

# Introduction à Linux

## Version 1.6 (2011 - 2012)



# Introduction à Linux : plan (1)

## I. Introduction

- ★ L'environnement de travail
- ★ L'interpréteur de commandes
- ★ Aide et documentation

# Introduction à Linux : plan (2)

## II. Principes et commandes de base

- ★ Système de fichiers
- ★ Commandes de base
- ★ Liens symboliques et physiques
- ★ Droits d'accès

# Introduction à Linux : plan (3)

## III. Redirections et processus

- ★ Entrées/sorties et redirections
- ★ Les tuyaux
- ★ Contrôle de tâche et de processus

# Introduction à Linux : plan (4)

## IV. Introduction au shell

- ★ Variables
- ★ Substitution de commandes
- ★ Alias
- ★ La commande for
- ★ La commande seq

# Introduction à Linux : plan (5)

## V. Programmation shell

- ★ Script shell
- ★ Variables, paramètres et substitutions
- ★ Structures de contrôle
- ★ Fonctions
- ★ Commandes utiles

# Introduction à Linux : plan (6)

## VI. Exercices de synthèse

- ★ Gestion de sauvegardes
- ★ Gestion d'une poubelle
- ★ Un mécanisme d'annulation

# Introduction à Linux : plan (7)

## VII. Le monde Linux

- ★ Le projet GNU et la licence GPL
- ★ Les distributions Linux
- ★ Linux et ses concurrents

# Introduction à Linux : plan (8)

## VIII. Epilogue

- ★ Ressources internet sur Linux
- ★ Licence et historique du cours

# I. Introduction

- ★ **L'environnement de travail**
- ★ **Principes de base de Linux**
- ★ **L'interpréteur de commandes**
- ★ **Aide et documentation**

# Principales caractéristiques

## Depuis le début dans les années 1970

- ★ Multi-utilisateur et sécurisé

Par défaut, les utilisateurs ordinaires ne peuvent pas toucher aux fichiers d'autres utilisateurs.

En particulier, ils ne peuvent ni modifier les paramètres du système, ni supprimer des programmes, etc.

Utilisateur “root” : utilisateur administrateur, ayant tous les droits.

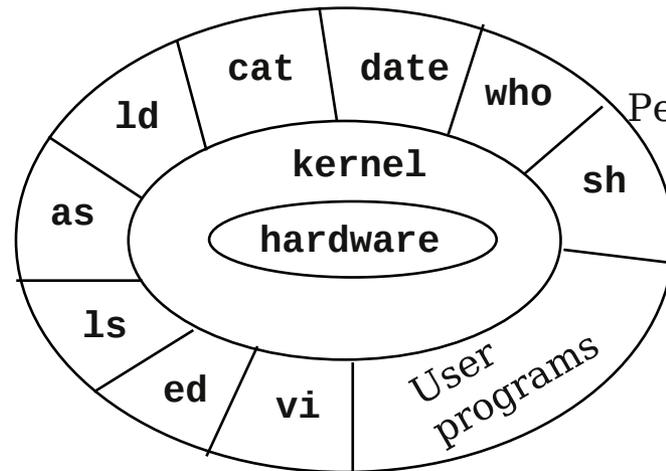
- ★ Multi-tâche, prise en charge de multiples processeurs

- ★ Extrêmement flexible

- ★ Prise en charge du réseau

# Couches dans un système Unix

- ★ Noyau
- ★ Bibliothèque C
- ★ Bibliothèques système
- ★ Bibliothèques d'applications
- ★ Programmes utilisateurs :  
au même niveau que les  
commandes systèmes



Peut être modifié

# Environnement de travail (1)

- ★ L'accès aux fonctionnalités et aux données dans votre environnement de travail Linux s'effectue de deux façons complémentaires :
  - ★ via une interface graphique : assurée par un Window Manager (**KDE**)
  - ★ via un (ou plusieurs) terminal (ou terminaux) en mode texte constitué par une fenêtre dans laquelle on interagit avec le système (**Konsole**  )
- ★ Les fonctionnalités graphiques changent avec le Window Manager : elles constituent une interface graphique aux commandes de bases du système
- ★ Les commandes de base ne changent pas : elles sont les mêmes (à quelques exceptions près) pour tous les systèmes Unix et sont standardisées
- ★ Tout est paramétrable et programmable : l'environnement graphique est entièrement paramétrable et le système de base sous-jacent est entièrement programmable

# Environnement de travail (2)

The screenshot displays a Linux desktop environment with a dark red background. In the top-left corner, there is a menu bar with 'Applications', 'Raccourcis', and 'Système'. The top-right corner shows the date and time: 'mar 28 sep, 19:17'. A terminal window titled 'ferry@ferry-laptop: ~/Images - Shell - Konsole' is open, showing the following command and output:

```
ferry@ferry-laptop:~/Images$ ls
tux wallpapers
ferry@ferry-laptop:~/Images$
```

In the background, a Mozilla Firefox browser window is open, displaying the website 'PolytechNice-Sophia: Sciences Informatiques'. The browser's address bar shows 'http://www.polytechnice.fr/page11.html'. The website content includes the logos for 'Université Nice Sophia Antipolis' and 'POLYTECH NICE-SOPHIA', along with navigation links for 'Futurs Elèves', 'Elèves', 'Collaborateurs', and 'Entreprises, R&D'. The main content area is titled 'Sciences Informatiques' and contains the following text:

**Sciences Informatiques :**

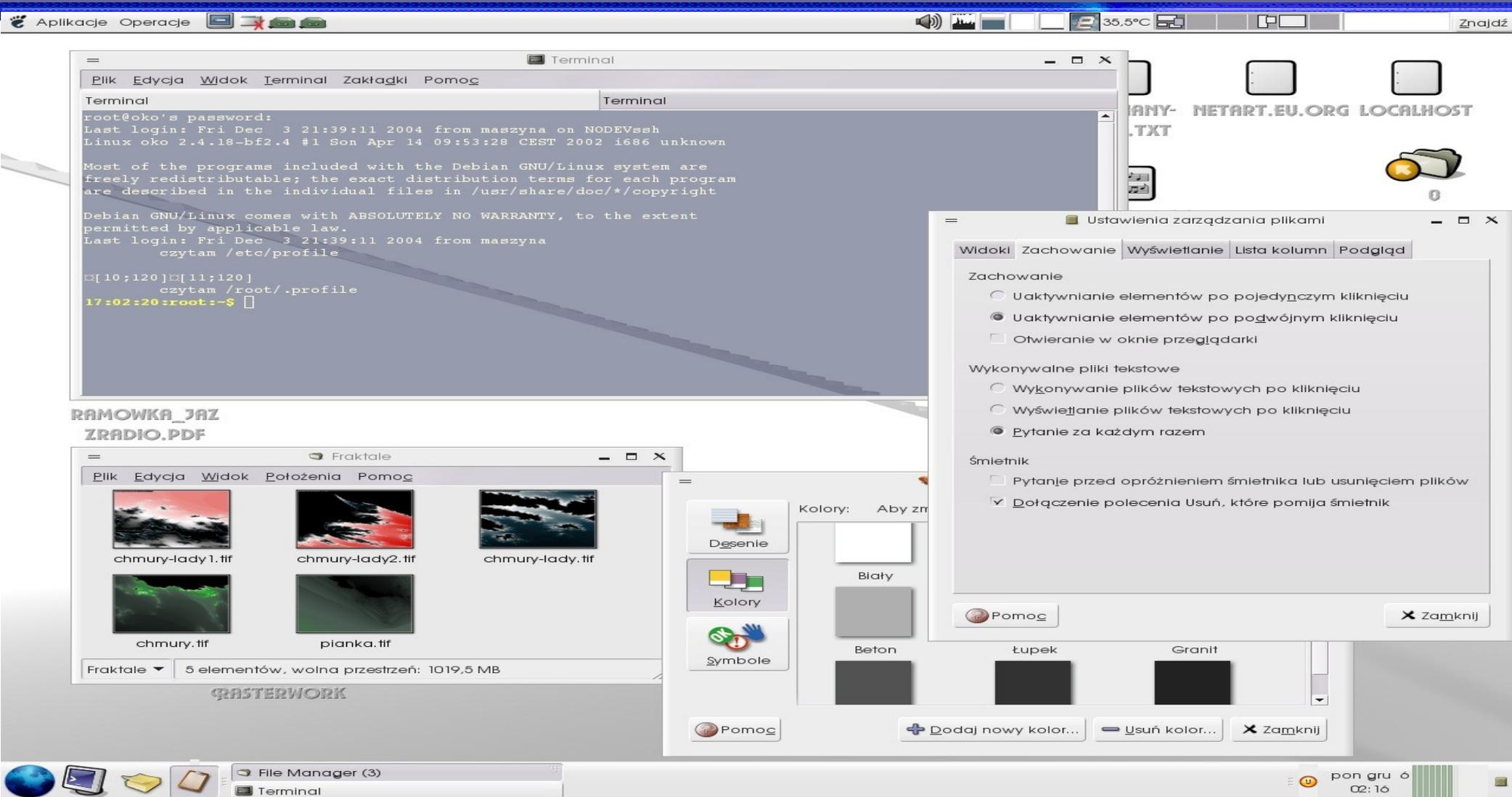
» **Philosophie & objectifs de la spécialité :**

Nous souhaitons former, au meilleur niveau international, de véritables professionnels maîtrisant les techniques informatiques d'aujourd'hui, aptes à assimiler celles de demain et à innover dans ce domaine sans cesse en évolution et qui offre un large spectre de métiers.

Les ingénieurs en informatique de Polytech'Nice - Sophia sont capables de répondre à la fois aux attentes des entreprises du secteur professionnel, dans les domaines scientifiques fortement utilisateurs d'informatique (mathématiques, électronique, biologie) ou dans les

Le thème de votre Gnome !

# Environnement de travail (3)

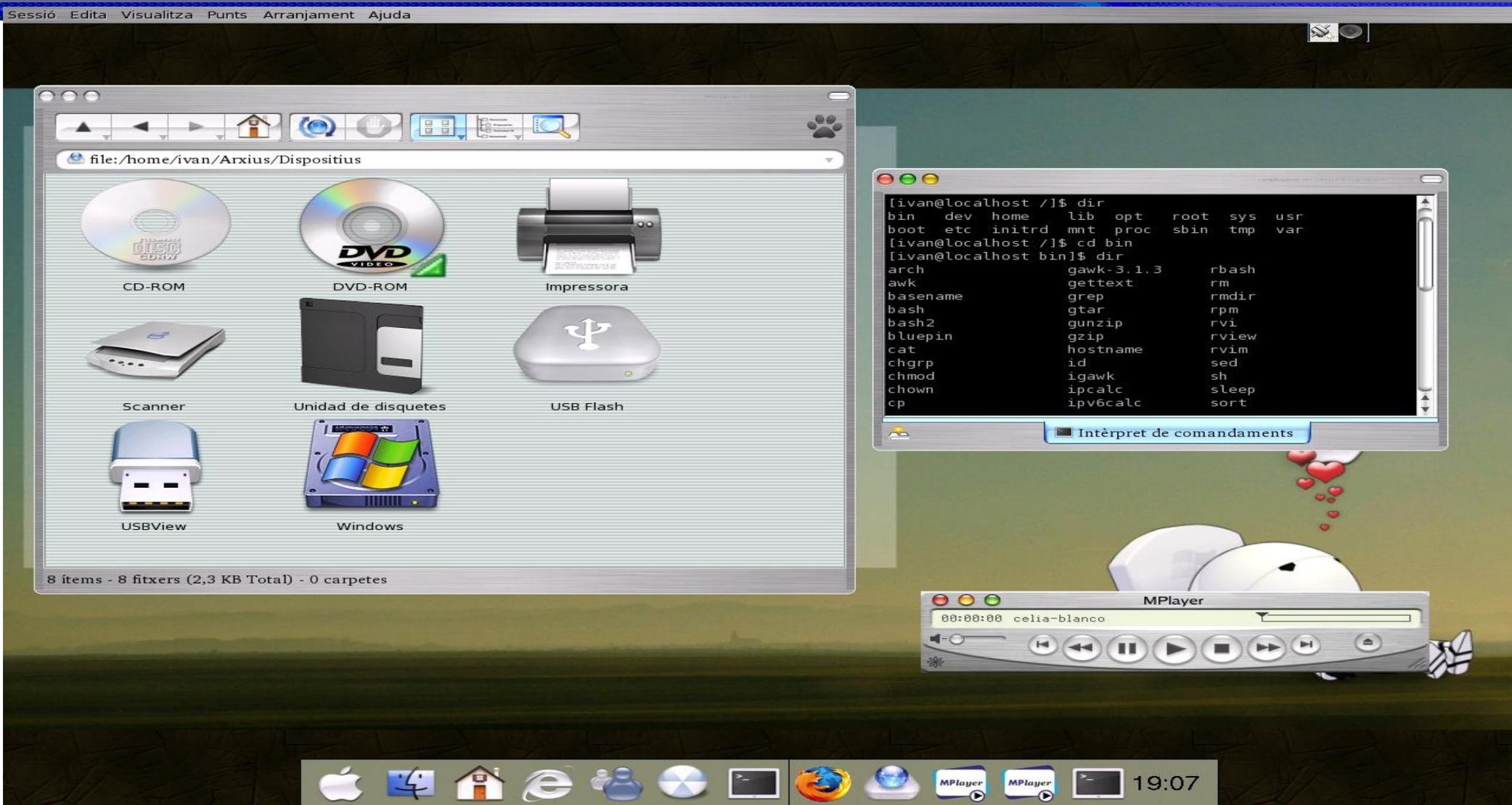


Le thème Ana pour GNOME (GTK 2.x) d'après Tarchalski

## I. Introduction

Environnement de travail

# Environnement de travail (4)



Le thème Mac OS X pour KDE 3.2+ d'après Ivanciko

# Quelques applications standards

- ★ **XEmacs** , **Kate** : éditeurs de texte
- ★ **OpenOffice**, **KOffice** : suites bureautique complètes
- ★ **Mozilla**, **Firefox**, **Konqueror**: navigateurs Internet
- ★ **Thunderbird**, **Kmail** : clients de messagerie
- ★ **Kopete**, **Gaim** : clients de messagerie instantanée
- ★ **GIMP** : éditeur graphique très puissant
- ★ **Gwenview** : afficheur de galeries de photos
- ★ **Amarok**, **XMMS**, **Noatun**, **Kaffeine** : lecteurs audio et multimédia
- ★ **K3b** : graveur de CD et DVD
- ★ **aMule**, **KMLDonkey**, **KTorrent** : clients P2P

# Éditeurs de texte

**Un éditeur de texte est une application qui sert à fabriquer (et modifier) des fichiers de texte (ASCII ou autres)**

Éditeurs de texte graphiques

- ★ **Emacs, XEmacs (très vivement conseillé !)**

- ★ **Kate** (uniquement sous KDE)

Éditeurs en mode texte uniquement

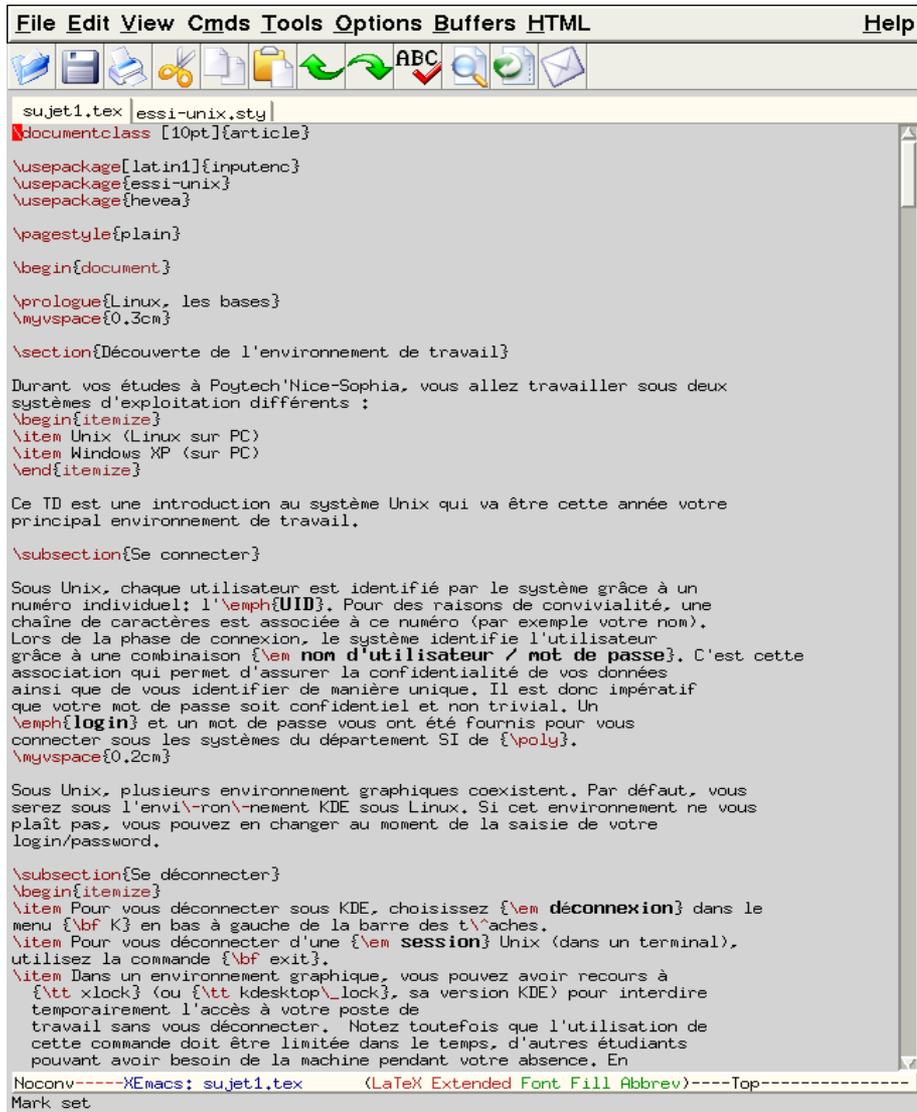
- ★ **vi** (semi-graphique et très populaire)

- ★ **nano** (très léger en mémoire et facile à apprendre)

Éditeurs spécialisés

- ★ **Quanta+** (pour le web)

# Emacs / XEmacs



The screenshot shows the Emacs editor window with a menu bar (File, Edit, View, Cmds, Tools, Options, Buffers, HTML, Help) and a toolbar. The main text area contains LaTeX source code for a document titled 'sujet1.tex' using the 'essi-unix.sty' style. The code includes package declarations, page style settings, and document content. The document content is in French and discusses the Unix environment, user identification, and connection procedures. The status bar at the bottom indicates 'Noconv---XEmacs: sujet1.tex (LaTeX Extended Font Fill Abbrev)---Top' and 'Mark set'.

```
File Edit View Cmds Tools Options Buffers HTML Help
sujet1.tex | essi-unix.sty |
\documentclass [10pt]{article}

\usepackage[latin1]{inputenc}
\usepackage{essi-unix}
\usepackage{hevea}

\pagestyle{plain}

\begin{document}

\prologue{Linux, les bases}
\myvspace{0.3cm}

\section{Découverte de l'environnement de travail}

Durant vos études à Poytech'Nice-Sophia, vous allez travailler sous deux
systèmes d'exploitation différents :
\begin{itemize}
\item Unix (Linux sur PC)
\item Windows XP (sur PC)
\end{itemize}

Ce TD est une introduction au système Unix qui va être cette année votre
principal environnement de travail.

\subsection{Se connecter}

Sous Unix, chaque utilisateur est identifié par le système grâce à un
numéro individuel: l'\emph{UID}. Pour des raisons de convivialité, une
chaîne de caractères est associée à ce numéro (par exemple votre nom).
Lors de la phase de connexion, le système identifie l'utilisateur
grâce à une combinaison {\em nom d'utilisateur / mot de passe}. C'est cette
association qui permet d'assurer la confidentialité de vos données
ainsi que de vous identifier de manière unique. Il est donc impératif
que votre mot de passe soit confidentiel et non trivial. Un
\emph{login} et un mot de passe vous ont été fournis pour vous
connecter sous les systèmes du département SI de {\poly}.
\myvspace{0.2cm}

Sous Unix, plusieurs environnement graphiques coexistent. Par défaut, vous
serez sous l'envi\ron\nement KDE sous Linux. Si cet environnement ne vous
plaît pas, vous pouvez en changer au moment de la saisie de votre
login/password.

\subsection{Se déconnecter}
\begin{itemize}
\item Pour vous déconnecter sous KDE, choisissez {\em déconnexion} dans le
menu {\bf K} en bas à gauche de la barre des tâches.
\item Pour vous déconnecter d'une {\em session} Unix (dans un terminal),
utilisez la commande {\bf exit}.
\item Dans un environnement graphique, vous pouvez avoir recours à
{\tt xlock} (ou {\tt kdesktop\_lock}, sa version KDE) pour interdire
temporairement l'accès à votre poste de
travail sans vous déconnecter. Notez toutefois que l'utilisation de
cette commande doit être limitée dans le temps, d'autres étudiants
pouvant avoir besoin de la machine pendant votre absence. En
Noconv---XEmacs: sujet1.tex (LaTeX Extended Font Fill Abbrev)---Top
Mark set
```

- Emacs et XEmacs sont très semblables (Xemacs par défaut)
- Fonctionnalités d'éditeur de texte extrêmement puissantes
- Editeur multi-modes programmable
- Parfait pour les utilisateurs avancés
- Bien plus d'un simple éditeur de texte (courrier, shell, navigateur)
- Raccourcis clavier non standards mais très complets et très puissants
- Nécessite un apprentissage assez long

# Interpréteurs de commandes (1)

- ★ Interpréteurs de commandes : outils pour exécuter des commandes tapées par un utilisateur dans un terminal
- ★ Appelés “shells” (coquilles) parce qu’ils masquent sous leur surface les détails du système d’exploitation sous-jacent
- ★ Les commandes sont tapées dans un terminal en mode texte, constitué par une fenêtre dans un environnement graphique, ou plus rarement par une console sur un écran en texte seul
- ★ Les résultats sont aussi affichés sur le terminal. Une sortie graphique n’est pas obligatoire (pour la plupart des commandes la sortie est du texte)
- ★ Les interpréteurs de commandes peuvent être programmables : ils fournissent toutes les ressources nécessaire pour l’écriture de programmes complexes (variables, conditions, boucles...)

# Interpréteurs de commandes (2)



```
[wolfix@home/gaetano] ls
asd/  cnam/  Downloads/  imash/  lfa/  Mail/  monet/  projets/  TMP/  wolfix.old/
bin/  Desktop/  Essi/  isia/  lib/  Master/  music/  RI/  unix/
C/  docs/  images/  js/  lpmi/  media/  perso/  tmp/  web/
[wolfix@home/gaetano] ls asd/
0304/  colles/  cours/  index.html  java/  misc/  solutions/  sujets/  support/
[wolfix@home/gaetano] ls bin/
2pages*  latexps*  mycvs*  pub  syncport*  twopages*  websync*  xunison*
backup*  listing*  print  start-ssh-add.sh*  tidy*  unison*  xframe*
fromweb*  mkps*  prosper*  startXemacs*  toweb*  webclient*  xskey*
[wolfix@home/gaetano] ls Downloads/
12412-flatcolourscheme.tar.gz  CrystalColor.kth  linux.love-1600x1200.png
14.jpg  essential-20050412.tar.bz2  linux.love-800x600.png
24253-kubuntu-lineart16-10.svgz.tar.gz  FairytaleWorld.tar.gz  linux-love.tar.gz
A DHTML Calendar.zip  gestaction3D.avi  README
ApplicationIN_kopplad0506.pdf  install_flash_player_7_linux-1.tar.gz  Thumbs.db
Cezanne_packaged.tar.bz2  knifty-0.4.2/  Torchlight_0_2_0_tar.bz2
CrystalClear.tar.gz  linux.love-1024x768.png
[wolfix@home/gaetano] ls projets/
0203/  0304/  0405/
[wolfix@home/gaetano] ls unix/
CoursZsh.ps  docs/  eg/  electrons/  essi/  harmo/
[wolfix@home/gaetano] date
Mon Aug 8 17:27:05 CEST 2005
[wolfix@home/gaetano]
```

penguins. all together now

Une session konsole sous KDE 3.4

# Interpréteurs les plus courants

★ **sh** : le Bourne shell (par Steve Bourne, obsolète)

Le shell de base qu'on trouvait dans tous les systèmes Unix : maintenant obsolète et remplacé par le **bash**

★ **csh** : le C shell (obsolète)

Shell avec une syntaxe à la C, qui a connu son heure de gloire

★ **tcsh** : le TC shell (toujours très populaire)

Une implémentation compatible avec le C shell, avec des fonctionnalités avancées (complète les noms de commandes, rappel de commandes antérieures et bien d'autres)

★ **bash** : le Bourne Again shell (par Steve Bourne, le plus populaire)

Une version améliorée de sh avec de nombreuses fonctions nouvelles

★ **ksh** : le Korn Shell (par David Korn, développé chez AT&T)

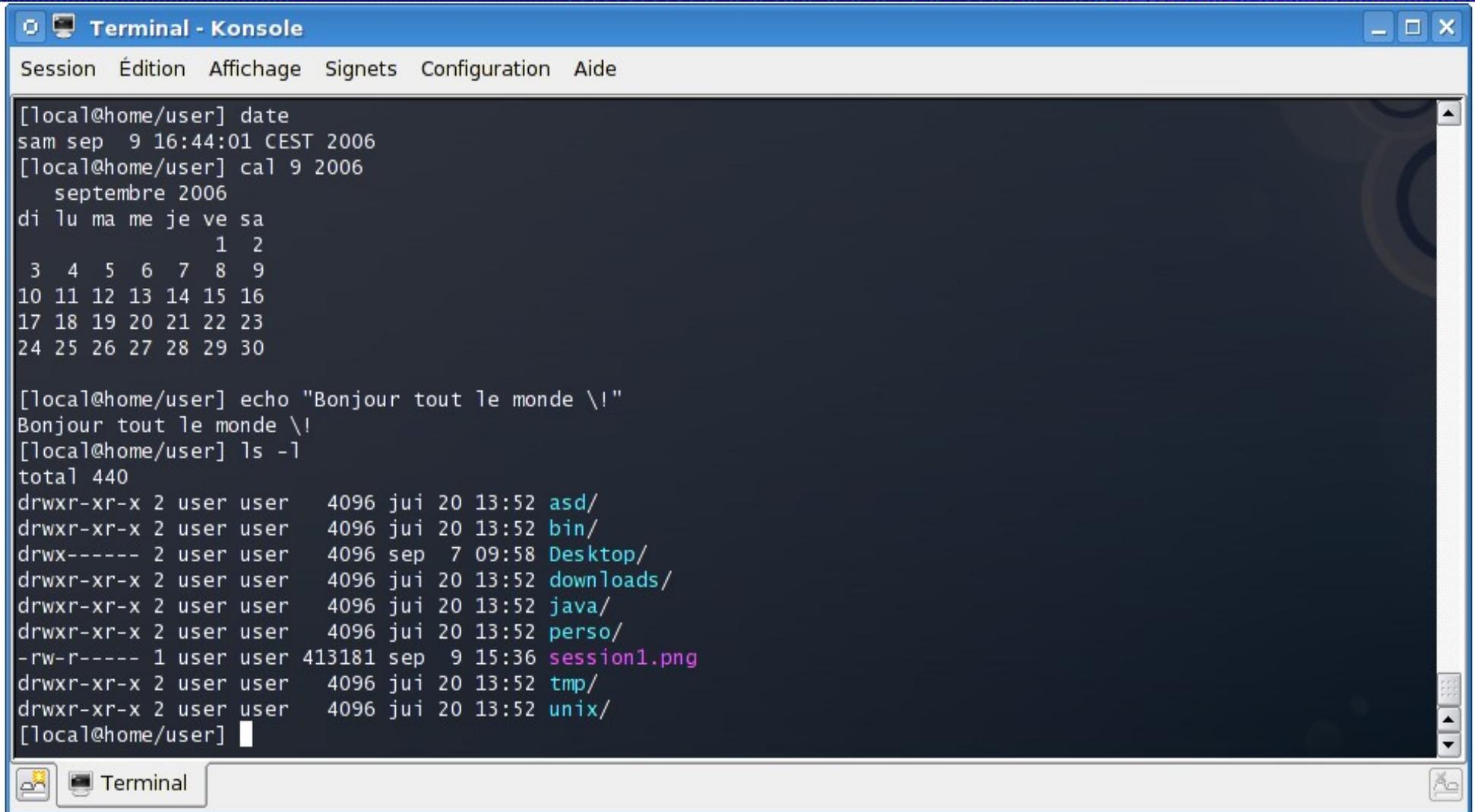
Compatible avec le Bourne shell et inclut de nombreuses fonctions du C shell

★ **zsh** : un Bourne shell étendu avec beaucoup d'améliorations. Il reprend la plupart des fonctions les plus pratiques de bash, ksh et tcsh

# Syntaxe des commandes (1)

- ★ Une commande Unix peut avoir 0, un nombre fixe ou un nombre variable d'arguments, plus un certain nombre d'options. Une option commence en général par un tiret ( - ) et deux tirets à la suite ( -- ) indiquent la fin des options sur une ligne de commande
- ★ `[hal@home/bob] date`  
Une commande sans paramètre qui affiche la date et l'heure courante
- ★ `[hal@home/bob] cal 2006`  
Une commande avec un paramètre (une année) qui affiche le calendrier complet de l'année en question
- ★ `[hal@home/bob] cal 9 2006`  
La même commande avec deux paramètres, le mois et l'année
- ★ `[hal@home/bob] echo`  
Sans argument cette commande affiche une ligne vide
- ★ `[hal@home/bob] echo Bonjour tout le monde`  
Affiche les quatres arguments `Bonjour`, `tout`, `le` et `monde`, et va à la ligne
- ★ `[hal@home/bob] echo -n Bonjour tout le monde`  
L'option `-n` empêche le passage à la ligne

# Syntaxe des commandes (2)



The screenshot shows a terminal window titled "Terminal - Konsole" with a menu bar containing "Session", "Édition", "Affichage", "Signets", "Configuration", and "Aide". The terminal content is as follows:

```
[local@home/user] date
sam sep  9 16:44:01 CEST 2006
[local@home/user] cal 9 2006
  septembre 2006
di lu ma me je ve sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

[local@home/user] echo "Bonjour tout le monde \!"
Bonjour tout le monde \!
[local@home/user] ls -l
total 440
drwxr-xr-x 2 user user  4096 jui 20 13:52 asd/
drwxr-xr-x 2 user user  4096 jui 20 13:52 bin/
drwx----- 2 user user  4096 sep  7 09:58 Desktop/
drwxr-xr-x 2 user user  4096 jui 20 13:52 downloads/
drwxr-xr-x 2 user user  4096 jui 20 13:52 java/
drwxr-xr-x 2 user user  4096 jui 20 13:52 perso/
-rw-r----- 1 user user 413181 sep  9 15:36 session1.png
drwxr-xr-x 2 user user  4096 jui 20 13:52 tmp/
drwxr-xr-x 2 user user  4096 jui 20 13:52 unix/
[local@home/user] █
```

L'interpréteur **bash** en action

# Exercices I.1

Testez quelques commandes simples

1. Affichez la date courante avec la commande `date`
2. Affichez le calendrier de l'année courante avec `cal 2008`
3. Que fait la commande `cal 9 2008` ? la commande `cal` ?
4. Testez la commande `echo` en évaluant `echo Bonjour tout le monde`
5. Que fait la commande `echo` sans argument ?
6. Essayez d'afficher le message `3 > 2 est vrai` à l'aide la commande `echo`
7. Recommencez en faisant précéder le caractère `>` du caractère `\` (*anti-slash*)
8. Recommencez en utilisant le caractère `'` pour délimiter la chaîne de caractères
9. Recommencez en utilisant le caractère `"` pour délimiter (d'une autre manière) la chaîne de caractères
10. Affichez le message précédent mais sans aller à la ligne (utilisez l'option `-n`)

# Edition des commandes (1)

L'édition d'une commande sous **bash** est possible pendant la saisie même de la commande, à l'aide de caractères spéciaux qui sont identiques à certains raccourcis clavier de **Emacs/XEmacs**. Lors de la saisie d'une commande, on est automatiquement en mode insertion, et on dispose des fonctionnalités suivantes :

## Déplacement du curseur

- ★ déplacement du curseur à gauche d'un caractère : flèche gauche ou `<ctrl-b>`
- ★ déplacement du curseur à droite d'un caractère : flèche droite ou `<ctrl-f>`
- ★ déplacement du curseur d'un mot à gauche : `<meta-b>`
- ★ déplacement du curseur d'un mot à droite : `<meta-f>`
- ★ déplacement du curseur en début de ligne : `<ctrl-a>`
- ★ déplacement du curseur en fin de ligne : `<ctrl-e>`

# Edition des commandes (2)

## Effacement de caractères

- ★ effacement du dernier caractère saisi (à gauche du curseur) : `<backspace>`
- ★ effacement du caractère courant (sous le curseur) : `<suppr>` ou `<ctrl-d>`
- ★ effacement du mot courant (à partir du curseur) : `<meta-d>`
- ★ effacement du texte du début de ligne jusqu'au curseur: `<ctrl-u>`
- ★ effacement de la ligne du curseur à la fin de la ligne : `<ctrl-k>`

## Insertion des caractères supprimés

Les caractères supprimés par les commandes précédentes peuvent être insérés à droite du curseur par `<ctrl-y>`

# Edition des commandes (3)

## Historique des commandes

Le shell **bash** gère un historique des commandes saisies et on peut se déplacer dans cet historique

- ★ ré-édition de la commande précédent la commande courante dans l'historique :  
flèche haut ou `<ctrl-p>`
- ★ ré-édition de la commande suivant la commande courante dans l'historique :  
flèche bas ou `<ctrl-n>`

## Complétion

Le shell **bash** comprend un mécanisme de complétion pour les noms de commandes comme pour les noms de fichiers

- ★ complétion simple : `<tab>` à la suite des caractères déjà saisis
- ★ complétion multiple : `<tab><tab>` à la suite des caractères déjà saisis

# Exercices I.2

Testez à nouveau et séquentiellement les commandes suivantes, mais cette fois en utilisant **intensivement** l'édition en ligne des commandes :

1. Affichez la date courante avec la commande `date`
2. Affichez le calendrier de l'année courante avec `cal 2008`
3. Que fait la commande `cal 9 2008` ? la commande `cal` ?
4. Testez la commande `echo` en évaluant `echo Bonjour tout le monde`
5. Que fait la commande `echo` sans argument ?
6. Essayez d'afficher le message `3 > 2 est vrai` à l'aide la commande `echo`
7. Recommencez en faisant précéder le caractère `>` du caractère `\` (*anti-slash*)
8. Recommencez en utilisant le caractère `'` pour délimiter la chaîne de caractères
9. Recommencez en utilisant le caractère `"` pour délimiter (d'une autre manière) la chaîne de caractères
10. Affichez le message précédent mais sans aller à la ligne (utilisez l'option `-n`)

# Aide sur les commandes

**La plupart des commandes de Linux proposent au moins un paramètre d'aide**

★ `-h`

(`-` est surtout utilisé pour introduire des options en 1 caractère)

★ `--help`

(`--` est toujours utilisé pour introduire l'option "longue" correspondante, qui rend les scripts plus faciles à comprendre)

Les commandes affichent souvent un court résumé des options disponibles quand vous utilisez un argument invalide.

# Pages de manuel

```
man <mot_clé>
```

Affiche une ou plusieurs pages de manuel pour <mot\_clé>

★ `man man`

La plupart des pages de manuel disponibles concernent des commandes Unix, mais aussi des fonctions, entêtes ou structures de données de bibliothèques C, ou même des fichiers de configuration du système

★ `man stdio.h`

★ `man fstab` (pour `/etc/fstab`)

Les pages de manuel sont recherchées dans les répertoires spécifiées par la variable d'environnement `MANPATH`

# Exercices I.3

1. A l'aide d'internet, trouvez le nom de la commande Linux qui permet de compresser des fichiers en utilisant l'algorithme de compression de Burrows Wheeler
2. Affichez la page de manuel de cette commande
3. Trouvez le numéro de la version de cette commande actuellement installée sur votre ordinateur

# Recherches sur Internet (1)

## Résolution de problèmes

- ★ La plupart des archives des forums et des listes de discussion sont publiques, et indexées très régulièrement par **Google**
- ★ Si vous recherchez la cause d'un message d'erreur, copiez-le mot à mot dans le formulaire de recherche, entouré par des guillemets. Il est très probable que quelqu'un d'autre aura déjà rencontré le même problème
- ★ N'oubliez pas d'utiliser Google Groups: <http://groups.google.com/>  
Ce site indexe plus de 20 années de messages de groupes de discussion

# Recherches sur Internet (2)

## Recherche de documentation

- ★ Recherchez `<outil> OR <outil> page` pour trouver la page d'accueil de l'outil ou du projet et ensuite trouver les plus récentes ressources de documentation
- ★ Recherchez `<outil> documentation or <outil> manual` (en anglais) dans votre moteur de recherche préféré

## Recherche de documentation générique

- ★ Wikipedia: <http://fr.wikipedia.org>  
De nombreuses et utiles définitions en informatique. Une vraie encyclopédie. Ouverte aux contributions de chacun

# Exercices I.4

Testez l'aide disponible sur les commandes en effectuant les opérations suivantes :

1. Affichez l'aide de la commande `cal`
2. Affichez le calendrier de l'année 2000
3. Affichez la page de manuel de la commande `date`
4. Utilisez la commande `touch` pour créer un fichier vide de nom `vide` dans votre répertoire de travail `linux`

## **II. Principes et commandes de base**

- \* Système de fichiers**
- \* Commandes de base**
- \* Liens symboliques et physiques**
- \* Droits d'accès**
- \* Impression sous Linux**

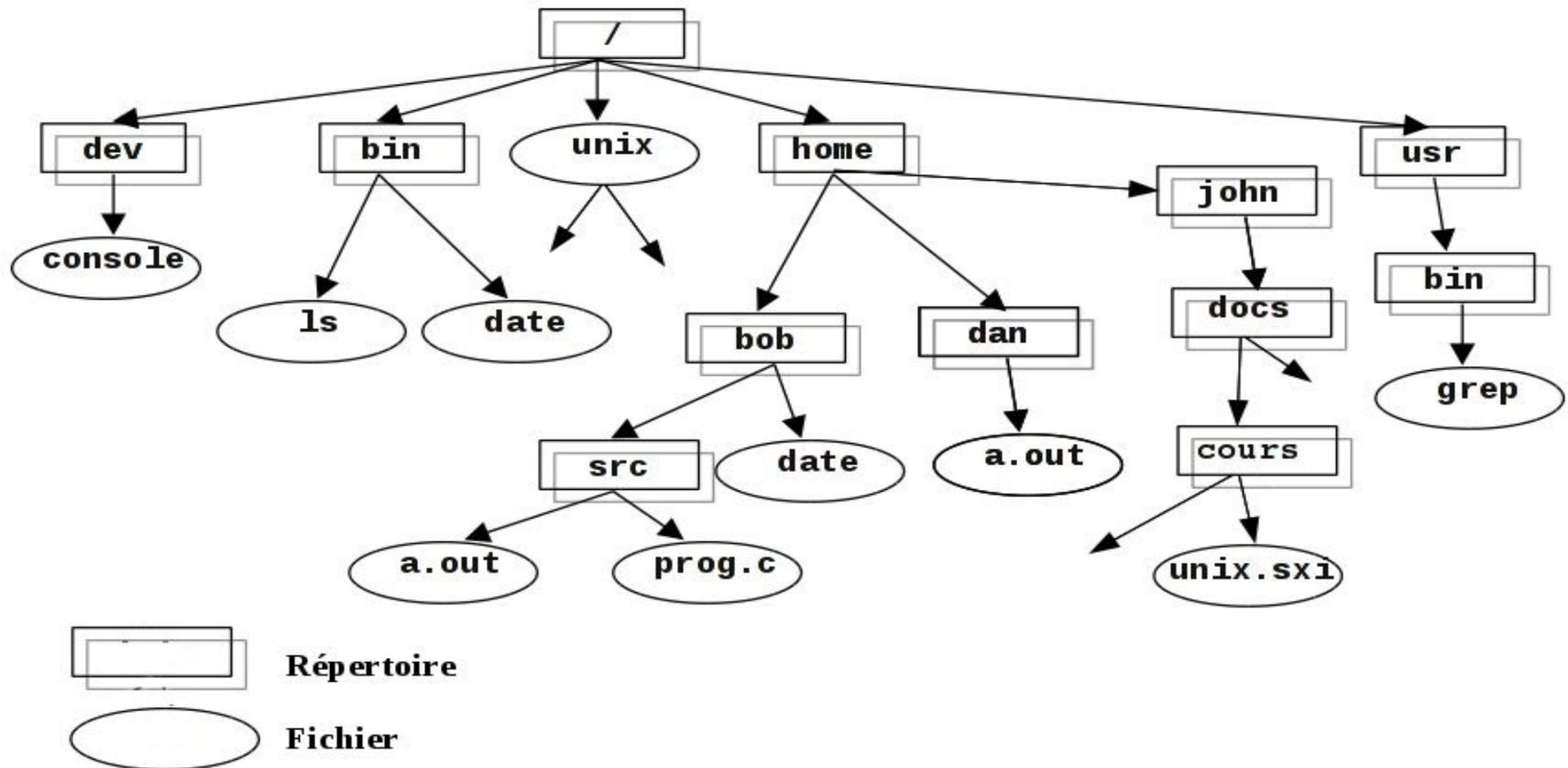
# Fichiers et répertoires

## Presque tout dans Unix / Linux est un fichier

- ★ Même les répertoires sont des fichiers, contenant une liste de fichiers et de répertoires.
- ★ Depuis le début d'Unix, aucune limitation majeure quant à la longueur d'un nom de fichier. Tout caractère (les espaces en particulier) peut être utilisé dans le nom, et les extensions sont facultatives. Les différences de majuscules ou de minuscules constituent des fichiers distincts.
- ★ Un *chemin* («path») est une séquence de répertoires imbriqués, avec un fichier ou un répertoire à la fin, séparés par le caractère /
  - ★ Chemin relatif: `docs/cours/unix.html`  
Relatif au répertoire courant
  - ★ Chemin absolu: `/home/john/docs/cours/unix.html`  
Chemin depuis le répertoire racine du système (/)

# Systeme de fichiers Linux (1)

## Structure arborescente



# Systeme de fichiers Linux (2)

**Rien d'imposé par le système. Peut varier d'un système à l'autre, même entre deux installations de Linux!**

/	Répertoire racine
/bin/	Commandes de base du système
/boot/	Images, initrd et fichiers de configuration du noyau
/dev/	Fichiers représentant des périphériques
/dev/hda	premier disque dur IDE
/etc/	Fichiers de configuration du système
/home/	Répertoires utilisateurs
/lib/	Bibliothèques de base du système (partagées)

# Systeme de fichiers Linux (3)

<code>/lost+found</code>	Fichiers détériorés que le système a essayé de récupérer
<code>/mnt/</code>	Systemes de fichiers montés <code>/mnt/usbdisk/</code> , <code>/mnt/windows/</code> ...
<code>/opt/</code>	Outils spécifiques installés par l'administrateur. Souvent remplacé par <code>/usr/local/</code>
<code>/proc/</code>	Accès aux informations du système <code>/proc/cpuinfo</code> , <code>/proc/version</code> ...
<code>/root/</code>	Répertoire utilisateur de l'administrateur
<code>/sbin/</code>	Commandes réservées à l'administrateur
<code>/sys/</code>	Contrôle du système et des périphériques (fréquence du processeur, gestion de l'alimentation des périphériques, etc.)

# Systeme de fichiers Linux (4)

`/tmp/`

Fichiers temporaires

`/usr/`

Programmes utilisateurs ordinaires, non essentiels au système : `/usr/bin/`, `/usr/lib/`,...

Outils spécifiques installés par l'administrateur.

(alternative : `/opt/`)

`/var/`

Données utilisées par le système ou ses serveurs

`/var/log/`, `/var/spool/mail` (courrier entrant),

`/var/spool/lpd` (travaux d'impression)...

# Répertoires spéciaux (1)

## Le répertoire relatif `./`

- ★ Le répertoire courant. Utile pour les commandes qui ont un répertoire comme argument. Également utile parfois pour lancer des commandes dans le répertoire courant (voir plus tard)
- ★ Ainsi `./lisezmoi.txt` et `lisezmoi.txt` sont équivalents

## Le répertoire relatif `../`

- ★ Le répertoire parent. Fait partie toujours partie du répertoire `.` (voir `ls -a`). Unique référence au répertoire parent.
- ★ Utilisation la plus courante:  
`cd ..`

# Répertoires spéciaux (2)

## Le répertoire absolu `~/`

- ★ Pas vraiment un répertoire spécial. Les interpréteurs de commande le remplacent juste par le répertoire utilisateur de l'utilisateur courant
- ★ Ne peut pas être utilisé dans la plupart des programmes, car il n'est pas un vrai répertoire (voir la variable `HOME`)

## Le répertoire absolu `~bob/`

- ★ De façon analogue, remplacé par les shells par le répertoire utilisateur de l'utilisateur `bob`

# Chemins absolus et relatifs

## Un fichier peut être désigné de façon absolue ou relative

- ★ Un chemin absolu identifie de manière unique un fichier ou un répertoire dans l'arborescence :

`/home/bob/src/a.out`      `/usr/bin/grep`

- ★ Un chemin relatif identifie un fichier ou un répertoire en fonction du répertoire courant (on est sous `~bob`) :

`src/a.out`      `../dan/a.out`

# La commande `ls`

**Affiche la liste des fichiers dans le répertoire courant, en ordre alphanumérique, sauf ceux qui commencent par le caractère “.”**

- ★ `ls -a` («all»: tous)  
Affiche tous les fichiers (y compris les fichiers `.*`)
- ★ `ls -l` (long)  
Affichage en format long (type, date, taille, propriétaire, permissions)
- ★ `ls -t` (temps)  
Affiche les fichiers les plus récents en premier
- ★ `ls -S` (“size”: taille)  
Affiche les fichiers les gros en premier
- ★ `ls -r` («reverse»: inversé)  
Affiche en ordre inverse
- ★ `ls -ltr` (les options peuvent être combinées)  
Format long, les fichiers les plus récents à la fin

**La commande `ls` possède une cinquantaine d'options !**

# Exercices II.1

Testez la commande `ls` en affichant, depuis votre répertoire personnel initial (*home directory*), la liste de **tous** vos fichiers et sous-répertoires :

1. de cinq façons différentes (faites varier le paramètre de `ls`)
2. sous un format *condensé*
3. sous un format *long* (donnant le propriétaire, les permissions, la taille, ...)
4. en affichant aussi les fichiers cachés (dont le nom commence par un point)
5. avec un format long et en affichant les fichiers cachés, mais dans l'ordre alphabétique inverse
6. avec un format long et en affichant les fichiers cachés, mais du plus récent au plus ancien
7. avec un format long et en affichant les fichiers cachés, mais du plus ancien au plus récent

# Les commandes **cd** et **pwd**

- ★ **cd rep** (*change directory*)  
Fait de **rep** le nouveau répertoire courant
- ★ **cd**  
Sans argument, fait du répertoire utilisateur le nouveau répertoire courant
- ★ **cd -** (le caractère tiret : '-')  
Fait du répertoire précédent le répertoire courant (le dernier répertoire depuis lequel on a fait cd) le nouveau répertoire courant
- ★ **pwd** (*print working directory*)  
Affiche le chemin absolu du répertoire courant

# Les commandes `mv` et `cp`

- ★ `mv ancien_nom nouveau_nom` (*move*)  
Change le nom du fichier ou du répertoire donné
- ★ `mv -i` (*interactive*)  
Si le fichier existe déjà, demander confirmation à l'utilisateur
- ★ `cp fichier_orig fichier_dest`  
Crée une copie d'un fichier d'origine
- ★ `cp fich1 fich2 fich3 ... rep`  
Copie tous les fichiers vers le répertoire de destination (dernier argument)
- ★ `cp -i` (*interactive*)  
Si le fichier de destination existe déjà, demander confirmation à l'utilisateur
- ★ `cp -r rep_orig rep_dest` (*recursive*)  
Copie du répertoire tout entier

# Les commandes `rm` et `rmdir`

- ★ `rm fich1 fich2 fich3 ...` (*remove*)  
Supprime les fichiers donnés
- ★ `rm -i fich1 fich2 fich3 ...` (*interactive*)  
Demande toujours à l'utilisateur de confirmer les suppressions
- ★ `rm -r rep1 rep2 rep3 ...` (*recursive*)  
Suppression des répertoires donnés et de tout leur contenu
- ★ `rm -f fich1 fich2 fich3 ...` (*force*)  
Suppression des fichiers donnés sans message ni question
- ★ `rmdir rep1 rep2 rep3 ...` (*remove directory*)  
Suppression des répertoires donnés seulement s'ils sont vides

# La commande `mkdir`

★ `mkdir rep` (*make directory*)

Fabrique `rep`, un nouveau répertoire, dans le courant. Si `rep` existe déjà, provoque une erreur

★ `mkdir rep1 rep2 ... repN`

Fabrique séquentiellement plusieurs répertoires

★ `mkdir rep1/rep2`

Il faut que le répertoire `rep1` existe déjà, sinon provoque une erreur

★ `mkdir -p rep1/rep2/.../repN` (*parent*)

Fabrique les répertoires `rep1 rep2 ... repN-1` si besoin, et enfin le répertoire `repN`

# Exercices II.2

Testez les commandes élémentaires relatives aux fichiers et des répertoires :

1. Créez un répertoire `system` sous votre répertoire de travail `linux`, puis un répertoire `tp1` sous `system`
2. Effacez le répertoire `system` avec la commande `rmdir`. Que constatez-vous ?
3. Après avoir effacé les répertoires `tp1` et `system`, créez à l'aide d'une seule commande les répertoires `system`, `system/tp1`, `system/tp2`
4. Renommez le répertoire `system` en `test`
5. Copiez un fichier de votre choix du répertoire `/bin` dans le répertoire `test/tp1` :
  - a) depuis le répertoire `/bin`
  - b) depuis le répertoire `test/tp1`
  - c) depuis votre *homedir*, en utilisant des chemins absolus
  - d) depuis votre *homedir*, en utilisant des chemins relatifs
6. Effacez à l'aide d'une seule commande les répertoires `test/tp1` et `test/tp2`

# Substitutions des noms de fichiers

★ `ls *txt`

L'interpréteur remplace d'abord `*txt` par tous les noms de fichiers et de répertoires finissant par `txt` (y compris `.txt`), sauf ceux commençant par `.` et enfin exécute la ligne de commande `ls`

★ `ls -d .*`

Affiche tous les fichiers et les répertoires commençant par `.`  
`-d` indique à `ls` de ne pas afficher le contenu des dossiers `.*`

★ `ls ?.log`

Affiche tous les fichiers dont le nom commence par **un** caractère et finit par `.log`

★ `rm -r log*`

Efface TOUS les fichiers et sous-répertoires du répertoire courant dont le nom commence par `log`

# Caractères génériques

## Plus généralement :

★ \* dénote 0, 1 ou plusieurs caractères

`*.c`                    `doc*unix.html`

★ ? dénote un (et exactement un) caractère quelconque

`?nix`                    `*.s??`

★ [...] dénote un caractère appartenant à un ensemble

`[Uu]nix`                `*.sx[ci]`

★ [^...] dénote un caractère n'appartenant pas à un ensemble

`.[^co]`                    `[^[ :upper: ]]*`

# Exercices II.3 (1)

Placez-vous dans le répertoire `/bin` puis affichez la liste de tous les fichiers :

1. dont le nom commence par la lettre `t`
2. dont le nom termine par la lettre `h`
3. dont le nom a exactement trois lettres
4. dont le nom contient la sous-chaîne `un`
5. dont le nom contient au moins un chiffre

Créez dans votre répertoire de travail `linux` (avec la commande `touch`) les fichiers de noms exact `fich1`, `fich2`, `fich11`, `fich12`, `fich1a`, `ficha1`, `.fich1`, `.fich2`, `afich` et `toto`, puis affichez :

6. les fichiers dont le nom commence par `fich`
7. les fichiers dont le nom commence par `fich` suivi d'un seul caractère
8. les fichiers dont le nom commence par `fich` suivi d'un seul caractère qui est un chiffre
9. les fichiers dont le nom commence par le caractère `.` (point)
10. les fichiers dont le nom ne commence pas par `f`

# Exercices II.3 (2)

Placez-vous dans le répertoire `/etc`, puis listez tous les fichiers et les noms de répertoires :

0. de configuration (i.e. dont le nom termine par le suffixe `.conf`)
1. dont le nom commence par la chaîne `cron`
2. dont le nom ne termine pas par le caractère `d`
3. Placez-vous dans votre répertoire `linux`, videz-le de son contenu, puis placez-y deux fichiers de noms `toto` et `titi` contenant respectivement les lignes `toto` et `titi` (utilisez un éditeur)
4. Devinez ce que va faire la commande `cp t*` et vérifiez en l'évaluant.
5. Construisez le répertoire `tata` (toujours sous le répertoire `linux`) puis devinez ce que va faire très exactement la commande `mv t* tata`. Vérifiez en l'évaluant.
6. Déplacez tous les fichiers placés sous `tata` dans `linux`
7. Fabriquez un fichier (avec la commande `touch`) de nom `-i`, puis effacez-le
8. Fabriquez un fichier de nom `t*a`, puis effacez-le (et n'effacez que lui !)
9. Fabriquez un fichier de nom `toto<espace>titi` puis effacez-le (et n'effacez que lui !)

# Afficher le contenu de fichiers

## Plusieurs façons d'afficher le contenu de fichiers

★ `cat fich1 fich2 fich3 ...` (concaténer)

Met bout à bout et affiche le contenu des fichiers donnés

★ `more fich1 fich2 ...` (plus de détails)

A chaque page, demande à l'utilisateur d'appuyer sur une touche pour continuer. Peut aussi aller directement à la première apparition d'un mot-clé (commande "/" suivie du mot-clé)

★ `less fich1 fich2 fich3 ...` (moins)

Un `more` plus puissant (le nom est un jeu de mot...) !

Ne lit pas le fichier entier avant de commencer à afficher

Permet de remonter en arrière dans le fichier (commande "?")

# Les commandes **head**, **tail** et **cat**

★ **head [-n N] fichier** (*tête*)

Affiche les **N** premières lignes (ou 10 par défaut) du fichier donné.  
N'a pas besoin d'ouvrir le fichier en entier pour le faire

★ **tail [-n N] fichier** (*queue*)

Affiche les **N** dernières lignes (ou 10 par défaut) du fichier donné  
Ne charge pas tout le fichier en mémoire. Très utile pour les gros fichiers.

★ **tail -f fichier** (*follow*)

Affiche les 10 dernières lignes du fichier donné et continue à afficher les nouvelles lignes au fur et à mesure qu'elles sont rajoutées en fin de fichier. Très pratique pour suivre les rajouts à un fichier de journal ("log")

★ **cat [-n] fichier**

Affiche le contenu du fichier donné. Avec l'option **-n** numérote chaque ligne affichée de **1** à **n** (le nombre de lignes)

# La commande `wc`

★ `wc fichier` (*word count*)

Affiche le nombre de lignes, de mots et de caractères du fichier `fichier`

★ `wc -c fichier`

Affiche le nombre d'octets contenus dans le fichier `fichier`

★ `wc -m fichier`

Affiche le nombre de caractères contenus dans le fichier `fichier`

★ `wc -l fichier`

Affiche le nombre de lignes contenues dans le fichier `fichier`

★ `wc -w fichier`

Affiche le nombre de mots contenus dans le fichier `fichier`

# La commande `sort`

- ★ `sort fichier`

Trie les lignes du fichier selon l'ordre des caractères et les affiche.

- ★ `sort -r fichier` (*reverse*)

Idem, mais en ordre inverse

- ★ `sort -ru fichier` (*unique*)

Idem, mais ne produit qu'une seule fois les lignes identiques.

- ★ Plus de possibilités seront abordées plus tard

# Exercices II.4

0. Affichez la première ligne du fichier `nanorc` de le répertoire `/etc`
1. Affichez les 10 dernières lignes du fichier `nanorc` de le répertoire `/etc`
2. Visualisez le fichier `/etc/fstab` successivement avec les commandes `cat`, `more` et `less`
3. Visualisez le fichier `/etc/fstab` avec la commande `less` et sautez directement aux lignes contenant la chaîne de caractères `ANSI` (en utilisant `/` et `n`)
4. Testez la commande `wc` et ses différentes options sur le fichier `/etc/fstab`
5. Créez un fichier de texte (dans votre répertoire `linux`) contenant quelques caractères sur deux lignes mais ne terminant pas par un *newline* (ligne vide en fin de fichier) et comptez le nombre de caractères du fichier avec la commande `wc`
6. Recommencez en ajoutant cette fois un *newline* en fin de fichier
7. Testez la commande `sort` sur plusieurs fichiers comme `/usr/lib/helix/README` ou `/var/www/icons/README`
8. Testez les commandes `cat`, `wc` et `sort` sans arguments et en saisissant des lignes interactivement (tapez `<ctrl-d>` pour terminer)

# La commande `grep` (1)

★ `grep motif fichiers`

Parcourt les fichiers donnés et affiche les lignes qui correspondent au motif spécifié.

★ `grep erreur *.log`

Affiche toutes les lignes contenant `erreur` dans les fichiers `*.log`

★ `grep -i erreur *.log`

Idem, mais indifférent aux majuscules et minuscules

★ `grep -ri erreur .`

Idem, mais récursivement dans `.` et ses sous-répertoires

★ `grep -v info *.log`

Affiche toutes les lignes des fichiers, sauf celles qui contiennent `info`

# La commande `grep` (2)

Le motif (le premier paramètre) de `grep` est une expression régulière qui utilise des caractères semblables aux caractères génériques du *shell* mais dont l'interprétation diffère

★ `grep 'es*ai' fichier`

Affiche les lignes qui contiennent `e` suivi d'un ou plusieurs `s` et terminant par `ai`

★ `grep 'es.ai' fichier`

Affiche les lignes qui contiennent `es` suivi d'un caractère quelconque et terminant par `ai`

★ `grep '[0-9]' fichier`

Affiche les lignes qui contiennent au moins un caractère numérique

★ `grep '^Je' fichier`

Affiche les lignes qui commencent par `Je`

★ `grep '\.$' fichier`

Affiche les lignes qui terminent par le caractère `.`

★ `grep -v '[;:0-9]' fichier`

Affiche les lignes qui ne contiennent ni chiffres, ni `;` ou `:`

# La commande `find`

## Plus facile à expliquer par quelques exemples

★ `find . -name "*.pdf" [-print]`

Recherche tous les fichiers `*.pdf` dans le répertoire courant (`.`) et ses sous-répertoires. Vous devez utiliser les guillemets pour empêcher le shell de substituer le caractère `*`

★ `find docs -name "*.pdf" -exec xpdf {} ';'`

Recherche tous les fichiers `*.pdf` dans le répertoire `docs` et les affiche l'un après l'autre

★ De nombreuses possibilités existent et il faut lire la page de manuel ! Cependant, les 2 exemples ci-dessus couvrent la plupart des besoins

# La commande `locate`

Outil de recherche à base d'expressions régulières, essentiellement pour trouver l'emplacement de fichiers ou de répertoires

★ `locate clef`

Affiche tous les fichiers sur votre système contenant `clef` dans leur nom

★ `locate "*.pdf"`

Affiche tous les fichiers `*.pdf` existant sur votre système.

★ `locate "/home/frigo/*mousse*"`

Affiche tous les fichiers `*mousse*` dans le répertoire indiqué (chemin absolu)

`locate` est très rapide grâce à l'indexation de tous les fichiers dans une base de données dédiée, qui est mise à jour régulièrement.

`locate` est beaucoup moins utile que `find` qui est beaucoup plus adapté à la recherche fine de fichiers, et qui permet en outre d'effectuer des actions sur les fichiers trouvés.

# Exercices II.5

0. Affichez les lignes du fichier `/etc/gai.conf` qui contiennent le caractère `#`
1. Affichez les lignes du fichier `/etc/gai.conf` qui commencent par le caractère `#`
2. Affichez les lignes du fichier `/etc/gai.conf` qui contiennent des nombres séparés par des points (comme par exemple 3.1 ou 10.2.1)
3. Affichez les lignes du fichier `/etc/mailcap` qui contiennent le caractère `$`
4. Affichez les lignes du fichier `/etc/profile` qui terminent par le caractère `\`
5. Affichez tous les fichiers réguliers (les fichiers, et non pas les répertoires ou les liens) vous appartenant (i.e. se trouvant dans votre *homedir* ou récursivement dans un de ses sous-répertoires)
6. Affichez le nom de tous les répertoires (et seulement des répertoires) vous appartenant
7. Affichez tous les fichiers cachés (i.e. de nom `.*`) vous appartenant
8. Créez plusieurs fichiers d'extension `.bak` dans votre *homedir* et dans certains de ses sous-répertoires, oubliez leur emplacement, puis effacez-les tous à l'aide d'une seule commande
9. A l'aide de la commande `touch`, créez plusieurs fichiers vides dans votre *homedir* et dans certains de ses sous-répertoires, puis effacez-les tous à l'aide d'une seule commande

# Comparaison de fichiers

★ `cmp fichier1 fichier2`

Compare deux fichiers quelconques.

★ `diff fichier1 fichier2`

Affiche les différences entre deux fichiers, ou rien si les fichiers sont identiques.

★ `diff -r rep1/ rep2/`

Affiche les différences entre fichiers de même nom dans les deux répertoires.

★ Pour examiner en détail les différences, mieux vaut utiliser des outils graphiques

# Le programme **kompare**

Un autre outil pratique pour comparer des fichiers et fusionner leurs différences

```
File Difference Settings Help
Makefile
76 incdir-$(CONFIG_ARCH_CO285) := ebsa285
77 machine-$(CONFIG_ARCH_FTVPCI) := ftvpci
78 incdir-$(CONFIG_ARCH_FTVPCI) := nexuspai
79 machine-$(CONFIG_ARCH_TBOX) := tbox
80 machine-$(CONFIG_ARCH_SHARK) := shark
81 machine-$(CONFIG_ARCH_SA1100) := sa1100
82 ifeq ($(CONFIG_ARCH_SA1100),y)
83 # SA1111 DMA bug: we don't want the kernel to live in p
84 textaddr-$(CONFIG_SA1111) := 0xc0208000
85 endif
86 machine-$(CONFIG_ARCH_PXA) := pxa
87 machine-$(CONFIG_ARCH_L7200) := l7200
88 machine-$(CONFIG_ARCH_INTEGRATOR) := integrator
89 machine-$(CONFIG_ARCH_CAMELOT) := epxa10db
90 textaddr-$(CONFIG_ARCH_CLPS711X) := 0xc0028000
91 machine-$(CONFIG_ARCH_CLPS711X) := clps711x
92 textaddr-$(CONFIG_ARCH_FORTUNET) := 0xc0008000
93 machine-$(CONFIG_ARCH_IOP3XX) := iop3xx
94 machine-$(CONFIG_ARCH_ADIFCC) := adifcc
95 machine-$(CONFIG_ARCH_OMAP) := omap
96 machine-$(CONFIG_ARCH_S3C2410) := s3c2410
97 machine-$(CONFIG_ARCH_LH7A40X) := lh7a40x
98 machine-$(CONFIG_ARCH_VERSATILE_PB) := versatile
99
100 TEXTADDR := $(textaddr-y)
101 ifeq ($(incdir-y),)
102 incdir-y := $(machine-y)
103 endif
104 INCDIR := arch-$(incdir-y)
105
106 export TEXTADDR GZFLAGS
107

Makefile
75 incdir-$(CONFIG_FOOTBRIDGE) := ebsa285
75 textaddr-$(CONFIG_ARCH_CO285) := 0x60008000
76 machine-$(CONFIG_ARCH_CO285) := footbridge
77 incdir-$(CONFIG_ARCH_CO285) := ebsa285
78 machine-$(CONFIG_ARCH_SHARK) := shark
79 machine-$(CONFIG_ARCH_SA1100) := sa1100
80 ifeq ($(CONFIG_ARCH_SA1100),y)
82 # SA1111 DMA bug: we don't want the kernel to live in p
83 textaddr-$(CONFIG_SA1111) := 0xc0208000
84 endif
85 machine-$(CONFIG_ARCH_PXA) := pxa
86 machine-$(CONFIG_ARCH_L7200) := l7200
87 machine-$(CONFIG_ARCH_INTEGRATOR) := integrator
88 machine-$(CONFIG_ARCH_CAMELOT) := epxa10db
89 textaddr-$(CONFIG_ARCH_CLPS711X) := 0xc0028000
89 machine-$(CONFIG_ARCH_CLPS711X) := clps711x
90 textaddr-$(CONFIG_ARCH_FORTUNET) := 0xc0008000
91 machine-$(CONFIG_ARCH_IOP3XX) := iop3xx
92 machine-$(CONFIG_ARCH_IXP4XX) := ixp4xx
93 machine-$(CONFIG_ARCH_OMAP) := omap
94 machine-$(CONFIG_ARCH_S3C2410) := s3c2410
95 machine-$(CONFIG_ARCH_LH7A40X) := lh7a40x
96 machine-$(CONFIG_ARCH_VERSATILE_PB) := versatile
97
98 ifeq ($(CONFIG_ARCH_EBSA110),y)
99 # This is what happens if you forget the IOCS16 line.
100 # PCMCIA cards stop working.
101 CFLAGS_3c589_cs.o := -DISA_SIXTEEN_BIT_PERIPHERAL
102 export CFLAGS_3c589_cs.o
103 endif
104
105 TEXTADDR := $(textaddr-y)
```

Comparing file file:/data/mike/handhelds/stock\_kernel/linux-2.6....data/mike/handhelds/stock\_kernel/linux-2.6.8.1/arch/arm/Makefile 1 of 11 differences, 0 applied 1 of 1 file

# Mesure de la taille des fichiers

★ `du -h fichier` (disk usage)

`-h`: affiche la taille du fichier donné, sous forme lisible par un humain: K (kilo-octets), M (mega-octets) or G (giga-octets).  
Sinon `du` rend le nombre brut de blocs occupés par le fichier sur le disque (difficile à lire).

★ `du -sh rep`

`-s`: rend la somme des tailles de tous les fichiers dans le répertoire donné

# Mesure de l'espace disque

## ★ `df -h rep`

Affiche des informations sur l'espace disque utilisé et disponible dans le système de fichiers qui contient le répertoire donné.

De même, l'option `-h` n'existe que dans GNU `df`.

## ★ Exemple :

```
> df -h .
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hda5	9.2G	7.1G	1.8G	81%	/

## ★ `df -h`

Affiche les informations d'espace disque pour tous les systèmes de fichiers disponibles sur le système. Quand des erreurs surviennent, utile pour vérifier si des systèmes de fichiers pleins.

# Exercices II.6

0. Copiez le fichier `/etc/fstab` dans votre répertoire de travail `linux`
1. Editez le fichier `fstab`, modifiez-en quelques lignes, ajoutez-lui une ligne et supprimez-lui une ligne
2. Testez les commandes `cmp` et `diff` sur les fichiers `/etc/fstab` et `linux/fstab`
3. Visualisez les différences entre les fichiers `/etc/fstab` et `linux/fstab` à l'aide de la commande `kompare`, que vous lancerez successivement :
  - a) par le menu  **Applications** (sous-menu Développement)
  - b) par la commande `kompare` tapée dans un terminal
  - c) par le lanceur de commande (menu **K** puis action Exécuter une commande)
4. Affichez la taille en méga-octets occupée par votre `homedir`
5. Affichez la taille en kilo-octets du fichier `/etc/fstab`
6. Affichez les informations d'espace disque pour tous les systèmes de fichiers disponibles sur votre système

# Compression

**Très utile pour compacter de gros fichiers et économiser de la place**

★ `[un]compress fichier`

Utilitaire de compression traditionnel d'Unix. Crée des fichiers `.Z`  
Seulement gardé pour raisons de compatibilité. Performance moyenne.

★ `g[un]zip fichier`

Utilitaire de compression GNU zip. Crée des fichiers `.gz`  
Assez bonne performance (un peu meilleur que Zip)

★ `b[un]zip2 fichier`

Le plus récent et le plus performant des utilitaires de compression. Crée des fichiers `.bz2` . La plupart du temps 20-25% meilleur que `gzip`.  
Utilisez celui-ci ! Maintenant disponible sur tous les systèmes Unix.

# Archivage (1)

**Utile pour sauvegarder ou publier un ensemble de fichiers**

★ **tar** : à l'origine "tape archive" ("archive sur bande")

★ Création d'une archive:

**tar cvf archive fichiers-ou-répertoires**

**c** (créer) : pour fabriquer l'archive

**v** (verbeux) : utile pour suivre la progression de l'archivage

**f** (fichier) : archive créée dans un fichier (et non sur une bande)

★ Exemple :

```
tar cvf /backup/home.tar /home
```

```
bzip2 /backup/home.tar
```

# Archivage (2)

- ★ Afficher le contenu d'une archive ou vérifier son intégrité:  
`tar tvf archive`
- ★ Extraire tous les fichiers d'une archive:  
`tar xvf archive`
- ★ Extraire seulement quelques fichiers d'une archive:  
`tar xvf archive fichiers-ou-répertoires`  
Les fichiers ou répertoires sont donnés avec un chemin relatif au répertoire racine de l'archive.

# Options supplémentaires GNU tar

GNU tar sous GNU / Linux

Permet de compresser et décompresser des archives au vol. Utile pour éviter de créer d'énormes fichiers intermédiaires. Plus facile à faire qu'en combinant tar et gip/bzip2

★ j : [dé]compresse au vol avec bzip2

★ z : [dé]compresse au vol avec gzip

★ Exemples

★ tar jcvf projet.tar.bz2 projet/

★ tar ztf projet.tar.gz

# Exercices II.7

0. Ouvrez votre navigateur et allez à l'URL <http://tldp.org/LDP/abs/>
1. Téléchargez l'archive `abs-guide.html.tar.gz` (cliquer sur *Advanced Bash Scripting Guide*)
2. Créez le répertoire `unix/abs` et déplacez l'archive dans ce répertoire
3. Visualisez le contenu de l'archive sans la décompresser ni l'extraire
4. Décompressez et extrayez l'archive en une seule commande
5. Créez un répertoire de nom votre *login*
6. Copiez sous ce répertoire tous les fichiers et sous-répertoires cachés de votre *homedir*
7. Fabriquez une archive compressée de ce répertoire (et non **pas** seulement du **contenu** de ce répertoire)
8. Echangez-le par mail avec celui d'un autre étudiant
9. Sauvez, décompressez et extrayez dans votre répertoire `tmp` l'archive que vous avez reçue

**ATTENTION** : ne décompressez **jamais** une archive, en particulier dans votre *homedir*, sans avoir auparavant listé son contenu, et vérifié que son contenu était bien dans un répertoire

# Liens symboliques

**Un lien symbolique est un fichier spécial qui est juste une référence au nom d'un autre (fichier ou répertoire)**

- ★ Utile pour simplifier et réduire l'utilisation du disque quand deux fichiers ont le même contenu.
- ★ Exemple:  
`/usr/local/bin/java -> /usr/jdk1.5.0_03/bin/java`
- ★ Comment distinguer les liens symboliques:
  - ★ `ls -l` affiche `->` et le fichier référencé par le lien
  - ★ GNU `ls` affiche les liens avec une couleur différente

# Création de liens symboliques

- ★ Pour créer un lien symbolique (même ordre que dans `cp`) :

```
ln -s nom_fichier nom_lien
```

- ★ Pour créer un lien vers un fichier dans un autre répertoire, avec le même nom :

```
ln -s ../LISEZ_MOI.txt
```

- ★ Pour créer plusieurs liens d'un coup dans un dossier donné :

```
ln -s fich1 fich2 fich3 ... rep
```

- ★ Pour supprimer un lien :

```
rm nom_lien
```

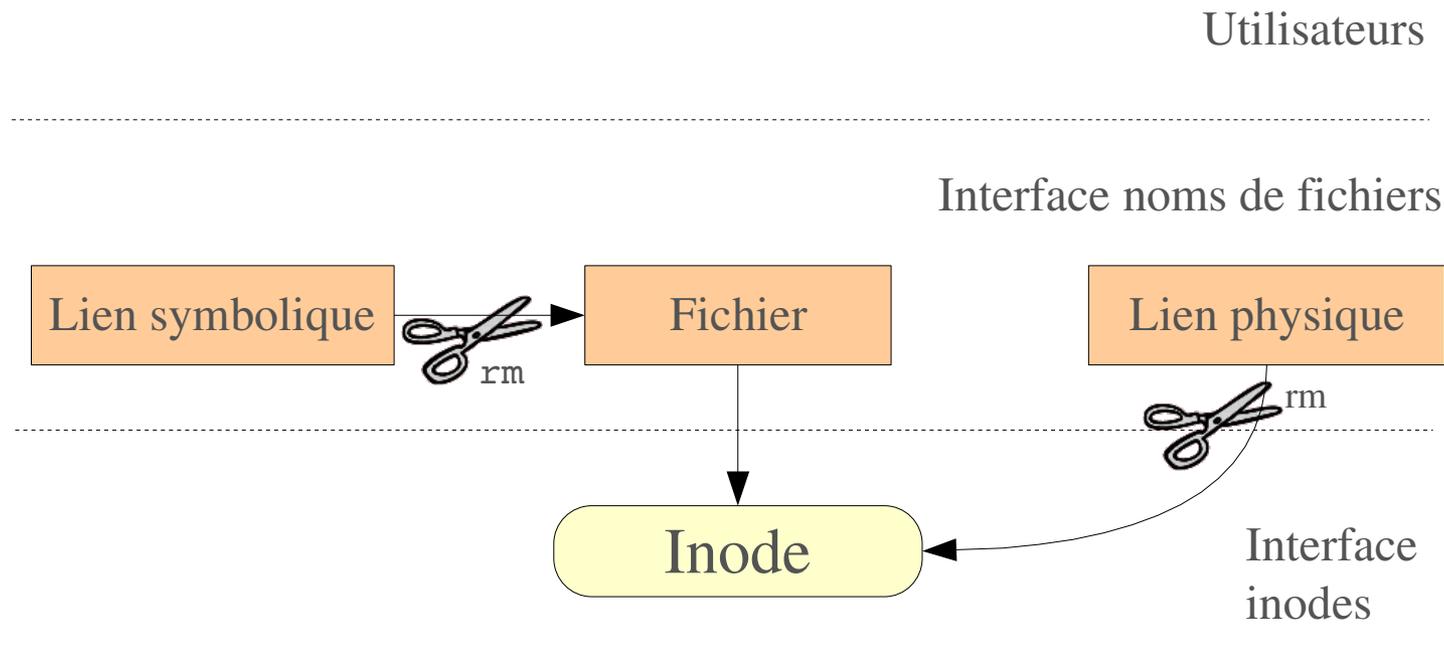
Bien-sûr, cela ne supprime pas le fichier référencé par le lien

# Liens physiques

- ★ Par défaut, `ln` crée des *liens physiques*
- ★ Un *lien physique* vers un fichier est un fichier ordinaire, avec exactement le même contenu physique
- ★ Bien qu'ils économisent toujours de la place, les liens physiques sont indiscernables des fichiers d'origine.
- ★ Si vous supprimez le fichier d'origine, cela n'affecte pas le contenu du lien physique.
- ★ Le contenu est supprimé quand il n'y a plus aucun fichier (lien physique) qui y fait référence.

# Noms de fichiers et inodes

## Pour mieux comprendre les liens symboliques et physiques



# Exercices II.8

0. Créez un fichier de texte `linux/lien` contenant quelques lignes
1. Créez un lien symbolique de `linux/lien` vers `linux/lien.sym`
2. Créez un lien physique de `linux/lien` vers `linux/lien.phy`
3. Vérifiez que le contenu des deux liens ainsi créés est bien identique à celui de `linux/lien`
4. Renommez le fichier `linux/lien` en `linux/link`
5. Le contenu des deux liens est-il toujours le même ? Expliquez !
6. Créez un lien symbolique sur le fichier `/etc/fstab`
7. Vérifiez que le contenu du lien est correct
8. Recommencez en créant cette fois un lien physique
9. Recherchez tous les liens symboliques vous appartenant (i.e. se trouvant dans votre *homedir*, ou récursivement dans un de ses sous-répertoires)

# Droits d'accès aux fichiers

Utiliser `ls -l` pour consulter les droits d'accès

3 types de droits d'accès :

- ★ Accès en lecture (**r**)
- ★ Accès en écriture (**w**)
- ★ Droits d'exécution (**x**)

3 types de niveaux d'accès:

- ★ Utilisateur (**u**)  
pour le propriétaire du fichier
- ★ Groupe (**g**)  
tout fichier a un attribut "groupe",  
qui correspond à une liste  
d'utilisateurs
- ★ Autres (**o**)  
pour tous les autres (propriétaire et  
groupe exclus)

# Contraintes de droits d'accès

- ★ Pour les fichiers, **x** sans **r** est autorisé mais sans valeur. Vous devez pouvoir lire un fichier pour l'exécuter
- ★ Les répertoires requièrent à la fois les droits **r** et **x** : **x** pour entrer, **r** pour accéder au contenu
- ★ Vous ne pouvez pas renommer, supprimer ou copier des fichiers dans un répertoire si vous n'avez pas accès en écriture à ce répertoire
- ★ Si vous avez accès en écriture à un répertoire, vous POUVEZ supprimer un fichier même si vous ne disposez pas de droits d'écriture pour ce fichier (souvenez-vous qu'un répertoire est juste un fichier décrivant une liste de fichiers). Cela permet même de modifier un fichier (le supprimer et le recréer) même protégé en écriture

# Exemples de droits d'accès

★ `-rw-r--r--`

Lisible et modifiable pour le propriétaire, seulement lisible pour les autres.

★ `-rw-r-----`

Lisible et modifiable pour le propriétaire, seulement lisible pour les utilisateurs appartenant au groupe du fichier.

★ `drwx-----`

Répertoire seulement accessible par son propriétaire

★ `-----r-x`

Fichier exécutable seulement par les autres, mais ni par votre groupe ni par vous-même. Droits d'accès typique d'un piège !

# chmod: modifier les permissions

- ★ `chmod permissions fichiers`

2 formats possibles pour `permissions` : format en base 8 ou format symbolique

- ★ Format en base 8 : `chmod abc fichiers`

avec `a`, `b` et `c` valant  $r*4+w*2+x$  (`r`, `w`, `x` égaux à 0 ou 1)

Exemple : `chmod 644 fichier` (`rw` pour `u`, `r` pour `g` et `o`)

- ★ Format symbolique : plus facile à comprendre sur des exemples

`chmod go+r` ajouter le droit en lecture au groupe et aux autres

`chmod u-w` supprimer le droit d'écriture pour le propriétaire

`chmod a-x` (*all*) enlever les droits d'exécution à tous les utilisateurs

# Autres options de `chmod` (1)

```
chmod -R a+rX linux/
```

Rend `linux` et tout ce qu'il contient accessible à tout le monde!

- ★ **R** : applique les changements récursivement
- ★ **X** : **x**, mais seulement pour répertoires et fichiers déjà exécutables. Très pratique pour ouvrir récursivement l'accès à des répertoires, sans donner le droit d'exécution à tous les fichiers

# Autres options de `chmod` (2)

```
chmod a+t /tmp
```

- ★ `t` : (“sticky”: collant). Permission spéciale pour les répertoires, autorisant uniquement l’effacement d’un fichier par son propriétaire ou par celui du répertoire
- ★ Utile pour les répertoires accessibles en écriture par plusieurs utilisateurs, comme `/tmp`

# Exercices II.9

Placez-vous dans le répertoire `linux`, puis :

- ★ Créez le fichier vide `fichier`, et examinez ensuite ses permissions
- ★ Modifiez successivement les permissions du répertoire `linux` pour :
  - ★ supprimer tous les accès à tout le monde
  - ★ ajouter l'accès en lecture, écriture et traversée pour vous seul
  - ★ ajouter l'accès en lecture et traversée pour le groupe
- ★ Créez le répertoire `linux/tmp`, puis le fichier `linux/tmp/perm`, et positionnez successivement les permissions adéquates pour que vous puissiez :
  - ★ lire, modifier et supprimer `perm`
  - ★ lire, modifier mais pas supprimer `perm`
  - ★ lire, supprimer mais pas modifier `perm`
  - ★ lire mais ni modifier et ni supprimer `perm`
- ★ Créez le fichier `linux/tmp/share` et positionnez les permissions nécessaires pour qu'un utilisateur de votre groupe puisse lire, modifier mais pas supprimer `share`

# Impression sous Unix

- ★ Multi-utilisateurs, multi-travaux, multi-clients, multi-imprimantes :  
Sous Unix / Linux, les commandes d'impression n'impriment pas vraiment. Elles envoient des tâches à des queues d'impression, soit sur la machine locale, soit sur des serveurs d'impression ou sur des imprimantes réseau
- ★ Système indépendant de toute imprimante :  
Les serveurs d'impression n'acceptent que des travaux en PostScript ou en texte. Les pilotes d'imprimante sur le serveur se chargent de la conversion vers le format propre à chaque imprimante
- ★ Système robuste :  
Redémarrez un système, il continuera à imprimer les travaux en attente
- ★ L'impression est soit pilotée par l'application (comme OpenOffice) soit lancée par l'utilisateur (impression de fichier PostScript ou texte)



# Impression de fichiers PostScript

- ★ Variable d'environnement utile: `PRINTER`

Définit l'imprimante par défaut sur le système. Exemple:

```
export PRINTER=niv2a
```

- ★ `lpr [-Pimp] fichiers`

Envoie les fichiers à la file d'attente de `imp`, l'imprimante spécifiée. Les fichiers doivent être en format texte ou PostScript

- ★ `a2ps [-Pimp] fichiers`

Convertit de nombreux formats vers PostScript et l'envoie le résultat vers la queue spécifiée. Fonctionnalités utiles: plusieurs pages / feuille, numérotation des pages, cadre d'informations

- ★ `a2ps -o fichier.ps fichier`

Convertit `fichier` dans le fichier PostScript `fichier.ps`

# Contrôle de travaux d'impression

## ★ `lpq [-Pqueue]`

Affiche tous les travaux d'impression de la queue par défaut ou de la queue donnée

```
lp is not ready
Rank      Owner    Job      File(s)                Total Size
1st       asloane  84       troyens_windows_nsa.ps 60416 bytes
2nd       amoore   85       gw_bush_erreurs_irak.ps 65024000 bytes
```

## ★ `cancel numéro_tâche [queue]`

Retire la tâche spécifiée de la queue d'impression

★ Le contrôle des travaux d'impression est également possible depuis le Window Manager : `KjobViewer` sous KDE

# Fichiers PostScript et PDF

## Manipulation d'un fichier PostScript

- ★ Visualisation avec `kghostview`
- ★ Conversion possible en PDF avec `ps2pdf` :  
`ps2pdf document.ps [document.pdf]`
- ★ Impression directe avec `lpr`

## Manipulation d'un fichier PDF

- ★ Visualisation avec `acroread`, `kpdf`
- ★ Conversion possible en PostScript avec `pdf2ps`:  
`pdf2ps document.pdf [document.ps]`
- ★ Impression après conversion en PostScript

# Exercices II.10

1. Créez le fichier de texte `linux/texte.txt` qui contient plusieurs ligne de texte
2. Fabriquez le fichier `linux/texte.ps`, version PostScript du fichier `linux/texte.txt`, à l'aide la commande `a2ps`, et visualisez-le successivement avec les commandes `kpdf` et `acroread`
3. Copiez le fichier `/usr/share/doc/bash-3.1/loadables/cat.c` dans votre répertoire `linux`
4. Fabriquez le fichier `linux/cat.ps`, version PostScript du fichier `linux/cat.c`, à l'aide la commande `a2ps`, et visualisez-le successivement avec les commandes `kpdf` et `acroread`
5. Recommencez, mais en produisant cette fois un fichier PostScript à raison d'une page (de source ASCII) par page (PostScript) dans lequel les lignes de code `C` sont numérotés

# III. Redirections et processus

- ★ **Entrées/sorties et redirections**
- ★ **Les tuyaux**
- ★ **Contrôle de tâche et de processus**

# Descripteur de fichier

- ★ Les entrées/sorties des commandes sont repérées par des descripteurs de fichier ("*file descriptor*")
- ★ Un descripteur de fichier est un *entier* associé à un fichier spécial
- ★ On peut modifier (en les *redirigeant*) les descripteurs de fichier associés aux entrées/sorties de n'importe quelle commande
- ★ Les entrées/sorties *standards* ne sont que des descripteurs de fichier prédéfinis associés par défaut aux entrées/sorties des commandes
- ★ Pour chaque commande, on distingue :
  - ★ l'entrée standard : c'est le clavier
  - ★ la sortie standard : c'est l'écran
  - ★ la sortie d'erreur standard : c'est l'écran

# Entrée standard (1)

**De nombreuses commandes, invoquées sans arguments en entrée, lisent leurs données sur l'*entrée standard***

- ★ Le *file descriptor* décrivant l'entrée standard est 0
- ★ L'entrée standard est aussi le fichier `/dev/stdin`
- ★ Par défaut, l'entrée standard est le clavier
- ★ L'entrée standard peut être écrite (redirigée) dans un fichier en utilisant le symbole `<`

# Entrée standard (2)

```
★ [hal@/home/bob] cat  
windows  
windows  
linux  
linux  
<Ctrl D>
```

`cat` prend l'entrée standard comme entrée, c'est à dire tout ce qui **est tapé** sur le clavier ! (jusqu'au premier **<Ctrl D>**)

```
★ cat < participants.txt
```

L'entrée standard de `cat` est prise dans le fichier indiqué. Note : ici, on écrirait plutôt `cat participants.txt`

```
★ ls < les_fichiers_a_lister
```

Ne marche pas car la commande `ls` ne lit pas sur l'entrée standard mais traite seulement ses arguments !

# Entrée standard (3)

★ `[hal@/home/bob] sort`

`windows`

`linux`

`<Ctrl D>`

`linux`

`windows`

`sort` prend l'entrée standard comme entrée, c'est à dire tout ce qui **est tapé** sur le clavier ! (jusqu'au premier `<Ctrl D>`)

★ `sort < participants.txt`

L'entrée standard de `sort` est prise dans le fichier indiqué. Note : ici, on écrirait plutôt `sort participants.txt`

★ `mail -s "Hello" dan < my_mail.txt`

L'entrée standard de la commande `mail` est redirigée vers le fichier `my_mail.txt` qui contient le corps du message

# Sortie standard (1)

Toutes les commandes qui produisent du texte sur le terminal le font en écrivant sur leur *sortie standard*

- ★ Le *file descriptor* décrivant la sortie standard est `1`
- ★ La sortie standard est aussi le fichier `/dev/stdout`
- ★ Par défaut, la sortie standard est l'écran
- ★ La sortie standard peut être écrite (redirigée) dans un fichier en utilisant le symbole `1>` ou simplement `>`
- ★ La sortie standard peut être rajoutée à la fin d'un fichier existant par le symbole `1>>` ou simplement `>>`

# Sortie standard (2)

★ `cat > un_fichiers.txt`

Copie l'entrée standard (les caractères tapés au clavier) dans `un_fichiers.txt`

★ `ls ~ > tous_mes_fichiers.txt`

Copie le résultat de la commande `ls` dans le fichier

`tous_mes_fichiers.txt`. Le fichier est écrasé s'il existait avant ou bien créé s'il n'existait pas.

★ `ls ~/bin >> tous_mes_fichiers.txt`

Ajoute la sortie de la nouvelle commande `ls` à la suite du fichier

`tous_mes_fichiers.txt`

★ `cat .zlogin .zshrc .zlogout > zsh_all.txt`

Concatène le contenu des trois fichiers `.zlogin`, `.zshrc` et `.zlogout` dans le fichier `zsh_all.txt`

★ `find / -name '*.gif' -print > images.txt`

Copie la sortie de la commande `find` dans le fichier `images.txt`

# L'erreur standard (1)

Les messages d'erreur d'une commande sont envoyés vers l'*erreur standard* (et non pas la sortie standard)

- ★ Le *file descriptor* décrivant l'erreur standard est 2
- ★ L'erreur standard est aussi le fichier `/dev/stderr`
- ★ Par défaut, l'erreur standard est l'écran
- ★ L'erreur standard peut être écrite (redirigée) dans un fichier en utilisant le symbole `2>`
- ★ L'erreur standard peut être rajoutée à la fin d'un fichier existant par le symbole `2>>`

# L'erreur standard (2)

- ★ Redirection de l'erreur standard vers un fichier :

```
gcc myfile.c 2> erreur.log
```

- ★ Redirection de la sortie et de l'erreur standard vers des fichiers distincts :

```
gcc myfile.c > comp.log 2> erreur.log
```

- ★ Redirection de la sortie et de l'erreur standard vers le même fichier :

```
gcc myfile.c > comp.log 2>&1
```

Syntaxe alternative :

```
gcc myfile.c &> comp.log
```

- ★ On peut ignorer les erreurs en redirigeant l'erreur standard vers le pseudo-fichier `/dev/null` :

```
find / -name '*.gif' > images.txt 2> /dev/null
```

# Exercices III.1

1. Fabriquez à l'aide d'une seule commande le fichier de texte `linux/date.txt` qui contient la date et l'heure courantes
2. Fabriquez le fichier `linux/debut` formé des 15 premières lignes de `/etc/fstab`
3. Fabriquez le fichier `linux/fin` formé des 15 dernières lignes de `/etc/fstab`
4. Fabriquez le fichier `linux/extremes` qui contient les lignes de `linux/debut` suivies des lignes de `linux/fin`
5. Fabriquez (en plusieurs étapes) le fichier `linux/extremes.2`, identique à `linux/extremes`, mais sans utiliser de fichiers intermédiaires
6. Fabriquez le fichier `linux/mes_reps.txt` qui contient la liste récursive de tous vos répertoires et sous-répertoires
7. Recherchez dans le système le chemin *absolu* du fichier `fstab` sans générer aucun message d'erreur
8. Fabriquez le fichier `linux/network.txt` qui contient exactement la liste récursive de tous les fichiers et répertoires du répertoire `/etc` et dont le nom contient la chaîne `net`

# Les tuyaux (1)

## Pour combiner et enchaîner les commandes entre elles

- ★ Les tuyaux Unix servent à rediriger la sortie d'une commande vers l'entrée d'une autre commande :

```
★ find . -name '*.gif' -print | more
```

```
★ ls | wc -l
```

```
★ cat *.log | grep -i erreur | sort
```

```
★ cat /home/*/devoirs.txt | grep note | more
```

- ★ On peut combiner tuyaux et redirections :

```
★ cat devoirs/*/note.txt | grep note > notes.txt
```

```
★ find / -name '*.cpp' 2> /dev/null | more
```

# Les tuyaux (2)

**Les tuyaux s'utilisent principalement avec des filtres destinés à modifier la sortie d'une commande par application successive d'autres commandes**

- ★ Recherche de ligne contenant un motif parmi les lignes du flux d'entrée avec les commandes `grep`, `egrep` et `fgrep`
- ★ Sélection de colonnes ou de champs sur les lignes du flux d'entrée avec la commande `cut`
- ★ Tri des lignes du flux d'entrée avec la commande `sort`
- ★ Remplacement de caractères sur les lignes du flux d'entrée avec la commande `tr`
- ★ Duplication du canal de sortie standard avec la commande `tee`

# Les commandes `tr` et `cut`

★ `tr chaine1 chaine2`

Remplace les caractères de `chaine1` par les caractères de `chaine2`

★ `echo Bonjour Tout le Monde | tr 'Bo1' 'b0L'`  
Affiche `b0nj0ur t0ut Le m0nde`

★ `echo Bonjour Tout le Monde | tr 'A-Z' 'a-z'`  
Affiche `bonjour tout le monde`

★ `echo 'a demain !' | tr -s ' '`  
Affiche `a demain !`

★ `cut [option]... [fichier]...`

Retourne des *parties* des lignes du fichier `fichier` (ou de l'entrée standard)

★ `echo '/usr/bin/X11:/usr/bin:/bin' | cut -d':' -f2`  
Retourne `/usr/bin`

★ `ls -l | tr -s ' ' | cut -d' ' -f3,6,8`  
Affiche les fichiers du répertoire courant précédés par le propriétaire et la date de dernière modification, à raison d'un fichier par ligne

# La commande `tee`

```
tee [-a] fichier
```

★ La commande `tee` peut être utilisée pour envoyer en même temps la sortie standard vers l'écran et vers un fichier.

```
★ make | tee compil.log
```

Lance la commande `make` et stocke sa sortie dans le fichier `compil.log`

```
★ make install | tee -a compil.log
```

Lance la commande `make install` et rajoute sa sortie à la fin du fichier `compil.log`

# Exercices III.2

1. Reprenez les exemples d'utilisation des tuyaux et expérimentez !
2. Lisez avec attention les pages de manuel des commandes `tr` et `cut`, et analysez avec soin les options `-s` (pour `tr`), `-d` et `-f` (pour `cut`)
3. Affichez le nombre de fichiers et/ou répertoires contenus dans votre *homedir*
4. Affichez la liste des fichiers (et seulement des fichiers) contenu dans votre *homedir*
5. Affichez le nombre total de fichiers réguliers (ni répertoire, ni lien symbolique) contenu récursivement dans votre *homedir*
6. Afficher le nombre total de fichiers du système de nom `*.cpp` auxquels vous avez accès (en lecture)
7. Affichez à l'aide d'une seule commande, en les numérotant de 40 à 50, les lignes 40 à 50 (incluses) du fichier `/etc/fstab`
8. Affichez la taille suivie du nom des fichiers de votre répertoire `.kde/share/wallpapers` sous forme lisible (par exemple `4.0K` au lieu de `4096`), et seulement la taille et le nom, à raison d'une ligne par fichier

# La notion de processus (1)

- ★ Depuis sa création, Unix prend en charge le vrai multi-tâche préemptif
- ★ Faculté de lancer de nombreuses tâches en parallèle, et de les interrompre même si elles ont corrompu leur propre état ou leur propres données
- ★ Faculté de choisir quels programmes précis vous lancez
- ★ Faculté de choisir les entrées utilisées par vos programmes, et de choisir où vont leurs sorties

# La notion de processus (2)

**En Linux, tout ce qui s'exécute est un processus**

★ Un processus :

- ★ une commande, un groupe de commande ou un script en cours d'exécution
- ★ son contexte d'exécution : entrées/sorties, variables d'environnement

★ Chaque processus :

- ★ s'exécute de façon indépendante et en temps partagé
- ★ possède ses propres entrées/sorties
- ★ peut être individuellement suspendu, réactivé ou tué

# Processus en tâche de fond

## Même mode d'utilisation dans tous les shells

★ Très utile :

★ Pour les tâches en ligne de commande dont les résultats peuvent être examinés plus tard, en particulier celles qui prennent beaucoup de temps.

★ Pour lancer des applications graphiques depuis la ligne de commande et les utiliser ensuite à la souris.

★ Démarrer une tâche et ajouter & à la fin

```
★ find / -name '*.gif' > ~/images.txt &
```

# Contrôle des tâches de fond

## ★ jobs

Fournit la liste des tâches de fond issues du même shell

```
[1]-  Running find / -name '*.gif' > ~/images.txt &  
[2]+  Running make install > /dev/null &
```

## ★ fg

fg %n

Faire de la dernière (ou n-ième tâche) de fond la tâche courante

## ★ Mettre la tâche courante en arrière plan :

<Ctrl Z>

bg

## ★ kill %n

Interrompt la nième tâche

# Exemples de contrôle de tâches

```
[hal@home/bob] jobs
[1]-  Running find / -name '*.gif' > ~/images.txt &
[2]+  Running make install > /dev/null &

[hal@home/bob] fg
make install

[hal@home/bob] <Ctrl Z>
[2]+  Stopped make install

[hal@home/bob] bg
[2]+  make install &

[hal@home/bob] kill %1
[1]+  Terminated find / -name '*.gif' > ~/images.txt &
```

# Liste de tous les processus

## ★ `ps -ux`

Affiche tous les processus appartenant à l'utilisateur courant.

## ★ `ps aux` (remarque: `ps -edf` sur systèmes System V)

Affiche tous les processus existant sur le système

## ★ `ps -aux | grep bart | grep bash`

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
bob	3039	0.0	0.2	5916	1380	pts/2	S	14:35	0:00	/bin/bash
bob	3134	0.0	0.2	5388	1380	pts/3	S	14:36	0:00	/bin/bash
bob	3190	0.0	0.2	6368	1360	pts/4	S	14:37	0:00	/bin/bash
bob	3416	0.0	0.0	0	0	pts/2	RW	15:07	0:00	[bash]

★ PID: (Process ID) Identifiant du processus

VSZ: (Virtual SiZe) Taille virtuelle du processus (code + données + pile)

RSS: (ReSident Size) Nombre de Ko occupés en mémoire

TTY: (TeleTYpe) Terminal

STAT: Statut: R (Runnable: exécutable), S (Sleep: endormi), W (paging: en cours de pagination), Z (Zombie)...

# Arrêt de processus

## ★ `kill pid ...`

Envoie un signal d'arrêt aux processus spécifiés. Cela permet aux processus de sauvegarder leurs données et s'arrêter eux-mêmes. A utiliser en premier recours.

Exemple:

```
kill 3039 3134 3190 3416
```

## ★ `kill -9 pid ...`

Envoie un signal d'arrêt immédiat. Le système lui-même se charge d'arrêter les processus. Utile quand une tâche est vraiment bloquée (ne répond pas à un `kill` simple)

## ★ `killall [-signal] commande`

Arrête toutes les tâches exécutant `commande`. Exemple:

```
killall bash
```

## ★ `kill -9 -1`

Arrête tous les processus de l'utilisateur courant (`-1` : tous les processus)

# Activité en temps réel des processus

**top** : Affiche les processus les plus actifs, triés par utilisation du proc.

```
top - 15:44:33 up 1:11, 5 users, load average: 0.98, 0.61, 0.59
Tasks: 81 total, 5 running, 76 sleeping, 0 stopped, 0 zombie
Cpu(s): 92.7% us, 5.3% sy, 0.0% ni, 0.0% id, 1.7% wa, 0.3% hi, 0.0% si
Mem: 515344k total, 512384k used, 2960k free, 20464k buffers
Swap: 1044184k total, 0k used, 1044184k free, 277660k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3809	jdoe	25	0	6256	3932	1312	R	93.8	0.8	0:21.49	bunzip2
2769	root	16	0	157m	80m	90m	R	2.7	16.0	5:21.01	X
3006	jdoe	15	0	30928	15m	27m	S	0.3	3.0	0:22.40	kdeinit
3008	jdoe	16	0	5624	892	4468	S	0.3	0.2	0:06.59	autorun
3034	jdoe	15	0	26764	12m	24m	S	0.3	2.5	0:12.68	kscd
3810	jdoe	16	0	2892	916	1620	R	0.3	0.2	0:00.06	top

★ L'ordre de tri peut être changé en tapant

**M**: utilisation mémoire, **P**: %CPU, **T**: temps d'exécution.

★ On peut arrêter une tâche en tapant **k** (kill) et son numéro

# Commandes `time`, `times` et `uptime`

## ★ `time` commande

Exécute la commande `commande` et affiche le temps consommé par cette exécution.

Exemple :

```
$ time ls -R / &> /dev/null
real    0m7.917s
user    0m1.698s
sys     0m2.294s
```

## ★ `times`

Affiche les temps cumulés pour tous les processus lancés depuis le début de la session

## ★ `uptime`

Affiche le temps depuis lequel le système est actif (donc le temps depuis le dernier *boot*) ainsi que les valeurs moyennes de charge du système

★ Il existe également de nombreuses applications graphiques pour contrôler les paramètres systèmes (charge, processus) comme par exemple `ksysguard` sous KDE

# Groupage de commandes

- ★ Séquence de commandes avec le symbole ;  
`echo "mes fichiers"; ls; echo "-----"`
- ★ Groupage logique avec les opérateurs `||` (ou) et `&&` (et)
  - ★ `cat ~/infos 2>/dev/null || echo "erreur"`  
N'exécute `echo` que si la première commande échoue
  - ★ `ls ~/infos 2>/dev/null && cat ~/infos`  
N'affiche le contenu d'`infos` que si la commande `ls` réussit
- ★ Groupage en sous-shell avec les parenthèses "`(`" et `)`"  
`pwd; ( cd ..; pwd ); pwd`  
Affiche successivement les chemins du répertoire courant, du répertoire père, puis à nouveau du répertoire courant

# Exécutions différées

★ `sleep 60`

Attend 60 secondes (ne consomme pas de ressources système)

★ `at 6:00pm today chmod o-rx /net/devoirs`

Enlève aujourd'hui à 18h les droits en lecture et en accès au répertoire `/net/devoirs` pour *"others"*

★ `crontab`

Gère la `crontab` de l'utilisateur pour lancer des commandes en temps différé

# Exercices III.3 (1)

1. Exécutez la commande `xclock -update 1`. Pouvez-vous exécuter une nouvelle commande dans le terminal dans lequel vous venez de lancer ce processus ? pourquoi ?
2. Suspendez l'exécution de ce processus sans le tuer. Exécutez la commande `jobs`. Qu'indique cette commande ? Reprenez l'exécution du processus suspendu en arrière plan
3. Exécutez la commande `xclock -digital -update 1` dans le même terminal. Suspendez l'exécution de ce processus sans le tuer. Exécutez la commande `jobs`. Qu'indique cette commande ? Reprenez l'exécution de ce dernier processus suspendu en arrière plan
4. Affichez le PID des deux processus que vous venez de lancer
5. Faites passer le premier processus (issu de `xclock -update 1`) en avant plan et arrêter ce processus. Exécutez la commande `jobs`. Qu'indique cette commande ?
6. Exécutez la commande `kill -9 %2`. Que se passe-t-il ? Exécutez la commande `jobs`. Qu'indique cette commande ?
7. Afficher le PID d'une commande par son nom, de manière que seul le PID et le nom de la commande exécutée s'affiche sur la ligne (ou sur les lignes si la même commande est lancée plusieurs fois)

# Exercices III.3 (2)

1. Trouvez deux façons de lancer la commande `xlogo` dans soixante secondes à partir de maintenant (utilisez la commande `xlogo` avec l'option `-d` et tapez : `xlogo -d : 0.0` )
2. A l'aide de la commande `crontab`, afficher un nouveau logo *Xwindow* toute les minutes
3. Exécutez la commande `top` et regardez en direct les processus qui s'exécutent sur votre machine, ainsi que la charge de votre machine
4. Lancez `ksysguard` et regardez en direct les processus qui s'exécutent sur votre machine, ainsi que la charge de votre machine
5. Affichez le temps que mets un `ls -R` pour s'exécuter depuis votre *homedir*, et n'affichez rien d'autre
6. Affichez le temps que mets un `ls -R` pour s'exécuter depuis le répertoire racine `/`, et n'affichez rien d'autre
7. Affichez exactement le temps depuis lequel le système linux dans lequel vous êtes a été

# IV. Compléments sur le shell

- ★ Variables
- ★ Alias
- ★ Substitution de commandes
- ★ Fichiers de configuration bash
- ★ La commande for
- ★ Les commandes basename et dirname

# VARIABLES D'ENVIRONNEMENT

- ★ Les shells permettent à l'utilisateur de définir des *variables*.  
Celles-ci peuvent être réutilisées dans les commandes shell.  
Convention : les noms sont en minuscules
- ★ Vous pouvez aussi définir des *variables d'environnement*: des variables qui sont aussi visibles depuis les scripts ou les exécutable appelés depuis le shell.  
Convention : les noms sont en majuscules
- ★ La commande `env`  
Affiche toutes les variables d'environnement existantes ainsi que leur valeur

# Variables de shell

- ★ Un seul type possible : la chaîne de caractères
- ★ Pas besoin d'être déclarée pour recevoir une valeur par affectation

```
une_variable=100
```

- ★ **Attention** : il ne faut **PAS** mettre d'espace entre la variable, le signe = et la valeur

- ★ Utilisation d'une syntaxe particulière pour accéder à la valeur

```
echo $une_variable
```

- ★ Pour qu'une variable garde sa valeur initiale dans un sous-shell

```
export une_variable
```

# Exemples de variables de shell

## Variables de shell

```
★ projdir=/home/bob/unix/projet  
ls -la $projdir; cd $projdir
```

## Variables d'environnement

```
★ cd $HOME  
★ echo $PWD  
★ echo $PRINTER  
★ export JAVA_SOURCE=$HOME/java/src
```

# Variables standards

Utilisées par de nombreuses applications

★ **HOME**

Répertoire de l'utilisateur courant.

★ **PATH**

Chemin de recherche des commandes

★ **USER**

Nom de l'utilisateur courant

★ **HOST**

Nom de la machine locale

★ **PRINTER**

Nom de l'imprimante par défaut

★ **SHELL**

Nom du shell courant

★ **MANPATH**

Chemin de recherche des pages de manuel.

★ **EDITOR**

Éditeur par défaut (vi, emacs...)

★ **TERM**

Nom du terminal / mode courant

★ **DISPLAY**

Écran sur lequel afficher les applications X (graphiques)

★ **LD\_LIBRARY\_PATH**

Chemin de recherche de bibliothèques partagées

# Les variables de type PATH

## ★ PATH

Spécifie l'ordre de recherche des commandes pour le shell

```
~/bin:/usr/local/bin:/usr/bin:/bin:/sbin
```

## ★ LD\_LIBRARY\_PATH

Spécifie l'ordre de recherche pour les bibliothèques partagées (codes binaires partagés par les applications, comme la bibliothèque C) pour `ld`

```
/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib
```

## ★ MANPATH

Spécifie l'ordre de recherche pour les pages de manuel

```
/usr/local/man:/usr/share/man
```

# Variables et substitutions

- ★ Substitution de la valeur d'une variable

```
echo $HOST
```

- ★ Substitution protégée de la valeur d'une variable

```
TMP=${USER}_$$ ( et non $USER_$$ )
```

- ★ Substitution du résultat d'une commande

```
nb_files=$(ls | wc -l) (ou `ls | wc -l` )
```

- ★ Autres substitutions

Dans les substitutions suivantes, si la variable `var` a une valeur, alors

`${var-VAL}` substitue `var`, sinon substitue `VAL`

`${var=VAL}` substitue `var`, sinon donne à `var` la valeur `VAL` et substitue `var`

`${var?VAL}` substitue `var`, sinon affiche `VAL` et quitte le shell

`${var+VAL}` substitue `VAL`, sinon ne substitue rien

# Alias

Les shells vous permettent de définir des *alias* (des raccourcis pour des commandes que vous utilisez très souvent)

Exemples

★ `alias ls='ls -F'`

Utile pour toujours lancer des commandes avec certains paramètres

★ `alias rm='rm -i'`

Utile pour faire que `rm` demande toujours une confirmation

★ `alias cherche='find $HOME -name'`

Utile pour remplacer des commandes utilisées régulièrement

★ `alias cia='. /home/sydney/env/cia.sh'`

Utile pour initialiser rapidement un environnement

(`.` est une commande shell pour exécuter le contenu d'un script shell)

★ En **bash** les alias sont définis dans le fichier `.bash_aliases`

# Les fichiers de bash (1)

**Des scripts shell sont lus automatiquement à chaque fois qu'un shell bash est lancé ou terminé**

- ★ Le script `/etc/profile` :  
non modifiable par l'utilisateur
- ★ Les scripts `~/.bash_profile` et `~/.bash_logout` :  
modifiables par l'utilisateur
- ★ Permet de personnaliser son environnement
- ★ Les scripts `~/.bash*` ne sont pas obligatoires

# La commande **for** (1)

- ★ Exécute les commandes **commandes** en donnant successivement à la variable **var** les valeurs **val\_1, ..., val\_2, val\_n**

```
for var in val_1 val_2 ... val_n
do
    commandes
done
```

- ★ **val\_i** peut être une chaîne de caractères quelconque incluant des caractères génériques (wildcards)

```
for file in *; do ... done
```

- ★ En l'absence de **val\_i**, la variable **var** prend comme valeurs successives les valeurs des **paramètres** du script :

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

# Les fichiers de bash (2)

★ `/etc/profile` et `~/.bash_profile`

Scripts shell lus en séquence à chaque fois qu'un shell `bash` est lancé lors d'un login

★ `~/.bashrc`

Scripts shell lus si le shell lancé est interactif et non pas un shell de login. En général, ce fichier est également **intégré** dans le fichier `~/.bash_profile`

★ `~/.bash_logout`

Scripts shell lus à la sortie d'un shell de login (après la commande **exit**)

# Les fichiers de bash (3)

- ★ `/etc/profile` et `~/.bash_profile`

Contient les initialisations des variables d'environnement propre à un shell de login (`umask`, `prompt`) et ensuite inclut le fichier `~/.bashrc`

- ★ `~/.bashrc`

Contient les initialisations des autres variables d'environnement comme le `PATH`, mais aussi des fonctions ou des alias

- ★ `~/.bash_logout`

Contient des commandes spécifiques à la sortie d'un shell login (nettoyage)

# La commande **for** (2)

- ★ Pour afficher les fichiers exécutables du répertoire courant :

```
for file in *  
do [ -f $file -a -x $file ] && echo $file; done
```

- ★ Pour compiler des fichiers sources C du répertoire courant :

```
for file in prog1.c prog2.c prog3.c  
do gcc -c $file; done
```

- ★ Pour imprimer tous les fichiers sources C du répertoire courant :

```
for file in *.c  
do lpr $file; done
```

- ★ Pour tuer tous les processus de l'utilisateur de nom "emacs" :

```
for proc in $(ps x | grep emacs | cut -d' ' -f2)  
do kill $proc; done
```

# La commande `seq`

- ★ Affiche une séquence de nombres

```
seq i j
```

Si `i` est plus petit ou égal à `j`, affiche les entiers `i, i+1, ..., j`, sinon ne fait rien

```
seq -s char i j
```

Remplace le séparateur (un `\n` par défaut) par le caractère `char`

```
seq i k j
```

Si `i` est plus petit ou égal à `j`, affiche les entiers `i, i+k, ..., j`, pour tout les `k` tels que `i+k` ne dépasse pas `j`, sinon ne fait rien

- ★ Utilisé la plupart du temps avec l'itération `for`

```
for i in $(seq 1 $max); do  
    mkdir "tmp$i"
```

```
done
```

# Manipulation des chemins

★ `basename fichier [suffix]`

Retourne le nom de base d'un fichier :

★ `basename xxx/yyy/zzz`

Retourne `zzz`

★ `basename xxx/yyy/zzz.c`

Retourne `zzz.c`

★ `basename xxx/yyy/zzz.c '.c'`

Retourne `zzz`

★ `dirname fichier`

Retourne le *préfixe* répertoire d'un fichier :

★ `dirname xxx/yyy/zzz`

Retourne `xxx/yyy`

La chaîne de caractères `fichier` n'a pas besoin d'être le chemin d'un vrai fichier !

# Exercices IV.1

1. Créez le répertoire `linux/java`, et sous ce répertoire, créez automatiquement plusieurs fichiers (vides) dont les noms sont de la formes `tpX.exoY.java` avec `X` prenant les valeurs de 1 à 10 et `Y` prenant les valeurs de 1 à 5 (soit 50 fichiers au total)
2. Fabriquez pour chaque fichier de nom `tp9.exo*.java` ou `tp10.exo*.java` une copie de nom `tp9.exo*.java.bak` ou `tp10.exo*.java.bak`
3. Renommez chaque fichier de nom `tp10.exo*.java.bak` en `tp10.exo*.txt`
4. Créez automatiquement sous le répertoire `linux/java` 10 répertoires `tp1`, `tp2`, ..., `tp10`
5. Déplacez tout fichier de nom `tpX*` dans le répertoire `tpX`
6. Créez le répertoire `linux/passwd`, et sous ce répertoire, créez automatiquement autant de sous-répertoires qu'il y a de lignes dans le fichiers `/etc/passwd`, chaque sous-répertoire ayant pour le nom le premier champ de chaque ligne (par exemple `root` pour la première ligne)
7. Créez automatiquement dans chaque répertoire de nom `rep` du répertoire `linux/passwd` un fichier de nom `info.txt` vide
8. Ajouter à chaque fichier `info.txt` du répertoire `linux/passwd/rep` le nom en clair de l'utilisateur ou du processus correspondant
9. Renommez tous les fichiers `linux/passwd/rep/info.txt` en `linux/passwd/info.rep` et effacez tous les sous-répertoires `linux/passwd/rep`

# V. Programmation shell

- ★ **Script shell**
- ★ **Variables, paramètres et substitutions**
- ★ **Structures de contrôle**
- ★ **Fonctions**
- ★ **Commandes utiles**

# Scripts shell

- ★ Langages du shell

- ★ Un langage de programmation interprété

- ★ Notion de variables, de paramètres, de structures de contrôles, de fonctions,...

- ★ Autant de langages que de shells (`sh`, `bash`, `ksh`, `zsh`,...)

- ★ Script shell

Un fichier exécutable contenant des instructions d'un shell donné

- ★ Scripts shell et commandes Linux

Equivalents et traités de la même façon par le système

# Contenu d'un script shell

- ★ L'en-tête

Indique le shell à utiliser pour interpréter les commandes placées dans le fichier : `#!/bin/bash` (pour le Bourne Again SHell)

- ★ Des variables

- ★ déclaration/affectation : `une_variable=123`

- ★ valeur : `$une_variable`

- ★ paramètres : `$1, $2, ..., $9`

- ★ Des appels à des commandes ou à des scripts

Toutes les commandes Linux et tous les scripts accessibles

- ★ Des instructions

Affectation/référence de variables, instructions conditionnelles et itératives, instructions d'entrée/sortie,...

# Variables et paramètres (1)

## ★ Variables spéciales

- ★ `$0` : le nom du script (du fichier)
- ★ `$#` : le nombre de paramètres du script
- ★ `$*` : la liste des paramètres du script
- ★ `$?` : le code de retour de la dernière commande exécutée
- ★ `$$` : le numéro de processus du shell courant (PID)

## ★ Paramètres

- ★ Seulement 9 noms de variable `$1, $2, ..., $9`
- ★ L'instruction `shift` supprime le premier paramètre et décale les autres vers la gauche
- ★ L'instruction `shift n` supprime les `n` premiers paramètres et décale les autres vers la gauche

# Variables et paramètres (2)

## Variables prédéfinies en bash et mise à jour automatiquement

- ★ **PPID** : le numéro du processus père
- ★ **PWD** : le nom du répertoire courant
- ★ **RANDOM** : un nombre entier aléatoire
- ★ **SECONDS** : le temps (en secondes) écoulé depuis le lancement du shell courant
- ★ **!** : le numéro du dernier processus lancé en arrière-plan
- ★ **\_** : le dernier mot de la dernière commande exécutée

## Variable d'environnement pour l'exécution de script

- ★ **BASH\_ENV** : le nom du fichier bash exécuté à chaque début de script

# Exemple de script (1)

```
#!/bin/bash
# début du script
echo "Hello $USER"
echo "You are currently on $HOST"
echo "The content of your homedir:"
/bin/ls
# affectation de la variable NBFILLES
NBFILLES=$(ls | wc -l)
echo "Total: $NBFILLES elements"
```

welcome.sh

# Exemple de script (2)

```
[hal@/home/bob] chmod a+x welcome.sh

[hal@/home/bob] ls -l welcome.sh
-rwxr-xr-x  1 bob bob 201 2005-08-01 17:05 welcome.sh*

[hal@/home/bob] ls -F
Desktop/      Trash/        emacs/        tmp/  welcome.sh*
Mail/         bin/          lib/          unix/

[hal@/home/bob] ./welcome.sh
Hello bob
You are currently on hal
The content of your homedir:
Desktop      Trash      emacs      tmp      welcome.sh*
Mail         bin        lib        unix
Total: 9 elements
```

# Exécution d'un script shell

- ★ `. my_script.sh` ou `bash my_script.sh`  
Exécute les commandes contenues dans le fichiers `my_script.sh` dans le **shell courant**
- ★ `./my_script` (le fichier `my_script` a les droits `rx`)  
Exécute les commandes contenues dans le fichiers `my_script.sh` dans un **sous-shell** du shell courant
- ★ Autres commandes qui s'exécutent dans un **sous-shell** du shell courant
  - ★ Les commandes reliées par des tuyaux  
`cat *.log | grep -i erreur | sort`
  - ★ Les commandes groupées entre "(" et ")"  
`pwd; ( cd ../; pwd ); pwd`
  - ★ Les structures de contrôles (voir plus loin)  
`if, case, for, while` et `until`

# Sortie d'un script shell

- ★ `exit n`

Termine et sort du script shell courant avec le code de retour `n`

- ★ `exit`

Termine et sort du script shell courant. Le code de retour est celui de la dernière commande exécutée dans ce script

- ★ Exemple

```
if [ $# -ne 2 ]; then
    echo "wrong number of arguments: $#"  
    exit 1
fi
```

- ★ Valeur du code de retour

`0` signifie que le script s'est exécuté correctement

Différent de `0` signifie qu'une erreur est survenue

# Exercices V.1

1. Écrivez la commande `args` qui prend un nombre quelconque d'arguments eux-même quelconques et qui, lorsqu'on l'exécute, affiche son nom, le nombre de ses arguments et la liste de ses arguments :

```
$ args fichier $USER 777 tutu
```

```
args est appelé avec 4 argument(s) : fichier zorro  
777 tutu
```

2. Modifiez *éventuellement* la commande `args` pour qu'elle fonctionne sans aucune modification, même après qu'on l'ait renommée (par exemple, par `mv args script`). Autrement dit, la commande doit toujours afficher son nom !
3. Écrivez la commande `words` qui affiche chaque mots du fichier (de texte) qu'on lui passe en argument, à raison d'un mot par ligne, chaque mot étant précédé de son nombre d'occurences. Les mots doivent apparaître triés par ordre alphabétique (considérez les commandes `tr`, `sort` et `uniq`).
4. Modifiez la commande `words` pour qu'elle prennent plusieurs fichiers (de texte) en paramètres.

# Variables de shell

- ★ Un seul type possible : la chaîne de caractères
- ★ Pas besoin d'être déclarée pour recevoir une valeur par affectation

```
une_variable=100
```

- ★ **Attention** : il ne faut **PAS** mettre d'espace entre la variable, le signe = et la valeur
- ★ Utilisation d'une syntaxe particulière pour accéder à la valeur

```
echo $une_variable
```
- ★ Pour qu'une variable garde sa valeur initiale dans un sous-shell

```
export une_variable
```

# Exemples de variables de shell

## Variables de shell

```
★ projdir=/home/bob/unix/projet  
ls -la $projdir; cd $projdir
```

## Variables d'environnement

```
★ cd $HOME  
★ echo $PWD  
★ echo $PRINTER  
★ export JAVA_SOURCE=$HOME/java/src
```

# Variables et substitutions

- ★ Substitution de la valeur d'une variable

```
echo $HOST
```

- ★ Substitution protégée de la valeur d'une variable

```
TMP=${USER}_$$ (et non $USER_$$ )
```

- ★ Substitution du résultat d'une commande

```
nb_files=$(ls | wc -l) (ou `ls | wc -l` )
```

- ★ Autres substitutions

Dans les substitutions suivantes, si la variable `var` a une valeur, alors

`${var-VAL}` substitue `var`, sinon substitue `VAL`

`${var=VAL}` substitue `var`, sinon donne à `var` la valeur `VAL` et substitue `var`

`${var?VAL}` substitue `var`, sinon affiche `VAL` et quitte le shell

`${var+VAL}` substitue `VAL`, sinon ne substitue rien

# Variables et lecture de valeurs

**La commande `read` permet de lire des chaînes de caractères sur l'entrée standard et de les affecter à des variables**

★ `read line`

Lit une ligne (une suite de caractères terminée par un *newline*) sur l'entrée standard et l'affecte à la variable `line`

★ `read var_1 var_2 ... var_n`

Lit une ligne sur l'entrée standard, la découpe en *mots*, et affecte chaque mot aux variables `var_1`, `var_2`, ..., `var_n`

★ S'il y a moins de mots que de variables, les variables de traîne sont initialisées avec la chaîne vide ""

★ S'il y a plus de mots que de variables, la dernière variable reçoit comme valeur la chaîne formée des mots de traîne

# Variables et affectations

La commande `set` permet d'affecter des chaînes de caractères aux variables spéciales d'un script shell (paramètres) `$1`, `$2`, ...

★ `set -- val_1 val_2 ... val_n`

Affecte les valeurs `val_1`, `val_2`, ..., `val_n` aux variables `$1`, `$2`, ..., `$n`

★ Mais aussi

★ `set` (sans paramètre)

Affiche toutes les variables positionnées avec leur valeur

★ `set -x`

Affiche les commandes avant qu'elles soient exécutées

★ `set -a [+a]`

Exporte toutes les variables affectées entre les instructions

`set -a` et `set +a` (seulement dans un script)

# Exercices V.2

1. Écrivez une première version de la commande `welcome1` qui ne prend aucun argument, et qui, lorsqu'elle est exécuté par l'utilisateur `zorro` sur la machine de nom `hal`, affiche les informations suivantes :

```
$ welcome1
```

```
Hello zorro, you're logged on hal and current date  
and time are: Fri Oct 6 17:56:03 CEST 2006
```

2. Écrivez une première version de la commande `nospace1` qui prend en argument un fichier ou un répertoire dont le nom contient des *espaces*, et qui renomme son argument en remplaçant les *espaces* par des `_` (soulignés). Vous devez utiliser la commande `tr` pour réaliser la commande `nospace1`. Exemple :

```
$ ls Bon*
```

```
Bonjour tout le monde
```

```
$ nospace1 Bonjour\ tout\ le\ monde
```

```
$ ls Bon*
```

```
Bonjour_tout_le_monde
```

# Structures de contrôle

## Identiques à un langage de programmation (comme C ou Java)

### ★ Instructions conditionnelles

- ★ Les formes `&&` et `||`  
Déjà vues !

- ★ L'instruction `if-then-else-fi`  
Sélection à une, deux ou plusieurs alternatives

- ★ L'instruction `case-esac`  
Aiguillages à valeurs multiples

### ★ Instructions itératives (boucles)

- ★ la boucle `for-done`  
Itération bornée

- ★ les boucles `while-done` et `until-done`  
Itérations non bornées

# Groupage de commandes

- ★ Séquence de commandes avec le symbole `;`  
`echo "mes fichiers"; ls; echo "-----"`
- ★ Groupage logique avec les opérateurs `||` (ou) et `&&` (et)
  - ★ `cat ~/infos 2>/dev/null || echo "erreur"`  
N'exécute `echo` que si la première commande échoue
  - ★ `ls ~/infos 2>/dev/null && cat ~/infos`  
N'affiche le contenu d'`infos` que si la commande `ls` réussit
- ★ Groupage en sous-shell avec les parenthèses `"( " et ")"`  
`pwd; ( cd ../; pwd ); pwd`  
Affiche successivement les chemins du répertoire courant, du répertoire père, puis à nouveau du répertoire courant

# La commande `if` (1)

## ★ Sélection à une alternative

```
if test_0
  then commandes_1
fi
```

## ★ Sélection à deux alternatives

```
if test_0
  then commandes_1
  else commandes_2
fi
```

## ★ Sélection à plusieurs alternatives

```
if test_0
  then commandes_0
  elif test_1
  then commandes_1
  ....
  then commandes_n-1
  else commandes_n
fi
```

Le code de retour de `test_0` est interprété comme un booléen :

★ code de retour `0` : le booléen est VRAI

★ code de retour différent de `0` : le booléen est FAUX

# La commande `if` (2)

- ★ Comparaison de fichiers

```
if cmp $1 $2; then
    echo "les fichiers $1 et $2 sont identiques"
else
    echo "les fichiers $1 et $2 sont différents"
```

- ★ Une version protégée de `cat`

```
if ls $1 &> /dev/null; then
    cat $1
else
    echo "le fichier $1 est introuvable"
fi
```

- ★ Version équivalente avec les constructeurs `&&` et `||`

```
ls $1 &> /dev/null && cat $1 || echo "..."
```

- ★ La forme `if` s'utilise surtout avec la forme `test`

# La commande `test` (1)

Une commande spéciale pour effectuer des tests, très utilisée avec les commandes `if`, `while` et `until`

★ Syntaxe standard

```
test expression
```

★ Syntaxe alternative

```
[ expression ]
```

Attention aux espaces entre "[", "]" et `expression`

★ Produit une valeur de retour (un entier) égale à 0 si le test est VRAI, à 1 sinon

★ `expression` est un **prédicat** spécifique qui ne fonctionne qu'avec la construction `test`

# La commande `test` (2)

## Prédicats sur les chaînes de caractères

- ★ `c1 = c2` (resp. `!=`)      VRAI si les chaînes sont égales  
(resp. différentes)
- ★ `-z chaine` (resp. `-n`)      VRAI si la chaîne est vide  
(resp. non vide)

## Prédicats sur les entiers

Ici `n1` et `n2` sont des chaînes de caractères représentant des entiers

- ★ `n1 -eq n2` (resp. `-ne`)      VRAI si les entiers sont égaux  
(resp. différents)
- ★ `n1 -gt n2` (resp. `-lt`)      VRAI si `n1` est strictement supérieur  
(resp. inférieur) à `n2`
- ★ `n1 -ge n2` (resp. `-le`)      VRAI si `n1` est supérieur (resp.  
inférieur) ou égal à `n2`

# La commande `test` (3)

## Prédicats sur les fichiers et répertoires

Pour que les tests suivants soient VRAIS, il faut avant tout que `fichier` existe sur le disque

- ★ `-r fichier` VRAI si `fichier` est lisible (droit `r`)
- ★ `-w fichier` VRAI si `fichier` est modifiable (droit `w`)
- ★ `-x fichier` VRAI si `fichier` possède le droit `x`
- ★ `-f fichier` VRAI si `fichier` n'est pas un répertoire
- ★ `-d fichier` VRAI si `fichier` est un répertoire
- ★ `-s fichier` VRAI si `fichier` a une taille non nulle

# La commande **test** (4)

On peut combiner les prédicats à l'aide de trois opérateurs et de parenthèses

- ★ `! expr`                    VRAI si `expr` est FAUX
- ★ `expr1 -a expr2`            VRAI si `expr1` et `expr2` sont VRAIs
- ★ `expr1 -o expr2`            VRAI si `expr1` ou `expr2` est VRAI
- ★ `( expr )`                    Attention : les parenthèses ont une signification pour le shell, il faut donc les faire précéder du caractère "\"

## Exemple

```
if [ $x -gt 0 -a \( $y -ge 1 -o $y -lt $x \) ]
then
...

```

# Exercices V.3

1. Écrivez le script `welcome2`, deuxième version du script `welcome1` (exercice II.1.1), qui affiche le message de bienvenue adéquat en fonction de l'heure de la journée. Le message doit être `Good morning` entre 0 heure et 12 heures, `Good afternoon` entre 13 heures et 18 heures, et `Good evening` entre 19 heures et 23 heures :

```
$ welcome2
```

```
Good morning zorro, you're logged on hal
```

2. Écrivez le script `info1` qui affiche les informations de l'unique fichier ou répertoire qui est son argument :

```
$ info tutu
```

```
tutu n'existe pas
```

```
$ info /etc
```

```
/etc est un répertoire
```

```
$ info lien.sym
```

```
lien.sym est un lien symbolique
```

```
$ info .bashrc
```

```
.bashrc est un fichier (rw)
```

Si le script `info1` est appelé avec zéro ou plus d'un argument, il doit produire un message d'erreur.

# La commande `case` (1)

- ★ Sélectionne `val_i`, le premier choix parmi `val_1, ..., val_2, val_n`, qui correspond à la valeur de `expr`, et exécute ensuite `commandes_i`

```
case expr in
  val_1) commandes_1;;
  val_2) commandes_2;;
  ....
  val_n) commandes_n;;
esac
```

- ★ `expr` peut être une chaîne de caractères quelconque
- ★ `val_i` peut être une chaîne de caractères quelconque incluant des caractères génériques (wildcards)
- ★ `commandes_i` est une suite de zéro, une ou plusieurs commandes séparées par des ";"

# La commande `case` (2)

★ Sélections simples sur une chaîne de caractères

```
case $lang in
    french) echo "Bonjour";;
    english) echo "Hello";;
    spanish) echo "Buenos Dias";;

    esac
```

★ Sélections avec motifs et expressions régulières

```
case $arg in
    -i | -r) echo "$arg is an option";;
    -*) echo "$arg is a bad option"; exit 1;;
    [0-9]*) echo "$arg is an integer";;
    *.c) echo "$arg is a C file"; gcc -c $arg;;
    *) echo "default value";;

    esac
```

# Exercices V.4

1. Écrivez la commande `viz1` qui prend un fichier en argument et le visualise. Le programme de visualisation sera choisi en fonction de l'extension du fichier. La commande doit traiter les extensions `.ps` (fichier PostScript) et `.pdf` (fichier PDF). Pour visualiser les fichiers PostScript, vous utiliserez la commande `kghostview` et pour les fichiers PDF la commande `kpdf`.
2. Écrivez la commande `viz2`, deuxième version de la commande `viz` précédente, qui maintenant peut visualiser également des fichiers d'extension `.txt` (fichier de texte). Un fichier de texte sera affiché par la commande `less` dans un `xterm` (considérez l'option `-e`). Prévoyez un message et une confirmation de fin de visualisation.
3. Écrivez la commande `viz3`, troisième version de la commande `viz`, qui maintenant peut visualiser des fichiers d'extension `ext`, mais aussi des fichiers d'extension `ext.gz` (fichiers compressés avec la commande `gzip`) où `ext` est l'une des trois extensions précédentes. Pour les fichiers compressés, vous devez décompresser les fichiers sans toutefois modifier les fichiers arguments !

# La commande **for** (1)

- ★ Exécute les commandes **commandes** en donnant successivement à la variable **var** les valeurs **val\_1, ..., val\_2, val\_n**

```
for var in val_1 val_2 ... val_n
do
    commandes
done
```

- ★ **val\_i** peut être une chaîne de caractères quelconque incluant des caractères génériques (wildcards)

```
for file in *; do ... done
```

- ★ En l'absence de **val\_i**, la variable **var** prend comme valeurs successives les valeurs des **paramètres** du script :

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

# La commande **for** (2)

- ★ Pour afficher les fichiers exécutables du répertoire courant :

```
for file in *
do [ -f $file -a -x $file ] && echo $file;
done
```

- ★ Pour compiler des fichiers sources C du répertoire courant :

```
for file in prog1.c prog2.c prog3.c
do gcc -c $file; done
```

- ★ Pour imprimer tous les fichiers sources C du répertoire courant :

```
for file in *.c
do lpr $file; done
```

- ★ Pour tuer tous les processus de l'utilisateur de nom "emacs" :

```
for proc in $(ps x | grep emacs | cut -d' '
-f2)
do kill $proc; done
```

# Exercices V.5

1. Écrivez la commande `info2`, deuxième version de la commande `info1` (exercice III.1.2), qui maintenant prend un nombre quelconque d'arguments (au moins un) et affiche les informations pour chacun d'entre eux.
2. Écrivez la commande `nospace2`, deuxième version de la commande `nospace1` (exercice II.1.2), qui prend maintenant en paramètres des fichiers ou des répertoires (au moins un) et qui, seulement quand le fichier ou le répertoire existe et quand son nom contient des *espaces*, le renomme en remplaçant ces *espaces* par des *soulignés*. Dans tous les cas, la commande ne provoque pas d'erreur.
3. Écrivez la commande `size1` qui prend en arguments des fichiers et qui, pour chacun, affiche sa taille en (kilo-)octets suivie de son nom. La commande affiche `???` à la place de la taille si le fichier n'est pas un fichier régulier.
4. Écrivez la commande `afind` qui prend en argument un répertoire et un nom, et qui cherche dans quelles bibliothèques C appartenants au répertoire le nom est défini. Une bibliothèque C est un fichier d'extension `.a` et on trouve en général les bibliothèques standards sous `/usr/lib`. Utilisez la commande `nm` pour connaître les noms externes définis dans une bibliothèque.

# La commande `seq`

- ★ Affiche une séquence de nombres

```
seq i j
```

Si `i` est plus petit ou égal à `j`, affiche les entiers `i, i+1, ..., j`, sinon ne fait rien

```
seq -s char i j
```

Remplace le séparateur (un `'\n'` par défaut) par le caractère `char`

```
seq i k j
```

Si `i` est plus petit ou égal à `j`, affiche les entiers `i, i+k, ..., j`, pour tout les `k` tels que `i+k` ne dépasse pas `j`, sinon ne fait rien

- ★ Utilisé la plupart du temps avec l'itération `for`

```
for i in $(seq 1 $max); do
```

```
    mkdir "tmp$i"
```

```
done
```

# La commande `while`

- ★ Exécute les commandes `commandes` tant que le test `un_test` est VRAI

```
while un_test
do
    commandes
done
```

- ★ `un_test` peut être une commande **quelconque** mais le plus souvent c'est un appel à la commande `test` (ou `[]`)

```
while [ $# -gt 0 ]; do
    echo $1; shift
done
```

- ★ La commande `un_test` peut être remplacée par `:` ou `true` qui retournent toujours le code `0` (VRAI)

# La commande `until`

- ★ Exécute les commandes `commandes` tant que le test `un_test` est FAUX

```
until un_test
do
    commandes
done
```

- ★ Équivalente à une commande `while` en prenant la négation du test

```
while [ $# -gt 0 ]; do      until [ $# -eq 0 ]; do
    echo $1; shift          echo $1; shift
done                        done
```

- ★ La commande `un_test` peut être remplacée par `false` qui retourne toujours le code 1 (FAUX)

# La commande `break`

La commande `break` permet de sortir d'une boucle sans terminer l'itération en cours et sans passer par le test

```
while true
do
    echo -n "list the current dir? (y/n/q) "
    read yn

    case $yn in
        [yY] ) ls -l . ; break;;
        [nN] ) echo "skipping" ; break;;
        q ) exit ;;
        * ) echo "$yn: unknown response";;
    esac
done
```

# La commande `continue`

La commande `continue` permet de passer directement à l'itération suivante sans nécessairement terminer l'itération en cours

```
for f in *; do
    [ -f $f ] || continue
    [ -r $f ] || continue
    echo "printing file $f:"
    case $f in
        *.ps ) lpr $f;;
        *.txt) a2ps $f;;
        *) echo "don't know how to print $f"
    esac
    echo "done"
done
```

# Structures de contrôle et redirections

★ L'évaluation d'une structure `if`, `case`, `for`, `while` ou `until` s'effectue dans un sous-shell

★ On peut rediriger l'entrée et/ou la sortie d'un tel sous-shell

```
for file in *.c; do
    gcc -c $file
done 2> compil.error
```

★ On peut lier un tel sous-shell avec un tuyau

```
ls ~dan/images/*.gif | while read file; do
    [ -f $(basename $file) ] || cp $file .;
done
```

★ On peut exécuter un tel sous-shell en arrière plan

```
for file in *.c; do gcc -c $file; done &
```

# Exercices V.6

1. Écrivez la commande `mycal` qui améliore la commande `/usr/bin/cal` en autorisant alternativement de qualifier les mois à l'aide de l'un des mots `jan`, `feb`, `mar`, `apr`, `may`, `jun`, `jul`, `aug`, `sep`, `oct`, `nov` et `dec`, et ce indépendamment de la casse (par exemple, `jan`, `Jan`, `JAN`, `JaN`, `jAn`, etc. sont équivalents). La commande `mycal` doit se comporter comme `/usr/bin/cal` et en particulier admettre les mêmes options. Quand l'unique argument est le mois en lettre, l'année est l'année en cours.
2. Écrivez la commande `size2`, deuxième version de la commande `size1` (exercice III.3.3), qui affiche maintenant ses résultats en colonne (considérez la commande `column`).
3. Écrivez la commande `chlnk` qui prend en paramètre un ou plusieurs répertoires, et qui affiche tous les liens symboliques invalides appartenant récursivement à ces répertoires. Un lien symbolique est invalide s'il ne pointe pas vers un fichier ou un répertoire existant.
4. Écrivez la commande `chext1` qui prend en paramètre deux extensions `ext1` et `ext2` ainsi qu'un répertoire, et qui renomme tous les fichiers du répertoire de nom `xxx.ext1` en `xxx.ext2`. Si le nom du répertoire n'est pas fourni, la commande s'applique au répertoire courant.

# Les fonctions (1)

- ★ Syntaxe :

Pour définir la fonction de nom `fonction` :

```
fonction () {  
    commandes  
}
```

- ★ Même mécanisme de passage de paramètre qu'un script shell
- ★ Peut contenir des appels à des commandes quelconques, à d'autres fonctions ou à elle-même (fonction récursive)
- ★ L'instruction `return n` permet de sortir d'une fonction avec le code de retour `n`
- ★ Attention : l'évaluation de la commande `exit` depuis l'intérieur d'une fonction termine le **script shell englobant**

# Les fonctions (2)

- ★ Une fonction est interprétée dans le shell courant (pas de sous-shell)
  - ★ Une sorte d'alias avec paramètres
  - ★ Peut être définie dans le `.bashrc` pour être conservée
- ★ Une fonction peut être exécutée dans le shell interactif courant
  - ★ Modification **possible** des variables d'environnement
- ★ Une fonction peut être exécutée dans le sous-shell interprétant un script
  - ★ Attention : l'évaluation de la commande `exit` depuis l'intérieur d'une fonction termine le **script shell englobant**
- ★ Une fonction peut être récursive

# Les fonctions (3)

Une fonction pour se déplacer facilement dans un répertoire

```
cdd() {  
  
    for dir in $(find $HOME -name $1 -type d); do  
        echo -n "$dir ? "  
        read yes  
        if [ -n "$yes" ]; then  
            cd $dir; return 0  
        fi  
    done  
  
}
```

# Les fonctions (4)

- ★ Une fonction pour les erreurs

```
usage() {  
    echo "usage: $(basename $0) FILE" 2>/dev/stderr  
    exit 1  
}
```

- ★ Une fonction pour confirmer une saisie avec la question en paramètre

```
confirm () {  
    echo -n $1  
    while read ANSWER; do  
        case $ANSWER in  
            y|Y) return 0;;  
            n|N) return 1;;  
            *) echo -n "please answer y or n: ";;  
        esac  
    done }
```

# Exercices V.7

1. Écrivez la commande `move1`, clone de la commande `mv`, mais qui n'admet aucune option et exactement deux arguments. Bien sûr, il est interdit d'utiliser la commande `mv` !
2. Écrivez la commande `move2`, deuxième version de la commande `move1` précédente, qui est toujours un clone de `mv` sans option mais qui cette fois peut prendre plus de deux arguments.
3. Écrivez la commande `move3`, troisième version de la commande `move1`, commande identique à `move2` mais qui admet éventuellement l'option `-i`.
4. Écrivez la commande `copy1`, clone de la commande `cp`, mais qui n'admet aucune option et exactement deux arguments (un fichier régulier suivi d'un fichier ou un répertoire).
5. Écrivez la commande `copy2`, deuxième version de la commande `copy1` précédente, qui est toujours un clone de `cp` sans option mais qui cette fois peut prendre plus de deux arguments.
6. Écrivez la commande `copy3`, troisième version de la commande `copy1`, commande identique à `copy2` mais qui admet éventuellement les options `-i` et `-r`.

# La commande `expr` (1)

Une commande pour effectuer des calculs arithmétiques sur les entiers

## ★ Utilisation

```
expr 2 + 3  
incr=$(expr $incr + 1)
```

## ★ Erreurs fréquentes

```
expr 2 +3           : +3 est vu comme un unique argument  
=> expr 2 + 3  
expr 2 * 3          : * est un méta-caractère du shell  
=> expr 2 \* 3  
expr 2 * ( 3 + 5 ) : les parenthèses '(' et ')' sont  
                    interprétées par le shell  
=> expr 2 \* \( 3 + 5 \)
```

# La commande `expr` (2)

Une commande pour effectuer des tests ou des calculs sur les chaînes de caractères

- ★ `expr match chaine modèle` (ou `expr chaine : modèle`)  
Teste si `chaine` correspond à `modèle` :

```
expr match linux 'li*'
```

- ★ `expr substr chaine i n`  
Affiche la sous-chaîne de `chaine` de longueur `n` et commençant au caractère à l'indice `i` (le premier caractère est à l'indice 1)
- ★ `expr index chaine caractère`  
Affiche l'indice de la première occurrence de `caractère` dans `chaine`, ou 0 si `caractère` n'est pas présent dans `chaine`
- ★ `expr length chaine`  
Affiche la longueur de `chaine`

# Arithmétique entière

La notation `((...))` permet d'effectuer simplement des tests ou des calculs arithmétiques sur les entiers

★ Test arithmétique :

```
if (( x > 1 )) ; then
    ...
```

Le `$` devant la variable `x` n'est pas nécessaire et les caractères `<` et `>` perdent leur signification spéciale

★ Calcul (et substitution) arithmétique :

```
x=10          y=20          z=$(( x + ( 2 * y ) / 5 ))
```

On peut utiliser les caractères `+`, `*`, `/`, `(`, `)` selon leur signification arithmétique usuelle

# La commande `eval` (1)

Une commande pour évaluer la commande qui lui est passée en paramètre

- ★ `eval commande`

Evalue commande comme si elle avait été directement interprétée par le shell courant :

- ★ le shell applique le découpage en sous-commandes, les substitutions de variables et de commandes et la génération de noms de fichiers à `commande`
- ★ le shell applique le découpage en sous-commandes, les substitutions de variables et de commandes et la génération de noms de fichiers au résultat précédent et finalement interprète le tout
- ★ Affectation avec calcul de la variable affectée

```
x=y; eval $x=123; echo $y (y a la valeur "123")
```

# La commande `eval` (2)

## Une fonction qui modifie une variable

```
function chvar() {  
    eval $1="$2"  
}
```

Exemple : modification de la variable `PATH`

```
[hal@/home/bob] echo $PATH  
/bin:/usr/bin:/usr/local/bin  
[hal@/home/bob] chvar PATH '~/bin:$PATH'  
[hal@/home/bob] echo $PATH  
/home/bib/bin:/bin:/usr/bin:/usr/local/bin
```

# La commande `exec`

## Une commande à deux usages

### ★ `exec commande`

Le shell courant exécute la commande `commande` qui **devient** le shell courant

★ aucun intérêt dans un shell interactif (ne le testez pas)

★ à ne pas confondre avec l'option `-exec` de la commande `find`

### ★ `exec` et redirections

★ `exec < file, exec > file, exec 2> file`

Redirigent l'entrée, la sortie ou les erreurs du **shell courant** vers le fichier `file`

★ `exec 3<&0`

*Affecte* l'entrée standard au descripteur de fichier 3

# La commande `getopts`

Une commande pour traiter plus simplement les arguments d'un script

```
while getopts f:ri opt; do
  case $opt in
    r) echo "r present";;
    i) echo "i present";;
    f) echo "f present with argument $OPTARG";;
  esac
done

shift $(expr $OPTIND - 1)

echo "command arguments:"

for arg in $@; do
  echo " $arg"
done
```

# La commande `select`

**Une commande pour fabriquer simplement un menu de saisie**

```
select action in ajouter supprimer modifier
do
    echo "Vous avez choisi $action"
    break
done
case $action in
    ajouter) ...;;
    supprimer) ...;;
    modifier) ...;;
esac
```

# Exercices V.8

1. Écrivez la commande `myhead`, clone de la commande `head`, mais admettant seulement les deux options `-n` et `-q` et ne lisant que dans un ou plusieurs fichiers (et pas sur l'entrée standard). Pour simplifier l'écriture de `myhead`, on interdira la forme `myhead -15` au profit de `myhead -n 15`. Enfin, lorsque l'option `-n` n'est pas présente, le nombre de lignes à afficher sera la valeur de la variable d'environnement `MYHEAD_DEFAULT` si elle existe, ou bien `10` sinon.
2. Écrivez la commande `myuniq`, clone de la commande `uniq`, mais admettant uniquement les trois options `-c`, `-d` et `-u` et ne prenant aucun argument (l'entrée et la sortie de la commande sont l'entrée et la sortie standards).
3. Écrivez la commande `chext2`, deuxième version de la commande `chext1` (exercice III.4.4), commande identique à `chext1` mais qui admet éventuellement les options `-i` et `-r`. Si l'option `-i` est présente, la commande demande confirmation à l'utilisateur avant d'effectuer le changement d'extension. Si l'option `-r` est présente, la commande cherche les fichiers d'extension `.ext1` récursivement dans le répertoire.
4. Écrivez la commande `kproc` qui prend en argument des mots et qui tue interactivement tous les processus dont le nom contient un ou plusieurs de ces mots. Pour chaque processus, la commande affiche toutes les informations qui lui sont associées.

## **VI. Exercices de synthèse**

- ★ Gestion de sauvegardes**
- ★ Gestion d'une poubelle**
- ★ Un mécanisme d'annulation**

# Gestion de sauvegardes

1. Le but de cet exercice est d'écrire les deux commandes `backup` et `restore`, commandes permettant de gérer simplement des sauvegardes de fichiers et de répertoires.
2. La commande `backup` prend comme arguments un ou plusieurs fichiers ou répertoires, et en crée des copies. Par exemple, `backup xxxx` fabrique la copie `xxxx.bak`, que `xxxx` soit un fichier ou un répertoire. Si `xxxx` est un répertoire, son contenu est copié récursivement dans `xxxx.bak` mais les éléments qu'il contient ont le même nom relatif, à moins qu'on utilise l'option `-r`. Afin d'éviter les sauvegardes de sauvegardes de sauvegardes... il est interdit de sauvegarder un fichier de sauvegarde avec la commande `backup`.
3. La commande `restore` prend comme arguments un ou plusieurs fichiers ou répertoires qui ont été sauvegardés, et les remplace par leur copie. Après restauration, les copies sont effacées. Pour faciliter l'utilisation de la commande `restore`, on peut l'appeler indifféremment avec le nom du fichier à restaurer ou avec le nom de la sauvegarde. Ainsi, `restore xxxx` et `restore xxxx.bak` sont équivalents. La commande `restore` admet également l'option `-r`.
4. L'extension utilisée pour les fichiers de sauvegarde est la valeur de la variable `BACKUP_EXTENSION` ou bien `back` si cette variable n'est pas définie.

# Gestion d'une poubelle

Le but de cet exercice est de concevoir et d'écrire un ensemble de commandes pour gérer une *poubelle*. La poubelle est un répertoire dans lequel on déplace des éléments (fichiers ou répertoires) qu'on peut éventuellement récupérer plus tard. La poubelle peut être vidée définitivement, après quoi les éléments qui s'y trouvaient ne sont plus accessibles. Les commandes à réaliser ne se limitent pas à déplacer les fichiers ou les répertoires simplement dans le répertoire poubelle. Par exemple, on peut mettre à la poubelle consécutivement deux fichiers de même nom relatif, ou plus délicat, de même nom absolu (mais néanmoins différents, car existant à des moments différents) et vouloir récupérer chacun d'eux ! L'ensemble des commandes à développer comprend au minimum les commandes :

0. `trash`, qui met à la poubelle un ou plusieurs fichiers ou répertoires
1. `intrash`, qui liste de manière lisible et structurée le contenu de la poubelle
2. `untrash`, qui permet de récupérer simplement un fichier ou un répertoire contenu dans la poubelle (à partir des informations fournies par `intrash`)
3. `cleantrash`, qui permet de vider tout ou partie de la poubelle en fonction de certains paramètres et options (par exemple, en fonction du temps depuis lequel le fichier ou le répertoire est dans la poubelle)

# Un mécanisme d'annulation (1)

Certaines commandes Linux modifient leur environnement, comme les commandes `rm`, `cp` ou `mv` pour ne citer que les plus courantes. Il serait utile de disposer d'un mécanisme d'*annulation* de l'exécution d'une telle commande qui permette de revenir à l'état initial (d'avant l'exécution de la commande). Le but de cet exercice est de réaliser un ensemble de commandes pour gérer l'*annulation* de certaines commandes.

Le principe d'annulation proposé repose sur une commande très particulière, la commande `undo`. Par exemple, si on veut pouvoir annuler l'effet de la commande `mv xxx yyy`, il faut que la commande `mv` elle-même produise avant de s'exécuter la commande annulante `mv yyy xxx`. La commande annulante peut être placée toujours dans le même script, la commande `undo` ! Ainsi, les effets de la commande `mv xxx yyy` peuvent être annulés par la commande `mv yyy xxx`, à condition que le fichier `yyy` n'existe pas au moment de l'exécution de la commande `mv`. Dans le cas contraire, le commande `mv` doit effectuer en premier une sauvegarde du fichier `yyy`, par exemple avec `mv yyy /tmp/yyy`, et les commandes annulantes (placées dans le script `undo`) deviennent `mv yyy xxx` suivie de `mv /tmp/yyy yyy`. Pour résumer, on dira que c'est la commande à (éventuellement) annuler qui, connaissant ses paramètres, est à même de produire les commandes susceptibles d'annuler ses effets !

# Un mécanisme d'annulation (2)

Notez que **seule** la dernière commande exécutée dans le shell interactif courant peut être (éventuellement) annulée. Aussitôt une nouvelle commande évaluée, si cette dernière n'est pas une commande annulable, la commande `undo` ne doit rien faire si ce n'est délivrer un message d'erreur.

Vous devez développer au minimum les trois nouvelles commandes `rm`, `cp` et `mv`, versions annulables des commandes initiales du même nom. Idéalement, ces commandes doivent admettre les mêmes options et les mêmes paramètres que les commandes initiales correspondantes, mais vous pouvez vous limiter aux options essentielles (`-i` et `-r` par exemple).

La commande `undo` doit admettre au minimum les options `-v` et `-n` :

- ★ `-v` (verbose) indique que la commande `undo` affiche les opérations intermédiaires qu'elle réalise pour annuler la dernière commande tout en les effectuant
- ★ `-d` (dry run) affiche les opérations intermédiaires qu'elle réaliserait pour annuler la dernière commande, mais sans les réaliser ! Remarquez qu'on doit pouvoir exécuter la commande `undo` après qu'on ait exécuté une commande annulable suivie d'un nombre quelconque de `undo -d` !

# VII. Le monde Linux

- ★ **Le projet GNU et la licence GPL**
- ★ **Les distributions Linux**
- ★ **Linux et ses concurrents**

# Le projet GNU

## GNU = GNU is Not Unix

- ★ Projet de réaliser un système à la Unix entièrement libre
- ★ Lancé en 1984 par Richard Stallman, un chercheur du MIT, à une époque où les sources de Linux n'étaient plus libres d'accès
- ★ Composants initiaux: compilateur C (gcc), make (GNU make), Emacs, bibliothèque C (glibc), outils de base (ls, cp ...)
- ★ Cependant, en 1991, le projet GNU n'avait toujours pas de noyau et tournait sur des Unix propriétaires

# Les Logiciels Libres

- ★ GNU, Linux et de nombreux autres programmes sont des *Logiciels Libres*
- ★ Le *Logiciel Libre* fournit à son utilisateur les 4 libertés suivantes :
  - ★ La liberté d'exécuter le programme, pour quelque but que ce soit
  - ★ La liberté d'étudier son fonctionnement, et de l'adapter à ses besoins
  - ★ La liberté de redistribuer des copies pour aider autrui
  - ★ La liberté d'améliorer le programme, et de partager ses améliorations avec autrui
- ★ Voir <http://www.gnu.org/philosophy/free-sw.fr.html>

# La licence GPL

- ★ Les licences *Copyleft* utilisent les lois sur le copyright pour permettre aux auteurs d'exiger que toute version modifiée reste aussi un logiciel libre
- ★ La licence GNU GPL (GNU General Public License) exige que toutes modifications et travaux dérivés soient aussi publiés sous licence GPL
  - ★ Ne s'applique qu'aux logiciels publiés
  - ★ Tout programme utilisant du code sous la GPL (statiquement ou même dynamiquement) est considéré comme une extension ce code
- ★ Pour plus de détails :
  - ★ Copyleft: <http://www.gnu.org/copyleft/copyleft.html>
  - ★ Questions / réponses sur la GPL: <http://www.gnu.org/licenses/gpl-faq.html>

# GNU Linux

- ★ Noyau libre semblable à un noyau Unix, conçu par Linus Torvalds en 1991
- ★ Le système complet se repose sur les outils GNU : bibliothèque C, gcc, binutils, fileutils, make, emacs...
- ★ Le système complet est donc appelé “GNU / Linux”
- ★ Très tôt partagé comme Logiciel Libre (Licence GPL), ce qui attira des contributeurs et des utilisateurs de plus en plus nombreux
- ★ Depuis 1991, connaît une croissance supérieure à tout autre système d'exploitation (Unix et autres)

# Distributions GNU Linux (1)

- ★ Se chargent de publier un ensemble cohérent de versions compatibles du noyau, de la bibliothèque C, des compilateurs, des outils
- ★ Les outils sont disponibles sous forme de *paquetages* qui peuvent facilement être installés, supprimés ou mis à jour. Les dépendances entre outils sont gérées plus ou moins automatiquement
- ★ Distributions commerciales: incluent de l'assistance technique. Le code source est libre, mais les binaires ne sont pas libres d'accès
- ★ Distributions communautaires: sources et binaires sont librement disponibles. Ne comprennent pas d'assistance technique

# Distributions GNU Linux (2)

- ★ Vous permettent d'installer Linux dans un emplacement libre sur votre disque dur, tout en gardant Windows (“double démarrage”)
- ★ Ont une interface très conviviale qui peut détecter automatiquement détecter la plupart des matériels (pas de pilote à installer)
- ★ Vous permettent de choisir les types d'applications à installer
- ★ Fournissent une interface de configuration conviviale
- ★ Distributions recommandées pour les débutants :  
Fedora Core, Mandriva ou (K)Ubuntu

# Distributions commerciales

- Red Hat: <http://www.redhat.com/>

La plus populaire. Fiable, sûre, conviviale et facile à installer, prise en charge par tous les fournisseurs de logiciel et de matériel.



- Suse (Novell): <http://www.suse.com/>

L'alternative principale. Facile à installer et conviviale. Pas encore d'infos sur sa stabilité. Pas encore prise en charge par tous les fournisseurs.



- Mandriva: <http://www.mandriva.com/>

Conviviale, facile à installer, plus innovante, mais moins stable (peut-être davantage depuis l'introduction des pré-versions communautaires). Cible principalement les utilisateurs individuels. Peu prise en charge par les fournisseurs.



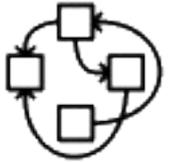
# Distributions communautaires

- Debian: <http://debian.org/>  
Très stable et sûre, mais plus difficile à configurer et à installer. Peu conviviale pour les utilisateurs. Version stables peu fréquentes (tous les 2 ou 3 ans). Recommandée pour les serveurs
- Fedora Core: <http://fedora.redhat.com/>  
Stable, sûre, conviviale, facile à installer. Sortie fréquente de nouvelles versions complètes
- Mandriva Community: <http://www.mandriva.com/>  
Facile à installer, sûre, conviviale, sortie fréquente de versions complètes, mais moins stable (pas assez de tests et de prise en compte des retours des utilisateurs et des testeurs)
- (K)Ubuntu: <http://www.ubuntulinux.org/> et <http://www.kubuntu.org/>  
Très facile à installer et facile à configurer. Une nouvelle version (promise) tous les six mois. Une distribution très à la mode !



# Autres systèmes Unix libres

GNU / Hurd: <http://www.gnu.org/software/hurd/hurd.html>



- ★ Outils GNU avec le Hurd, le micro-noyau de GNU
- ★ De plus en plus mûr, mais pas encore assez pour être utilisé par tous. Jusqu'en 2004 surtout utilisé par ses développeurs eux-mêmes.

## Famille BSD

★ FreeBSD: <http://www.freebsd.org/>

Système BSD puissant, multi-plateforme, sûr et populaire.



★ OpenBSD: <http://openbsd.org/>

Système BSD puissant, multi-plateforme, sûr et populaire.

Construit pour une fiabilité et une sécurité extrêmes. Populaire pour serveurs sur Internet.



★ NetBSD: <http://netbsd.org/>

Distribution BSD dont le but est d'être extrêmement portable (par exemple disponible sur ARM)

# Linux et ses concurrents (1)

## Linux est une alternative sérieuse aux systèmes commerciaux

### Sécurité

- ★ Sans virus

La plupart des virus sont conçus pour tirer parti des failles de sécurité de Windows et n'ont aucun effet sur GNU / Linux.

- ★ Décourage les pirates

Même si vous êtes connecté en permanence à Internet, votre système attire moins les pirates.

- ★ A l'épreuve des virus

Même si vous exécutez un virus compatible avec Linux, il n'aurait pas la permission de modifier le système.

- ★ A l'épreuve des erreurs

Les utilisateurs ne peuvent ni toucher au système ni aux fichiers d'un autre utilisateur. Ils ne peuvent endommager que leurs propres fichiers.

# Linux et ses concurrents (2)

## Respect de la vie privée

- ★ Votre système ne va pas discrètement recueillir des informations sur les films ou les sites internet que vous préférez

## Convivialité

- ★ Vos programmes sont conçus pour des utilisateurs par des utilisateurs. Ils sont mieux susceptibles de satisfaire vos besoins

## Liberté

- ★ Les données que vous créez vous appartiennent pour toujours. Elles ne sont pas prisonnières d'une application propriétaire à travers un format propriétaire (parfois breveté)
- ★ On peut facilement contacter les développeurs pour leur suggérer de nouvelles fonctionnalités
- ★ Vous êtes libres d'aider votre entourage en partageant vos programmes avec lui
- ★ Vous êtes libres d'améliorer ces programmes vous mêmes

# Linux et ses concurrents (3)

## **Vous pouvez utiliser Linux pour :**

- ★ La bureautique : traitement de texte, tableur, présentations
- ★ Internet : navigation et courrier électronique
- ★ Le multimédia : vidéo, son et graphiques (y compris appareils photo numériques)
- ★ Votre activité professionnelle (développement et programmation)

## **Vous pouvez utiliser un autre système d'exploitation pour :**

- ★ Les jeux : la plupart des jeux grand public ne sont encore conçus que pour Windows ou Mac
- ★ Utiliser des logiciels propriétaires spécifiques ou des cdroms éducatifs
- ★ Utiliser du matériel non encore pris en charge sous Linux

# Essayer Linux

**Kaella** est un cdrom Linux sans installation

<http://kaella.linux-azur.org/>

- ★ Entièrement en français
- ★ Charge Linux en mémoire, rien n'est installé sur votre disque dur
- ★ D'étonnantes capacités de reconnaissance du matériel
- ★ Plus de 2 Go d'applications disponibles
- ★ Vous pouvez accéder à vos fichiers Windows, les ouvrir et même les modifier avec des applications sous Linux
- ★ Une excellente façon d'essayer et de montrer GNU / Linux
- ★ Permet d'effectuer une installation permanente sur votre disque dur

# VIII. Epilogue

- ★ **Ressources internet sur Linux**
- ★ **Licence et historique du cours**

# Ressources Linux sur internet

## Une petite sélection de ressources internet diverses sur Linux

- ★ Le site officiel de Linux  
<http://www.linux.org/>
- ★ Le site officiel GNU  
<http://www.gnu.org/>
- ★ Le site de Linux-France  
<http://www.linux-france.org/>
- ★ La page web du Linux Journal  
<http://www.linuxjournal.com/>
- ★ Toutes les nouveautés sur Linux  
<http://www.linuxcentral.com/>
- ★ Un site sur les distributions Linux  
<http://distrowatch.com/>
- ★ Un site d'informations sur Linux  
<http://www.toolinux.com/>
- ★ Un site de forums sur Linux  
<http://www.linuxquestions.org/>
- ★ Un site dédié aux débutants Linux  
<http://www.delafond.org/survielinux/>
- ★ Un site généraliste sur Linux  
<http://linuxfr.org/pub/>
- ★ Le site de Linux-Azur  
<http://www.linux-azur.org/>
- ★ Le site de la conférence Solutions Linux  
<http://www.solutionslinux.fr/>

# Licence

© 2005, 2006, 2007, 2008, 2009, 2010 Marc Gaëtano  
[gaetano@polytech.unice.fr](mailto:gaetano@polytech.unice.fr)

© 2004, Michael Opdenacker  
[michael@free-electrons.com](mailto:michael@free-electrons.com)

© Le pingouin  est dû à Daniel Joos ([daniel@joosweb.de](mailto:daniel@joosweb.de))

Ce document est publié selon les termes de la Licence de Documentation Libre GNU, sans parties non modifiables.

Les auteurs vous accordent le droit de copier et de modifier ce document pourvu que cette licence soit conservée intacte.

Voir <http://www.gnu.org/licenses/fdl.html>

# Historique du cours

- ★ Les contributions de la version initiale proviennent de M. Opdenacker. Détails disponibles sur <http://free-electrons.com/docs/>

- ★ 28 sep. 2004, première publication.

- ★ 20-24 sep. 2004, première session pour Atmel, Rousset (France)

Les contributions de la version actuelle proviennent de M. Gaëtano.

Des ressources diverses associées à ce cours ont disponibles sur

<http://www-local.polytech.unice.fr/~gaetano/>

- ★ 5 sep. 2005, version 1.0

- ★ 4 sep. 2006, version 1.1

- ★ ...

- ★ 9 sep. 2010, version 1.5