# Cours de Systèmes d'Exploitation (Unix)

EISTI

2016/2017

Département informatique

BERRICHE Fatima Zahra

# Présentation du module

- **Volume horaire : 15h**
  - 5 séances Cours/TP (3h)
- **Séances n°1** ➔ Commandes linux de base
- **Séances n°2** ➔ Système de gestion de fichier
- **Séances n°3** ➔ Find et grep
- **Séances n°4** ➔ Introduction aux scripts Shell(Partie 1)
- **Séances n°5** ➔ Introduction aux scripts Shell(Partie 2)

# Caractéristiques d'UNIX

- **Portabilité** (écrit en C)
- **Multi-utilisteurs** Plusieurs utilisateurs peuvent se connecter et travailler en même temps.

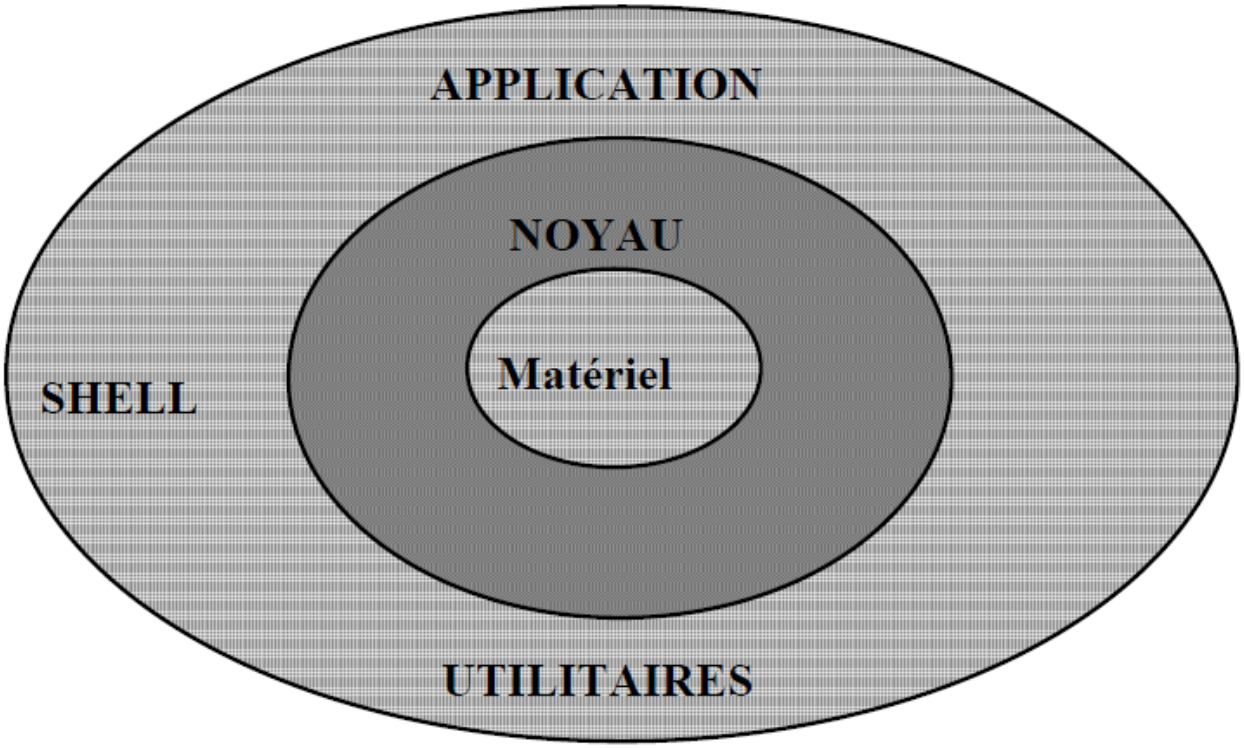
  Chaque utilisateur a son environnement personnel.
- **Multi-tâches** Un même utilisateur peut lancer plusieurs travaux simultanément.
- **Interactif** Il est possible de dialoguer avec l'ordinateur.

Possibilité aussi de lancer des processus (tâches) en arrière plan et en mode différé (batch).

# Caractéristiques d'UNIX

- **Un système de fichiers hiérarchisé** (Organisation arborescente)
- **Un mécanisme de protection**
  - identification des utilisateurs par mot de passe
  - protection des fichiers
  - Un super-utilisateur
- **Une vision simplifiée par l'utilisateur des entrées sorties** Les périphériques sont représentés par des noms de fichier, et peuvent être utilisés comme des fichiers ordinaires)
- **Le choix d'un langage de commandes** : Les shells.

# Structure du système UNIX



# Structure du système UNIX

- Le matériel fournit des pilotes, des périphériques, des opérations élémentaires pour gérer la mémoire.
- Le noyau gère les tâches de base du système :
  - L'initialisation du système
  - La gestion des ressources
  - La gestion des processus
  - La gestion des fichiers
  - La gestion des Entrées/Sorties
- L'utilisateur communique avec le noyau par l'intermédiaire d'un SHELL. Les shells sont aussi des langages de commandes et de programmation.
- Les utilitaires sont des outils d'interfaçage avec le système, de programmation et de communication.

# Les fichiers UNIX

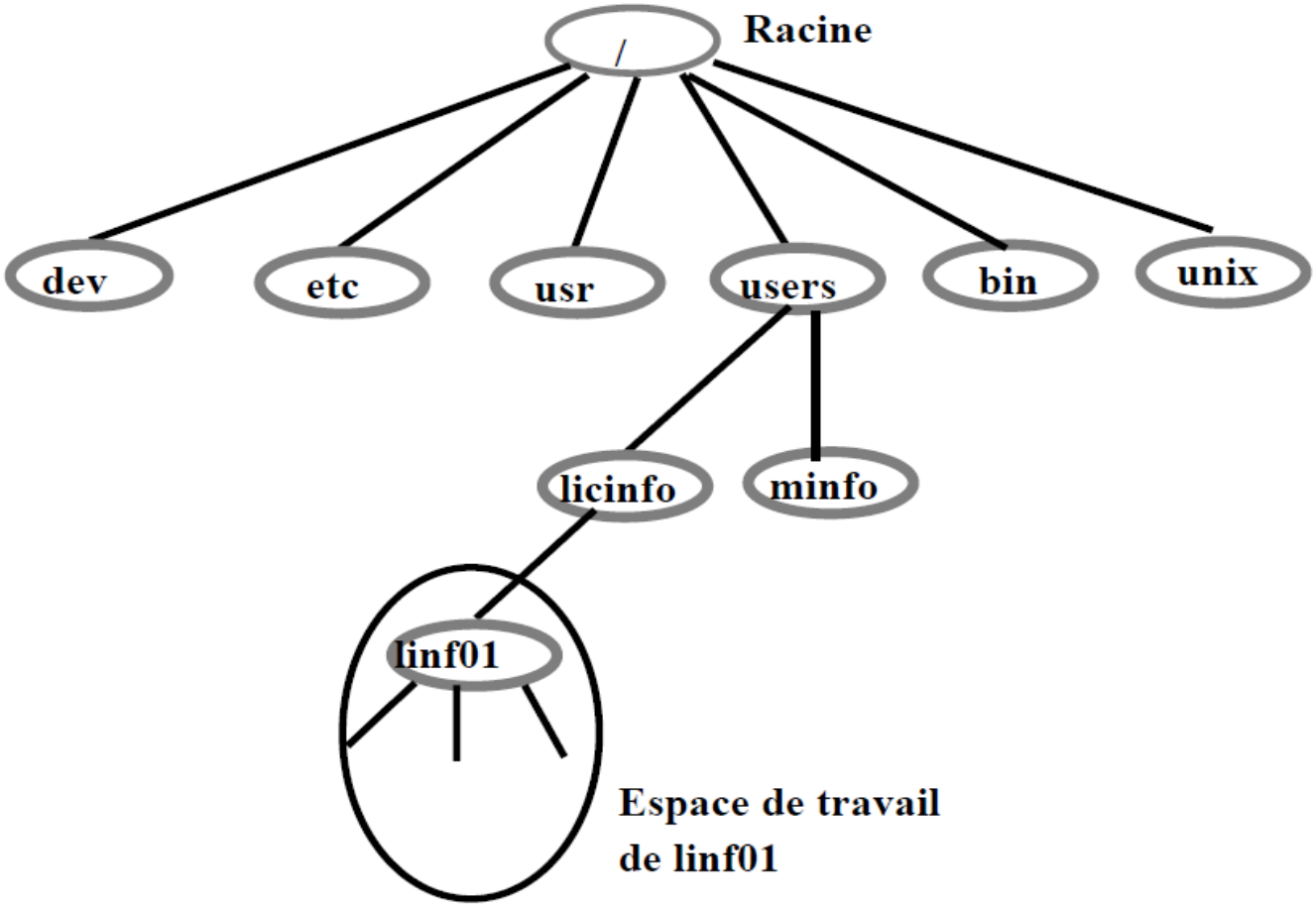
- Un fichier dans UNIX est une séquence d'octets, le noyau n'impose aucune structure spécifique aux fichiers.
- Le rôle d'un fichier est de conserver, de traiter et transmettre de l'information.

# Les fichiers UNIX

**Différents types de fichiers :**

- **Ordinaires** (ascii ou binaires)
- **répertoires** ( contiennent des noms de fichiers et/ou de catalogues)
- **Spéciaux** Ils sont associés aux périphériques Ex : disques, imprimantes, terminaux
- **Les filtres :** permettent la communications entre processus

# Organisation des fichiers



# Organisation des fichiers

- **/dev** contient les fichiers associés aux périphériques :

  Ex : lp : imprimante

  tty**n** terminal

  hk**n** disque dur

  mt**n** bande magnétiques

- **/etc contient les fichiers d'administration**
- **/bin** contient les commandes Unix
- **/usr** commandes et bibliothèques supplémentaires
- **/users** répertoire des utilisateurs
- **/unix** fichiers systèmes

# Déplacement dans le système de fichiers

- A la connexion, l'utilisateur est automatiquement positionné dans son répertoire de connexion .

  Ex : /users/deustiosi/iosi01

- **Pour se déplacer dans la hiérarchie :**

  **cd** <nom_répertoire>

- nom_répertoire peut être absolu ou relatif :

  **absolu :** préciser tout le chemin

  **relatif :** par rapport au répertoire courant

  Ex : cd /users/deustiosi (chemin absolu)

  cd iosi01 (chemin relatif)

**Remarque:** un chemin absolu commence toujours par "/".

# Déplacement dans  le système de fichiers

- **Quelques commandes simples**

  **pwd** affiche le répertoire courant

  **cd** retour au répertoire de connexion

  **cd ..** monter d'un niveau dans la hiérarchie

  **cd ../..** monter de 2 niveaux dans la hiérarchie

  Caractère "." désigne le répertoire courant.

# Principales commandes sur les fichiers

➔ **Opérations de base sur les répertoires**

- **Affichage du contenu d'un répertoire :**

commande : ls [**options**] <nom_répertoire>

l'option **-l** permet d'obtenir l'ensemble des informations relatives à chaque fichier du répertoire :

- type de fichier : "**-**" (fichier ordinaire), "**d**" (répertoire), "**b** ou **c** " (fichiers spéciaux)
- droits d'accès
- nom du propriétaire
- nombre de liens
- taille
- nom
- date de création

# Principales commandes sur les fichiers

L'option -R permet d'afficher récursivement le contenu d'un répertoire.

**Exemple:** ls / (afficher les répertoires à la racine)

dev bin usr users etc unix …

ls -R /dev :

liste des fichiers dans /dev

/bin :

liste des fichiers dans /bin

/users :

liste des fichiers dans/users

# Principales commandes sur les fichiers

- **Création d'un répertoire**

  **mkdir** <nom_répertoire>

  **rmdir** <nom_répertoire> Supprime un répertoire vide.

  **rm -R** <nom_répertoire> supprime tout le répertoire.

- **Visualisation du contenu d'un fichier**

  **cat** <nom_fichier>

  **more** <nom_fichier>

- **Renommage et déplacement d'un fichier**

  **mv** <source> <destination>

**Exemple** :

mv essai.c tp1.c (renommage)

mv tp TPSE (déplacement )

mv tp TPSE/tp2 (déplacement et renommage)

# Principales commandes sur les fichiers

- **Copie d'un fichier**

  **cp** <source> <destination>
- **Création de liens sur un fichier**

  **ln** <ancien> <nouveau>
- **Suppression d'un fichier**

  **rm** <nom_fichier>

**Exemple** :

rm *.o supprimer tous les fichiers d'extension ".o"

rm a* supprimer tous les fichiers dont le nom commence par "a "

rm * supprime tous les fichiers

rm *.? supprimer tous les fichiers ayant une extension d'une lettre.

* : une chaîne quelconque de caractères

? : un caractère quelconque

# Autres commandes utiles

- **touch** créer un fichier
- **wc** donne le nombre de caractères (-c),de mots (-w) ou de lignes (-l)
- **sort** permet de trier par ordre alphabétique les lignes d'un fichier.
- **grep** recherche d'un motif dans un fichier
- **head** afficher les premières lignes
- **tail** afficher les dernières lignes
- **diff** permet de comparer deux fichiers
- **find** permet de recherche un fichier
- **lpr** imprimer un fichier
- **lpq** afficher les fichiers en attente d'impression.
- **lprm** détruire des fichiers en attente d'impression.
- **man**(Très utile) donne le manuel d'utilisation d'une commande.

# Protection des fichiers

- L'accès aux fichiers est déterminé par trois bits de permission: **r w x** (Read, Write, eXecute) applicables à trois classes d'utilisateurs : **u g o** le propriétaire, le groupe et les autres (Users, Group, Others).
- **mode symbolique**

  **chmod** <qui><permission><opération><fichier>

  <qui> valant :

  **u**: utilisateur **g**: groupe **o**:autres et **a** :tous

  <permission> :

  **+** : pour autoriser **-** : pour interdire

  <opération> :

  **r** : lecture **w** : écriture **x** : exécution

# Protection des fichiers

**Exemples**

chmod g+w montp.c (les membres du groupe peuvent écrire dans le fichier "montp.c")

chmod og-rwx montp.c (protection en lecture, écriture et exécution)

- **mode octal**

  **chmod** <permission><fichier>

permission : UGO (**U**ser,**G**roup ,**O**thers : chiffre octal codant les bits **r w x**)

**Exemple**

chmod 740 montp (rend le fichier accessible en lecture au groupe et inaccessible aux autres)

# Protection des fichiers

**Représentation des droits d'accès:**

| Correspondances de représentation des droits | | |
|---|---|---|
| **Droit** | **Valeur alphanumérique** | **Valeur octale** |
| aucun droit | --- | 0 |
| exécution seulement | --x | 1 |
| écriture seulement | -w- | 2 |
| ecriture et exécution | -wx | 3 |
| lecture seulement | r-- | 4 |
| lecture et exécution | r-x | 5 |
| lecture et écriture | rw- | 6 |
| tous les droits (lecture, écriture et exécution) | rwx | 7 |

# Commande find

- La commande **find** permet de retrouver des fichiers à partir de certains critères

  **find** <répertoire de recherche> <critères de recherche>

- Les critères de recherche sont les suivants :**-name** recherche sur le nom du fichier,

  **-perm** recherche sur les droits d'accès du fichier,

  **-links** recherche sur le nombre de liens du fichier,

  **-user** recherche sur le propriétaire du fichier,

  **-group** recherche sur le groupe auquel appartient le fichier,

  **-type** recherche sur le type (d=répertoire, c=caractère, f=fichier normal),

  **-size** recherche sur la taille du fichier en nombre de blocs (1 bloc=512octets),

  **-atime** recherche par date de dernier accès en lecture du fichier,

  **-mtime** recherche par date de dernière modification du fichier,

  **-ctime** recherche par date de création du fichier

# Commande grep

- La commande **grep** permet d**e** rechercher un motif dans un fichier

  **-v** affiche les lignes ne contenant pas la chaîne

  **-c** compte le nombre de lignes contenant la chaîne

  **-n** chaque ligne contenant la chaîne est numérotée

  **-x l**igne correspondant exactement à la chaîne

  **-l** affiche le nom des fichiers qui contiennent la chaîne

**Exemple**

grep -l printf *.c (affiche la liste des fichiers contenant "printf").

# Scripts Shell

# Présentation

- Le Shell n'est pas qu'un simple interpréteur de commandes, mais dispose d'un véritable langage de programmation ➔ Script Shell avec notamment une gestion des variables, des tests et des boucles, des opérations sur variables, des fonctions

- Le Script Shell permet d'automatiser une série d'opérations.
- Enchaînement de plusieurs commandes Unix
- Peu d'interaction avec l'utilisateur

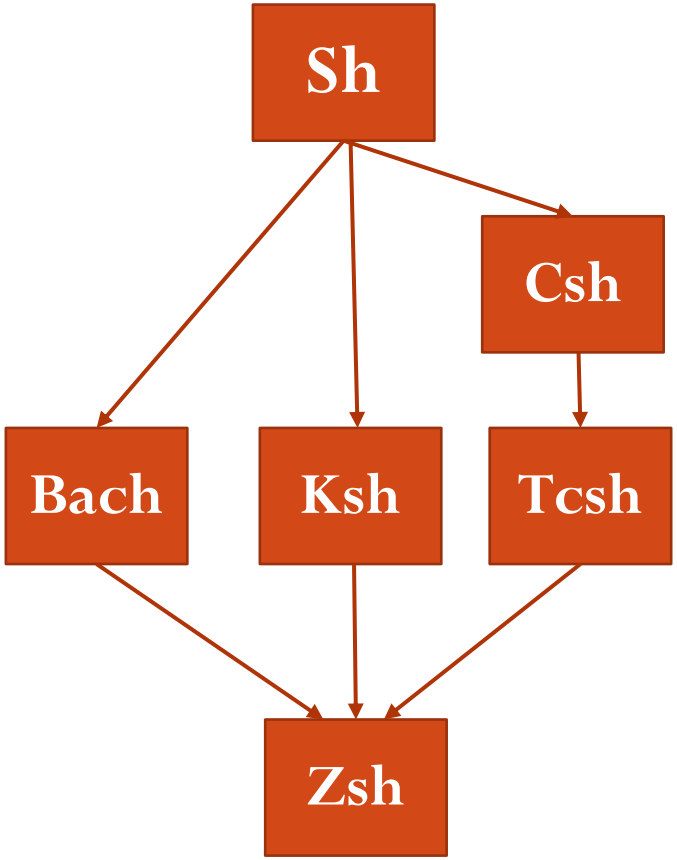
# Langages de script

- Différents types de shell ⟺ différents langages de script
- Plusieurs types de langages
- Écriture rapide
- Pas de compilation
- Commandes centrées sur les objets existants (commandes unix)

# Famille des scritps

- **sh** : *Bourne Shell*. SHell d'origine (le plus simple)
- **bash** : *Bourne Again Shell*. Une amélioration du *Bourne Shell*, disponible par défaut sous Linux et Mac OS X.
- **ksh** : *Korn Shell*. Un shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec bash.
- **csh** : *C Shell*. Un shell utilisant une syntaxe proche du langage C.
- **tcsh** : *Tenex C Shell*. Amélioration du *C Shell*.
- **zsh** : *Z Shell*. Shell assez récent reprenant les meilleures idées de bash, ksh et tcsh.

# Famille des scritps




Chacun hérite de la plupart des fonctionnalités de son ancêtre.

# Exemple Script shell

- # : Commentaires
- echo : écriture sur sortie standard

```
#!/bin/bash
# Fichier "helloworld"
echo "Hello world“
```

- Pour exécuter un script, il doit avoir les droits en exécution…(+x)

# Variables du shell

- **Stocker temporairement des informations en mémoire.**
- Définition d'une variable :

  maVariable=" une va r i a b l e "

  maVariable2=54
- un nom = et une valeur textuelle ou littérale
- PAS D'ESPACE AUTOUR DU EGAL

# Utilisation des variables

**echo** $maVariable

une variable

**echo** $maVariable2

54

echo maVariable

maVariable

- Besoin de précéder le nom par $ pour avoir la valeur
- echo : permet d'afficher ce qui suit

# Substitution

- Substitution dans les chaînes de caractères

  maVariable=" variable"

  echo "Voici une $maVariable "

➔ Voici une variable

- Non substitution dans les suites de caractères

  echo 'Voici une $maVariable '

  Voici une $maVariable

- Stockage du résultat d'une commande dans une variable

  maVariable = ` ls -l | grep -e / ^ d / `

# Résumé

- Trois types de chaînes
  - Chaînes littérales : ’
  - Chaînes substituées : "
  - Chaînes exécutées : ‘

- ${toto} : permet de borner le nom de la variable
- Lève l’ambiguîté lors de l’interpéttation

# Les tests

- Le shell propose deux principales façons de réaliser un test ; ces deux méthodes sont équivalentes :
  - **test** expression
  - [ expression ]

# Principaux tests

- test -f monFichier

  vrai si monFichier existe
- test -r monFichier

  vrai si monFichier est accessible en lecture
- test -w monFichier

  vrai si monFichier est accessible en écriture
- test -d monFichier

  vrai si monFichier est un répertoire

# Tests sur les entiers

- entier1 -eq entier2 : Vrai si entier1 est égal à entier2.
- entier1 -ge entier2 : Vrai si entier1 est supérieur ou égal à entier2.
- entier1 -gt entier2 : Vrai si entier1 est strictement supérieur à entier2.
- entier1 -le entier2 : Vrai si entier1 est inférieur ou égal à entier2.
- entier1 -lt entier2 : Vrai si entier1 est strictement inférieur à entier2.
- entier1 -ne entier2 : Vrai si entier1 est différent de entier2.

# Tests sur les chaînes

- -n "chaîne" ➔ Vrai si la chaîne n'est pas vide.
- -z "chaîne" ➔ Vrai si la chaîne est vide.
- "chaine1" = "chaine2" ➔ Vrai si les deux chaînes sont identiques.
- "chaine1" != "chaine2" ➔ Vrai si les deux chaînes sont différentes.

# Combinaison de tests

- NON expression

  test ! expr

  [ ! expr ]

- expr1 OU expr2

  test expr1 -o expr2

  [ expr1 -o expr2 ]

- expr1 ET expr2

  test expr1 -a expr2

  [ expr1 -a expr2 ]

# Structures de contrôles

- Conditionnelles

**if** *condition*

**then** *commandes*

**fi**

# Structures de contrôles

- Alternatives simples

**if** *condition*

**then** *commandes*

**else** *commandes*

**fi**

# Structures de contrôles

- Alternatives multiples

**if** *condition*

**then** *commandes*

**elif** *condition*

**then** *commandes*

**else** *commandes*

**fi**

# Structures de contrôles

- Aiguillages multiples

```
case var in
   cas1 ) . . . . . ;;

   cas2 ) . . . . . ;;
   *) . . . . . ;;

esac
```

# Structures de contrôles

- Exemple

```
case $var in
[0-9]*) echo "$var est un nombre.";;
[a-zA-Z]*) echo "$var est un mot.";;
*) echo "$var n'est ni un nombre ni un mot.";;

esac
```

# Structures de contrôles

- Boucle For

➔ Permet de parcourir une liste de valeurs et de boucler autant de fois qu'il y a de valeurs.

**for** var **in** l i s t e _ v a l e u r

**do** . . . . . .

**done**

# Structures de contrôles

- Exemple

```
for variable in  'valeur1' 'valeur2' 'valeur3'
do
echo "La variable vaut $variable"
done
```

# Structures de contrôles

- Boucle While

**while** cond

**do**

. . .

. . .

**done**

# Structures de contrôles

- Exemple

```
echo "Tapez le mot de passe :   "
read password

while test "$password" != mot de passe correct
do
echo "Mot de passe incorrect."
echo "Tapez le mot de passe"
read password
done
```

# Structures de contrôles

- Boucle Until

**until** *condition*
**do**
*commandes*
**done**

# Structures de contrôles

- Exemple

```
echo "Tapez le mot de passe :   "
read password

until test "$password" = mot de passe correct
do
echo "Mot de passe incorrect."
echo "Tapez le mot de passe"
read password
done
```

# Divers

- Expression arithmétique : $(( expr ))
- Exécuter un script en lui passant des arguments comme pour n'importe quelle autre commande

| Variable | Description |
|---|---|
| $# | Nombre d'arguments. |
| $* | Liste des arguments. |
| $0 | Nom de la commande. |
| $1 | Valeur du premier paramètre. |
| $i | Valeur du ième paramètre si i compris entre 1 et 9. |
| $9 | Valeur du neuvième paramètre. |

# Divers

- Fonction

**nom** ()

{ instruction1

instruction2 … }

# Divers

- Exemple

```
#!/bin/bash
effacer_fichier (){
echo "$1"
echo "Voulez-vous vraiment l'effacer ? (o/n)   "
 read reponse
if [ $reponse = "o" ]
then
rm $1
fi }
for fichier in *.bak
do
effacer_fichier $fichier
done
```

```
Rectification :
        for fichier in `ls *.bak`    # le *.bak n'est pas interprêté seul
```