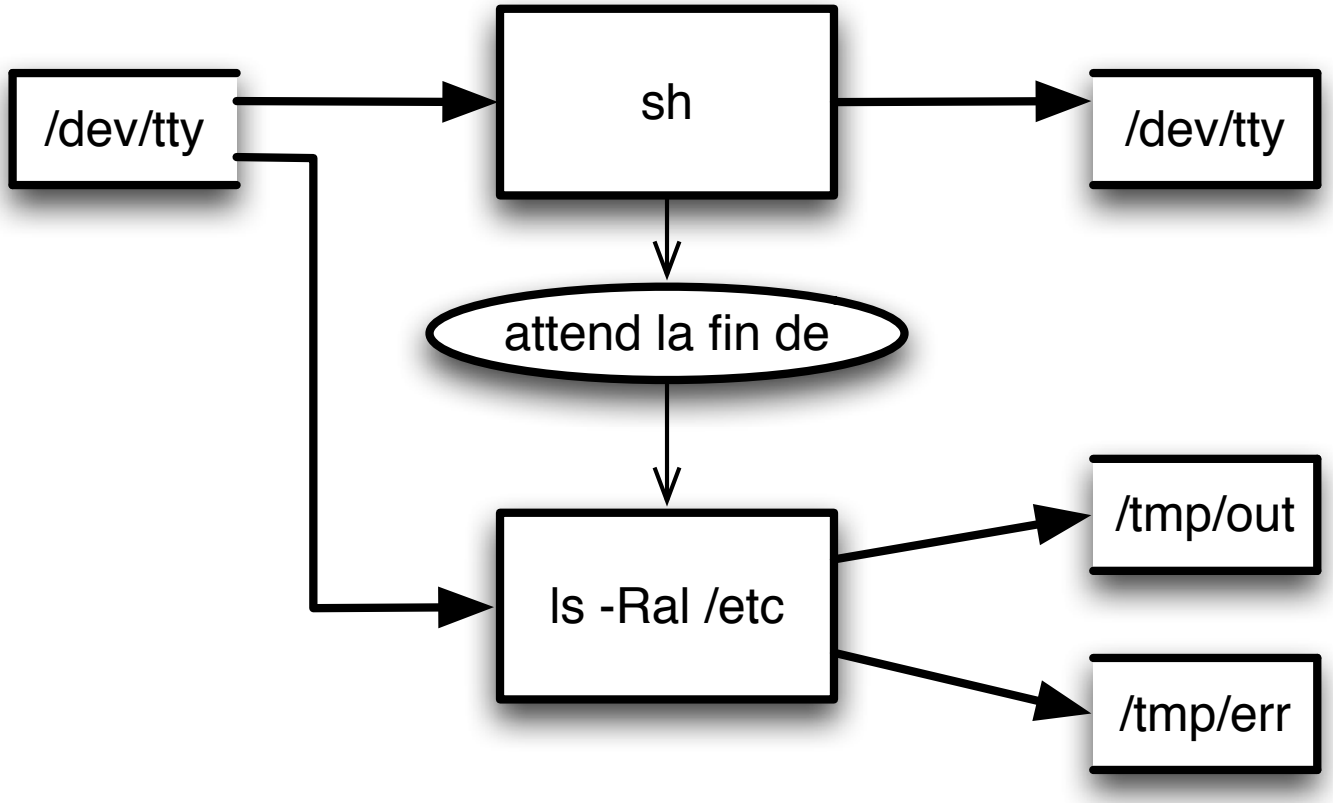
- La sortie est ajoutée à la fin du fichier

Les flux standards

- Un processus peut disposer de plusieurs flux.
- Les flux sont numérotés à partir de zéro.
- Les trois premiers sont usuellement nommés :
  - `stdin` (pour standard input) et porte le numéro zéro,
  - `stdout` (pour standard output) et porte le numéro un,
  - `stderr` (pour standard error output) et porte le numéro deux.

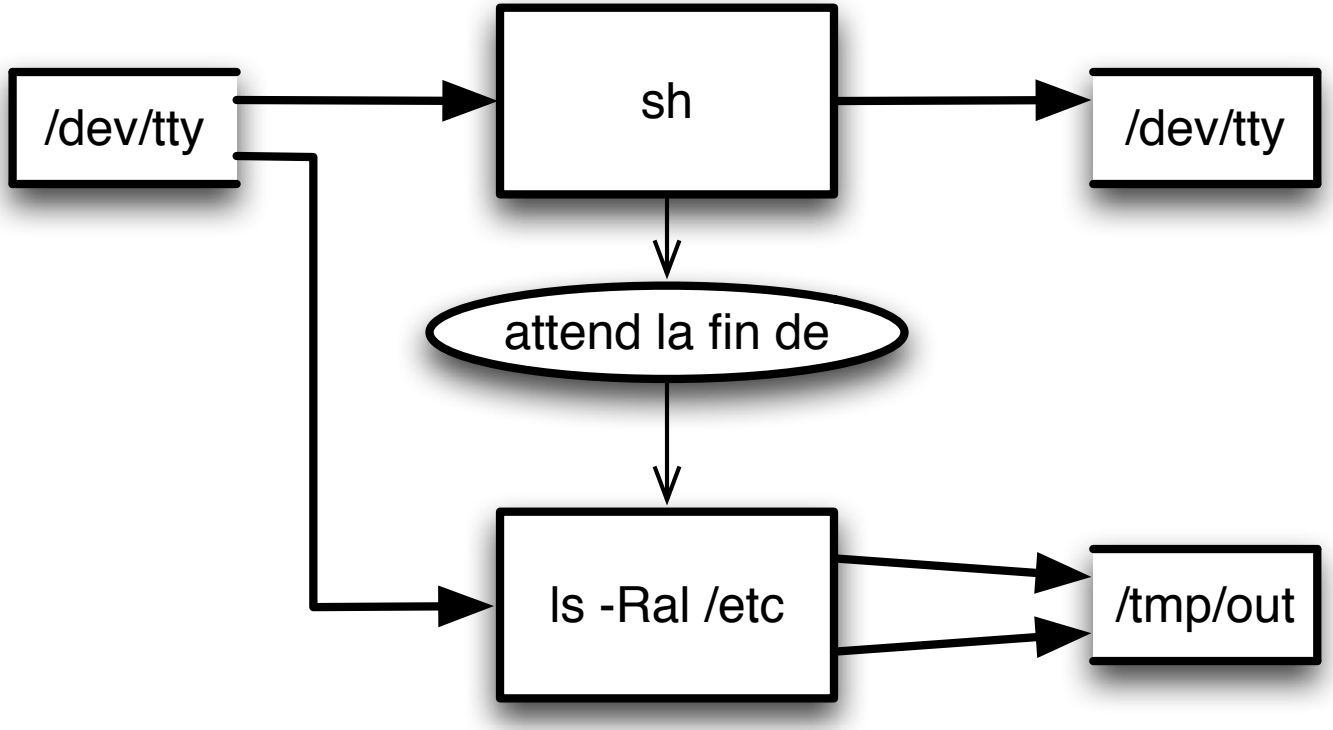
Redirection de flux

```
--> ls -Ral /etc > /tmp/out 2> /tmp/err
```



Redirection de flux

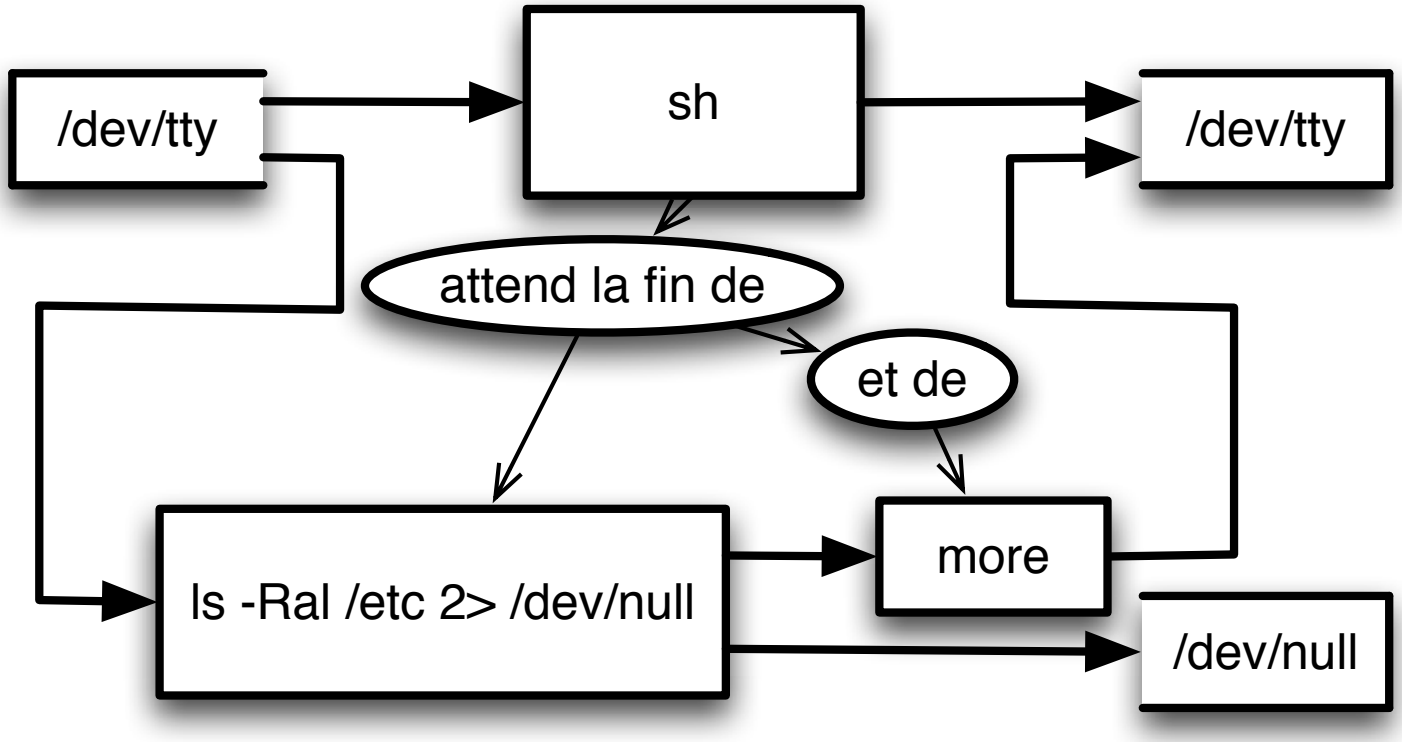
```
--> ls -Ral /etc > /tmp/out 2>&1
```



- Remarque : la commande `ls -Ral /etc 2>&1 > /tmp/out` a un comportement différent.

Exécution en cascade : Pipe

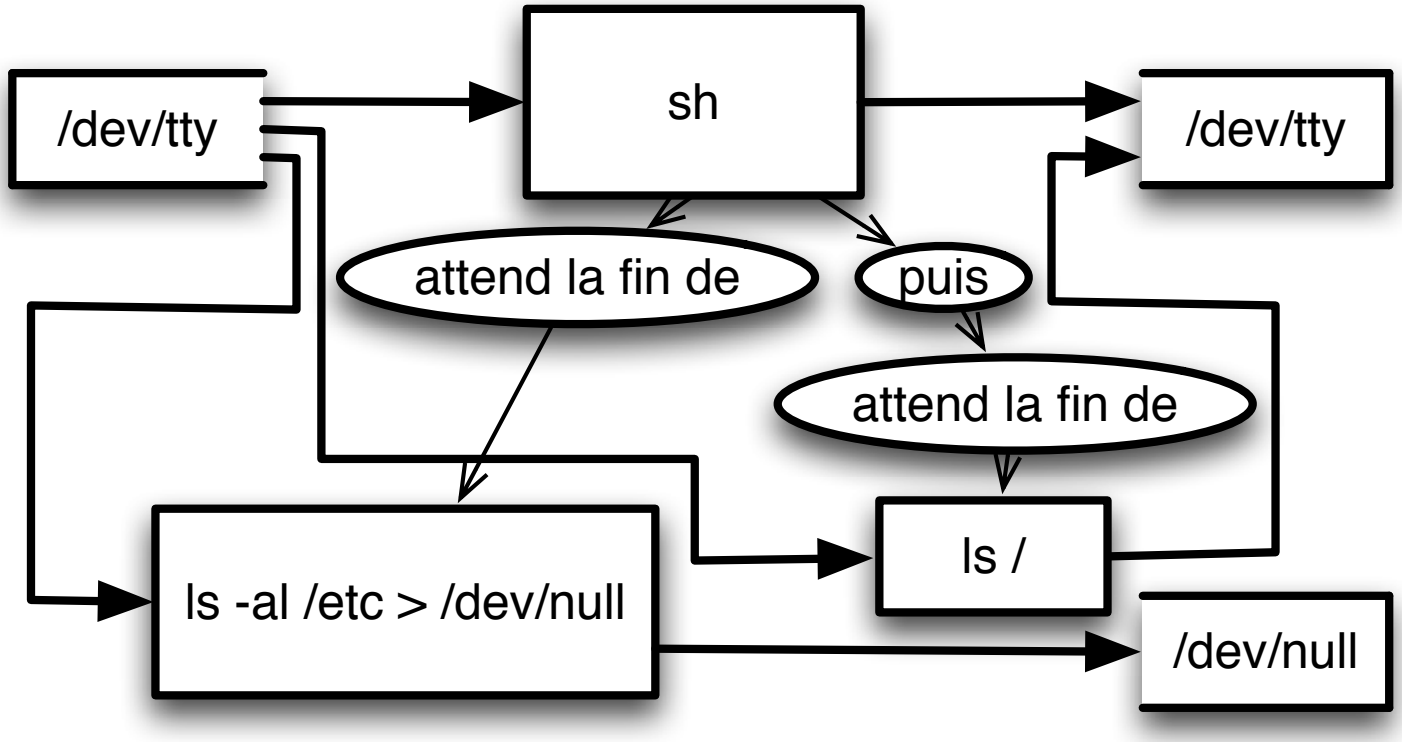
```
--> ls -Ral /etc 2> /dev/null | more
```



- Les commandes `ls` et `more` sont exécutées en parallèle

Exécution en séquence

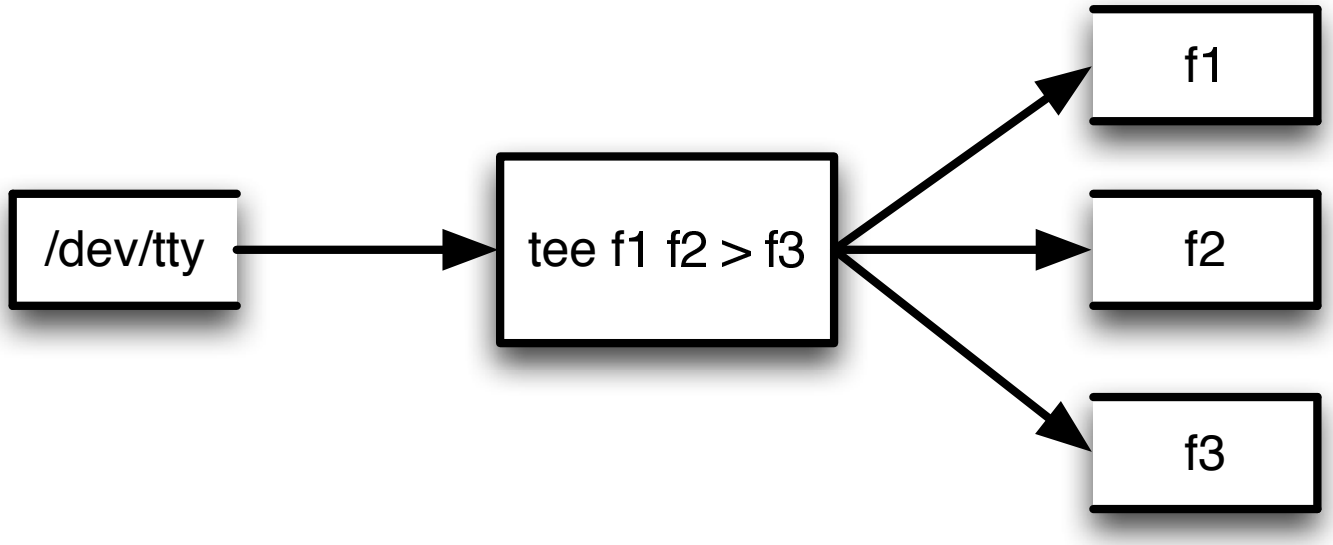
```
--> ls -al /etc > /dev/null ; ls /
```



- Une ligne de commande de la forme : com_1 && com_2 && . . .&& com_n exécute la commande com_{i+1} que si com_i s'est terminée normalement

Quelques commandes : `tee`

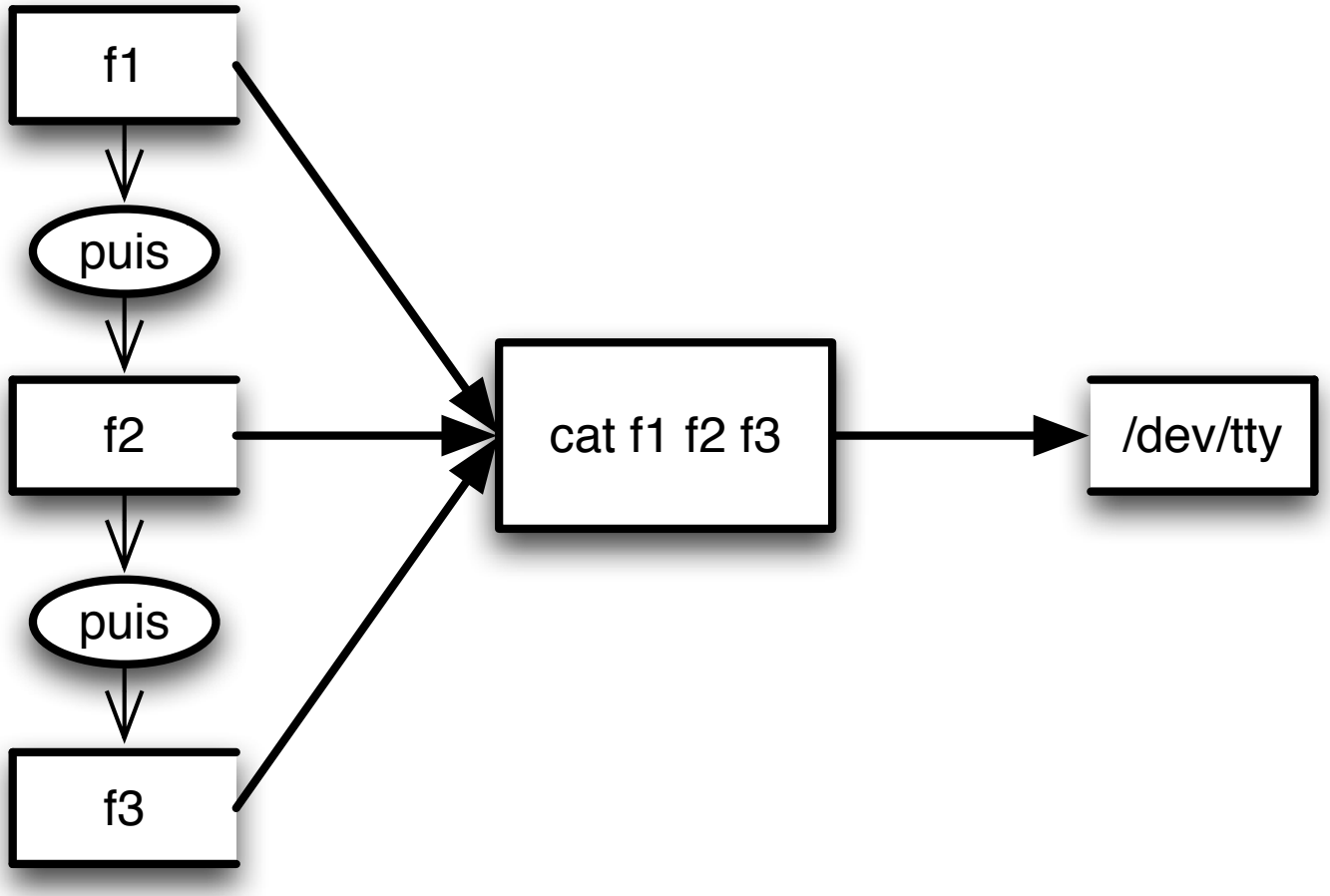
```
--> tee f1 f2 > f3
```



- Utile pour sauver des résultats intermédiaires : `... | tee f1 | ...`

Quelques commandes : `cat`

```
--> cat f1 f2 f3
```



Quelques commandes : `more`

```
--> seq 4012 | more
```

- Affichage page par page de son entrée.

Quelques commandes : `head`

```
--> seq 4012 | head -n 5
```

- Affichage des 5 premières lignes

Quelques commandes : `tail`

```
--> seq 4012 | tail -n 5
```

- Affichage des 5 dernières lignes

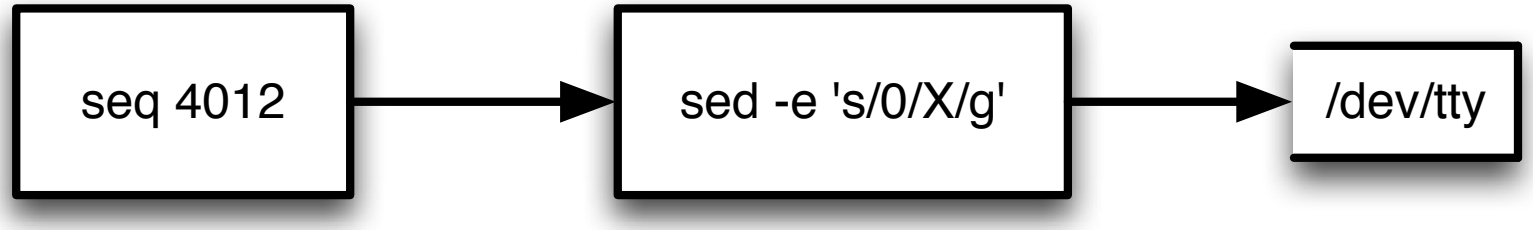
Quelques commandes : grep

```
--> seq 4012 | grep 0$
```

- Affichage des lignes qui terminent par 0.

Quelques commandes : sed

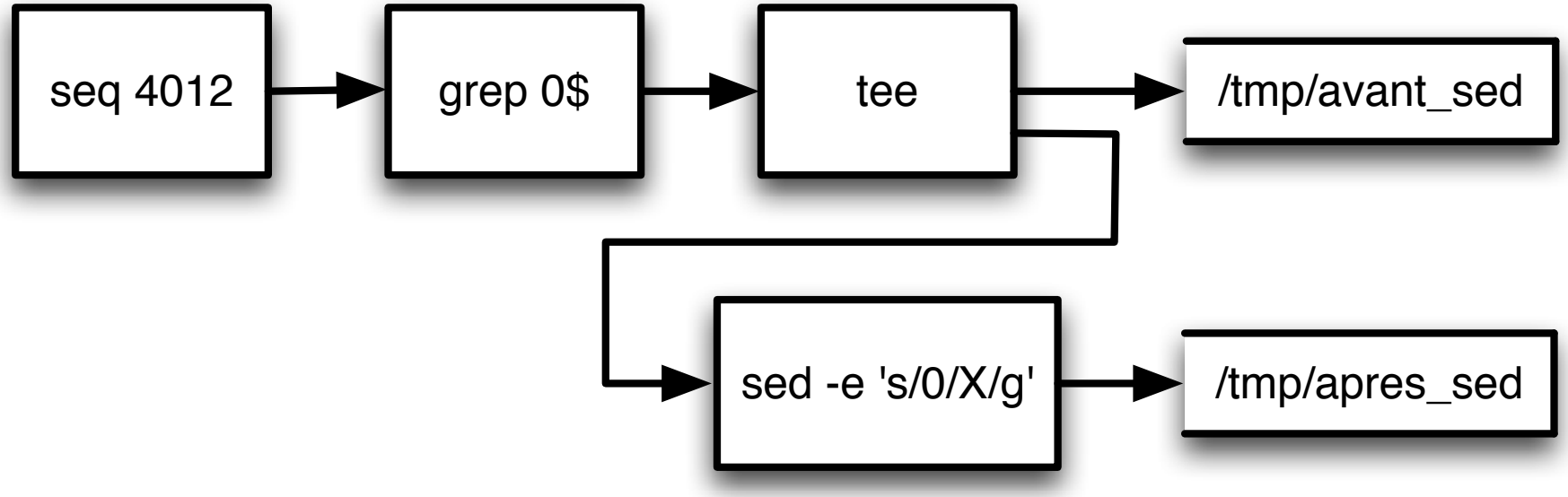
```
--> seq 4012 | sed -e 's/0/X/g'
```



- Remplace tous les 0 par des X.

Exemple de composition

```
--> seq 4012 | grep 0$ | tee /tmp/avant_sed | \
      sed -e 's/0/X/g' > /tmp/apres_sed
```



Shell comme langage de programmation

Script shell

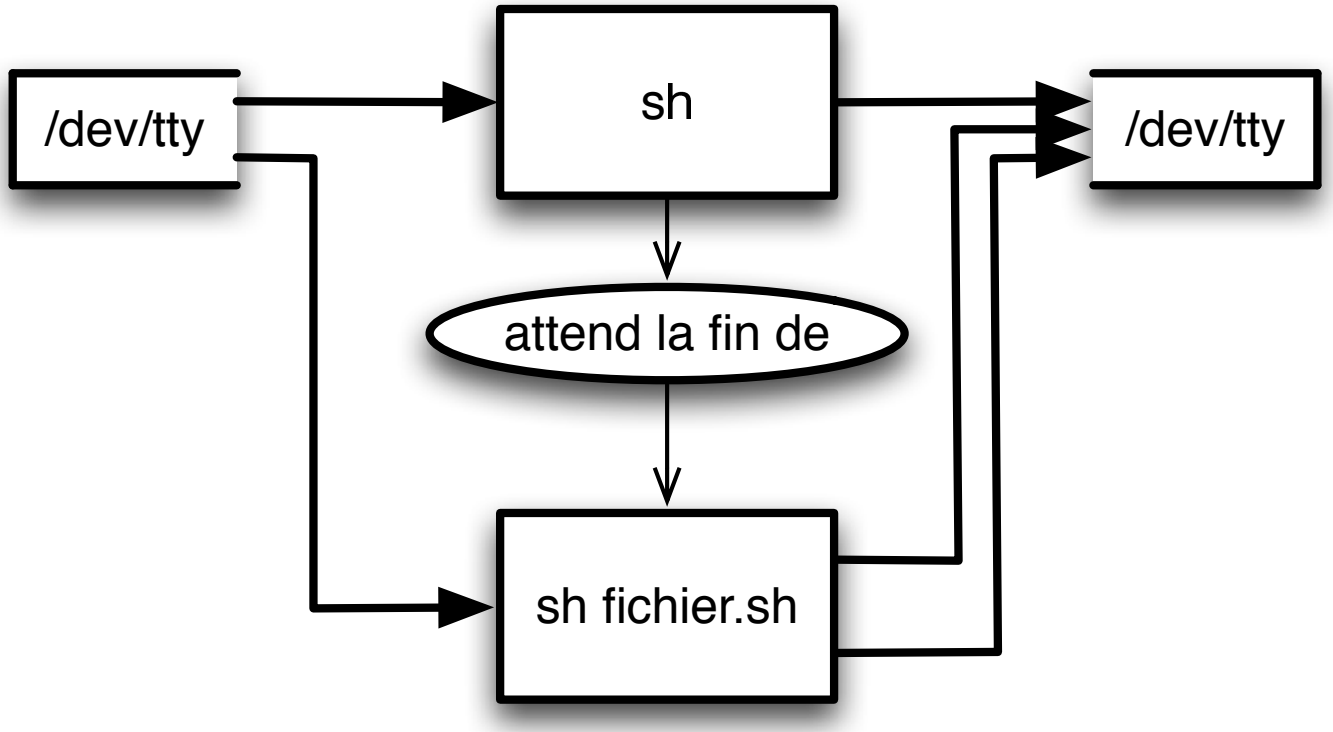
Un script shell contient :

- une suite arbitraire de commandes
- des déclarations de variables
- des structures de contrôle
- des commentaires (lignes qui commencent par #)

Remarque : nous supposons que nous utilisons `ksh` ou `bash`

Exécution d'un script

```
--> sh fichier.sh
```



- Invocation équivalente : `./fichier.sh`
  - si `fichier.sh` a les droits en exécution (cf. `chmod`)
  - et le fichier commence par la ligne : `#!/bin/sh`

Les variables

Déclarations :

- initialisation par une constante : `nom=chaîne`
- initialisation par lecture de l'entrée standard : `read nom`
- initialisation par exécution d'une commande : `nom=‘cmd‘`

Accès :

- `$nom` ou `${nom}`

Exemple :

```
dir=/tm
echo ${dir}p
```

Les variables d'environnement

- afficher toutes les variables : `env`
- définir une nouvelle variable : `export nom`
- exemple de variables d'environnement : `$HOME`, `$PATH`, `$USER` ...

Les arguments

- Exemple `params.sh` :

```
#!/bin/sh
echo la commande a $# parametres
echo liste des parametres : $*
echo nom de la commande : $0
echo premier parametre : $1
echo dixieme parametre : ${10}
```

- La commande `shift` permet de “manger” des paramètres

Code de retour

Terminaison :

- terminer avec le code de retour 3 : `exit 3`
- terminer avec le code de retour 0 : `exit`

Lire de code de retour :

- le code de retour d'une commande est stocké dans la variable `$?`

Expressions

Expressions booléennes :

- code de retour des commandes
- tester si un fichier existe : `test -f fichier` ou `[ -f fichier]`

Expressions arithmétiques :

- la commande `expr` permet de calculer des expressions arithmétiques
- exemple : `expr 1 + 1`

Expressions composées :

- séquence : `;`, `&&`, `||`
- parallèle : `&`, `|`

Structures de bloc

Il y a deux types de bloc : () et {}

```
--> pwd; export mot="Bonjour"
/Users/louis
--> (echo $mot; pwd; cd ..; mot="Au revoir"; pwd; echo $mot)
Bonjour
/Users/louis
/Users
Au revoir
--> pwd; echo $mot
/Users/louis
Bonjour
--> { echo $mot; pwd; cd ..; mot="Au revoir"; pwd; echo $mot; }
Bonjour
/Users/louis
/Users
Au revoir
--> pwd; echo $mot
/Users
Au revoir
```

Structures de bloc

- Redirections : les deux programmes suivants sont équivalents

```
PROG=gros_calcul
echo debut de $PROG    > log
date                   >> log
$PROG
echo fin de $PROG      >> log
date                   >> log

PROG=gros_calcul
{ echo debut de $PROG
  date
  $PROG                > /dev/tty
  echo fin de $PROG
  date ; }             > log
```

Structures de contrôle : conditionnelle

```
if [ $# -ne 1 ]
then
  echo nombre de parametres incorrect >&2
  exit 2
fi
```

```
if test -f $1 ; then echo $1 existe ; else echo "$1 n'existe pas" ; fi
```

- Contrairement au langage C :
  - 0 correspond à Vrai
  - un entier différent de 0 correspond à Faux

Structures de contrôle : filtrage de motif

```
case $# in
  0) pwd ;;
  1) if test -f $1
     then
       cat $1
     elif test -d $1
     then
       ls $1
     else
       echo "erreur sur le parametre" >&2
     fi;;
  *) echo "plus d'un parametre" ;;
esac
```

Structures de contrôle : itérations bornées

Exemple 1 :

```
for dir in /etc /tmp
do
    ls $dir
done
```

Exemple 2 :

```
for arg in $*; do
    echo $arg
done
```

Structures de contrôle : itérations non bornées

- Boucle “tant que” :

```
while condition
  commandes
done
```

- Boucle “jusqu’à ce que” :

```
until condition
  commandes
done
```

Les fonctions

- Déclaration :

```
function nom { commandes ; }
nom () { commandes ; }
```

- Appel : comme un appel de commande

```
f() {
    echo la fonction a $# paramètres
    echo liste des paramètres : $*
    echo nom de la fonction : $0
    echo premier paramètre : $1
    echo dixième paramètre : ${10}
}
f 1 2 3
echo "--------------------"
f $*
```

Les fonctions

count.sh

```
count () {
    CPT=‘expr $CPT + 1‘
    echo Appel $CPT
}

CPT=0
for i in $* ; do
  count
done
echo $# = $CPT
```

Configurer son environnement de travail

- Définir des alias

```
alias l='ls'
alias ll='ls -lh'
alias la='ls -al'
```

- Fichiers exécutés au démarrage de `sh` :
  - `/etc/profile`
  - `~/.profile`
- Pour `bash` regarder aussi `~/.bashrc`.

Un peu de C

Interface avec le programme appelant : valeur de retour

- Exemple, `mtrue` : ma commande `true`

```
#include <stdlib.h>
int main ()
{
  exit(EXIT_SUCCESS);
}
```

- Exemple, `mfalse` : ma commande `false`

```
#include <stdlib.h>
int main ()
{
  exit(EXIT_FAILURE);
}
```

Interface avec le programme appelant : arguments de la commande

- Accès aux arguments de la commande
  - `int main (int argc, char *argv[]);`
    - `argc` nombre d'arguments + 1
    - `argv[0]` nom de la commande
    - `argv[1]` à `argv[argc-1]` arguments
    - `argv[argc]` pointeur nul

Interface avec le programme appelant : arguments de la commande

- Exemple, `mecho` : ma commande `echo`

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {
  int i;
  for( i = 1; i < argc; i++) {
    printf("%s␣", argv[i]);
  }
  putchar('\n');
  exit(EXIT_SUCCESS);
}
```

ma commande `echo`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main (int argc, char *argv[]) {
  int i;
  int arg1 = 1;
  int println = 1;
  if (argc > 1 && !strcmp(argv[1], "-n")) {
    arg1 = 2;
    println = 0;
  }
  else if (argc > 1 && *argv[1] == '-') {
    fprintf(stderr, "%s: invalid option %s", argv[0], argv[1]);
    exit(EXIT_FAILURE);
  }
  if (arg1 < argc) {
    printf("%s", argv[arg1]);
  }
  for( i = arg1+1; i < argc; i++) {
    printf(" %s", argv[i]);
  }
  if (println) {
    putchar('\n');
  }
  exit(EXIT_SUCCESS);
}
```