Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

15 février 2012

Partie #1

Organisation du module

Les enseignements

- 5 cours de 1h30 : Présentation générale du système d'exploitation Linux,
- 5 TP de 3h00 : Mise en pratique des commandes Linux en salle machines,
- et du travail personnel . . .

Votre présence est obligatoire

- Contrôle des présences.
- Rapport des absences.

L'évaluation

- Une composition à la fin du module (sur papier et/ou sur papier).

Plan

Définition

Definition

"Un ordinateur est une machine électronique programmable capable de réaliser des calculs logiques sur des nombres binaires."

C'est une machine

Le fonctionnement d'un ordinateur est basé sur une architecture matériel (processeur, support de stockage, interfaces utilisateurs, connexion, ...) dont le fonctionnement est soumis aux lois de la physique.

C'est une machine programmable

Cette machine est capable de remplir des tâches différentes selon les instructions qui lui sont adressées. Ces instructions, rédigées sous forme de programmes par les informaticiens, sont traitées en fin de course par le matériel de l'ordinateur.

Interaction avec le matériel

Heureusement, la plupart du temps, l'informaticien n'a pas a interagir directement avec le matériel. Pour traiter avec les composants, tous les ordinateurs modernes disposent d'une couche logicielle appelée Système d'Exploitation. Cette couche est en charge de faire la passerelle entre l'informaticien, ses outils, les programmes qu'il développe et, les composants et leur fonctionnement.

Les interfaces

La forme classique

- Un ordinateur est classiquement composé d'une unité centrale et de périphériques matériel (écran, clavier, souris, disques durs, imprimantes/scaner, ...).
- Les interfaces permettent l'interaction avec l'environnement (utilisateurs ou autres).

Des formes très variées

- Les ordinateurs modernes sont multiformes,
- Ils remplissent des tâches très variées.

Points communs et différences

= Matériel

Des capacités de calcul	CPU et/ou GPU
De la mémoire	RAM, Disque dur, ...

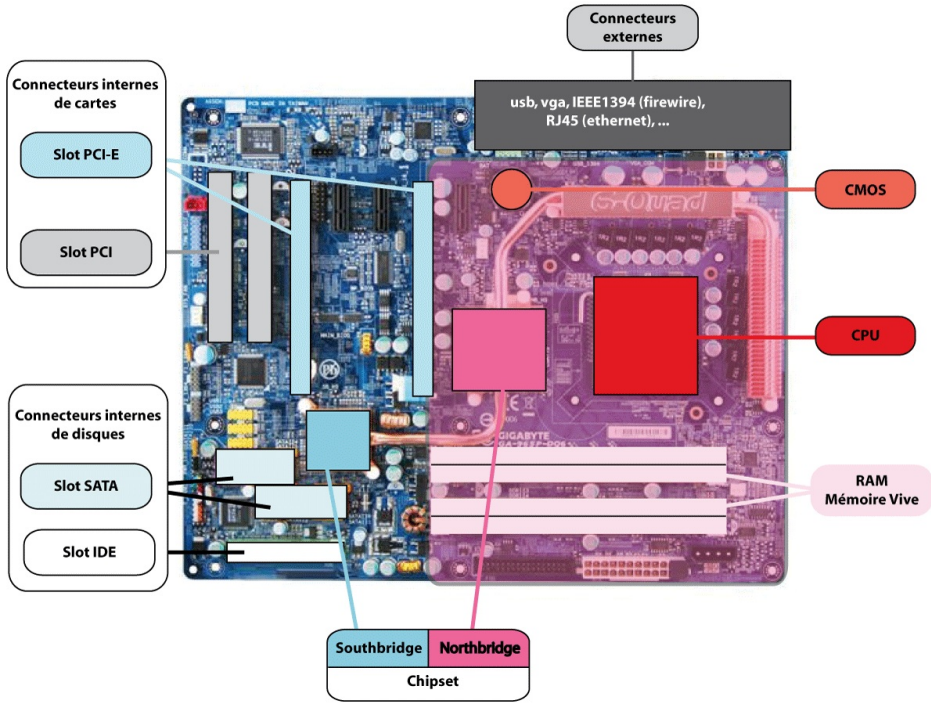
= Logiciels

Pour dialoguer avec le matériel	Système d'exploitation, Firmware
Pour accomplir ses tâches	logiciels, programmes, ...

≠ Périphériques

Interfaces	Connexions réseau, écrans, claviers, ...

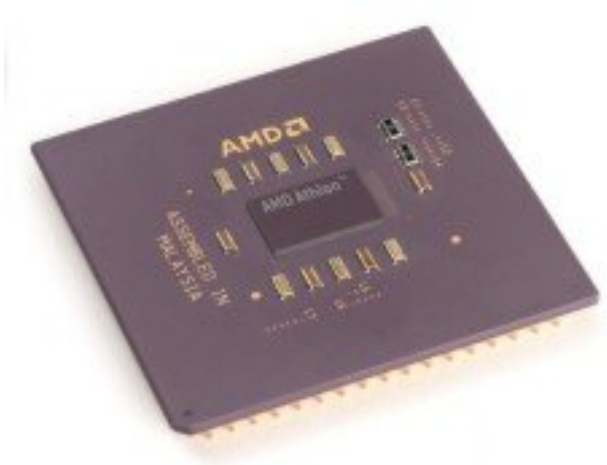
La carte mère



La carte mère est l'élément central d'un ordinateur.

- C'est l'élément sur lequel sont assemblés et mis en relation tous les composants matériel,
- Elle permet à tous ses composants de fonctionner ensemble efficacement.

Les unités de calcul



CPU - Central Processing Unit

- C'est une puce qui traite des instructions élémentaires en réalisant des calculs binaires,
- Fréquence 3GHz ($3x10^9$ de cycles par secondes).

GPU - Graphics Processing Unit

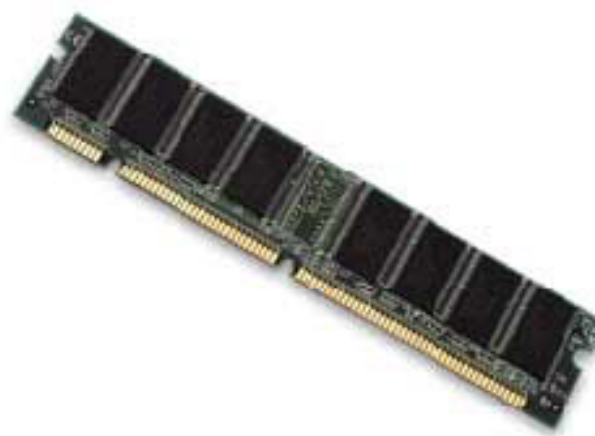
C'est une puce placée sur les cartes graphiques

- Elle prend en charge les nombreux calculs de rafraichissement des images 3D,
- Une carte graphique moderne peu compter une grande quantité de ces puces.

Les mémoires et les supports de stockages

RAM : Random Access Memory

- Mémoire volatile : maintenue par une tension électrique,
- Accès rapide,
- Taille limitée car assez chère.



ROM : Read Only Memory

- Mémoire non-volatile maintenue par une conception physique,
- Taille limitée car très chère,
- Contient les instructions d'amorçage du système.

Disque Dur, clef-usb, ...

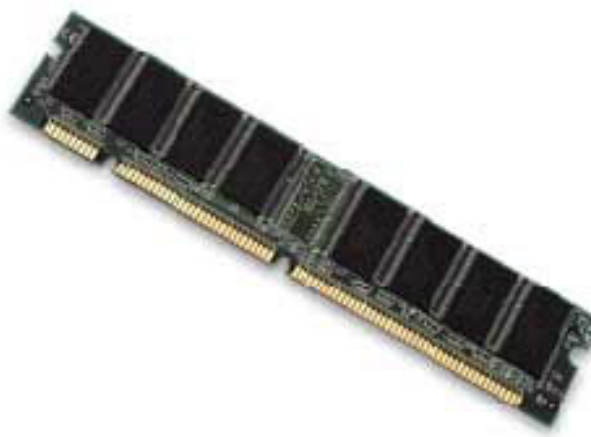
- Mémoire non-volatile (enregistrement magnétiques le plus souvent),
- Accès lent,
- Taille très grande (support de stockage de masse), beaucoup moins chère.

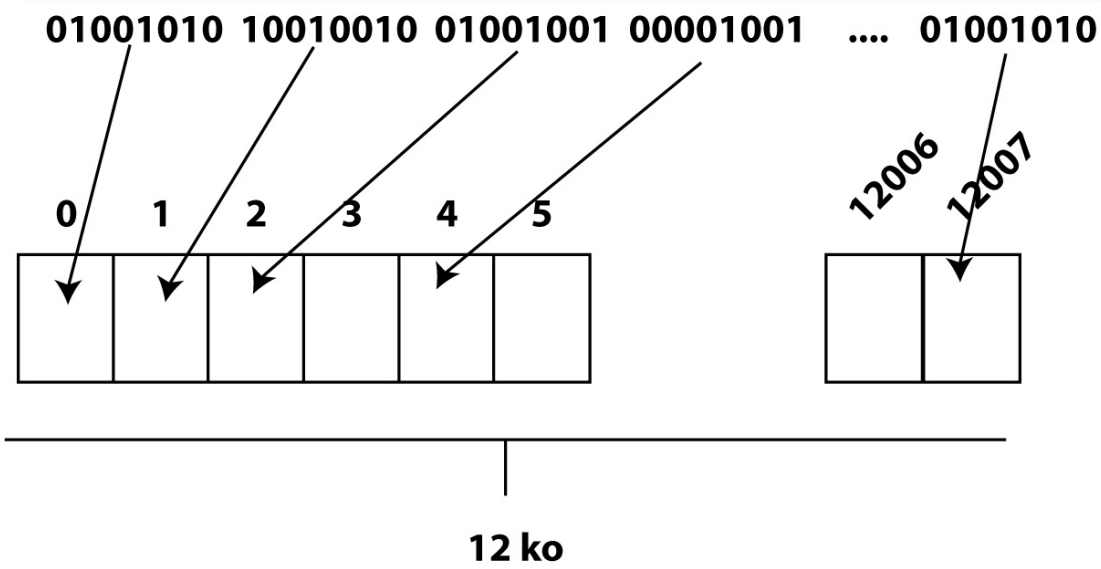
Les mémoires et les supports de stockages

Organisation de la mémoire

Les ordinateurs réalisent des calculs logiques sur des données binaires

- Les données et les instructions sont stockées sous forme de blocs repérés par une adresse,
- Les blocs contiennent une information binaire organisée en octet. Chaque octet contient 8 bits d'information qui sont lus comme une suite ordonnée de 0 ou de 1 ou de Vrai et de Faux.
- Un octet peut prendre $2^8 = 256$ valeurs différentes.





Les périphériques

Des composants externes

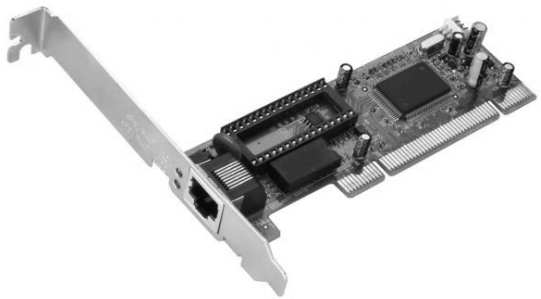
En fonction de leur tâche, de nombreux composants *ad hoc* peuvent être *greffés* sur la structure de base précédemment décrite. Par exemple :

- Ordinateur de Maison : Écran, souris, imprimante, scanner, joystick, modem, ...
- Ordinateurs de bord : Sondes, actioneurs, ...,
- Télephone : Antenne, récepteurs, ...,
- Robot médical : Interface haptique, bras mécaniques, ...

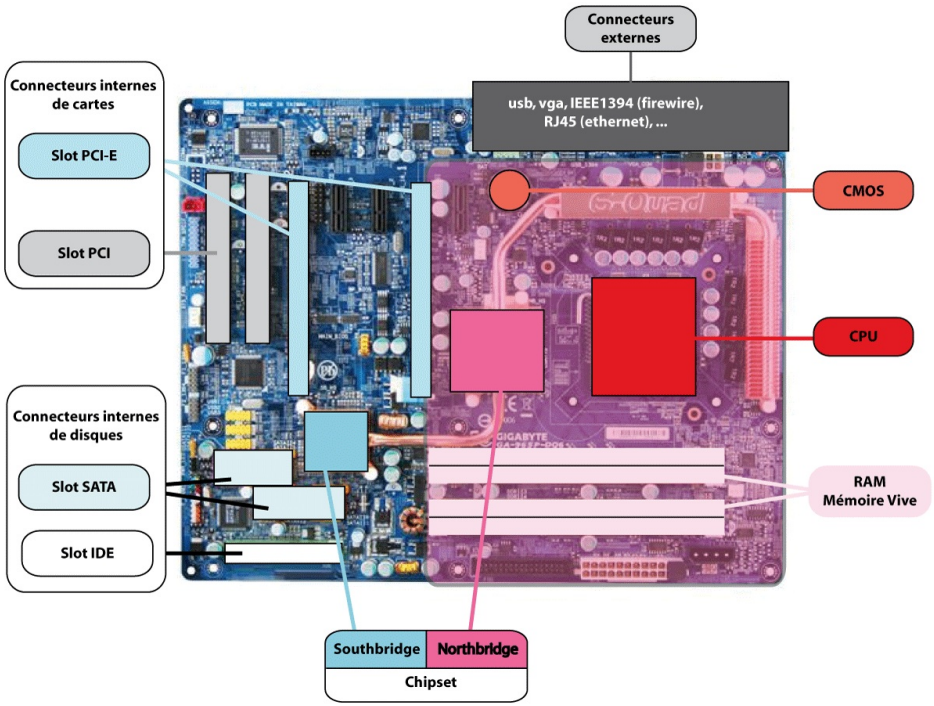
Des composants internes

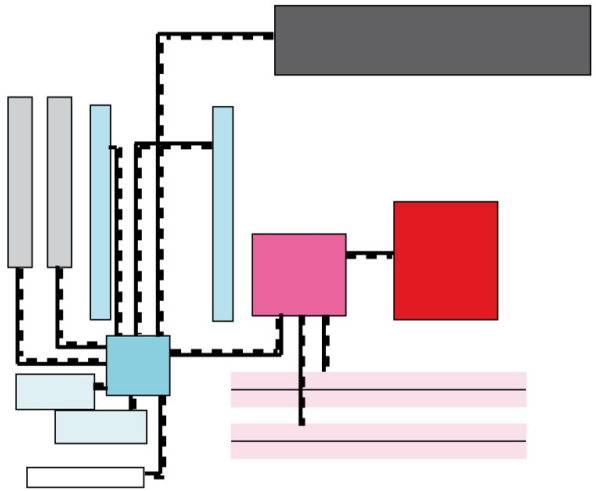
En fonction des possibilités des cartes mères plusieurs types de composants peuvent être ajoutés :

- Cartes vidéo, Cartes son, disques durs internes, lecteurs, ...
- Cartes d'acquisition ou de pilotage de périphériques, ...



Les bus





La carte mère intègre les bus.

- Les bus sont des unités physiques qui assurent le transport efficace de l'information entre les différents composants connectés à la carte mère,
- La largeur (8, 16, 32 64 bits) et la fréquence ($10^2 - 10^3$ MHz) des bus règlent le débit d'information entre les composants. Cela conditionne donc fortement l'efficacité d'une configuration matériel.

L'horizon matériel

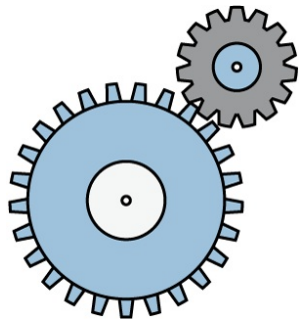
Interaction avec le matériel

- Heureusement le programmeur ou l'utilisateur n'interagit pas directement avec le matériel (sauf pour remplacer une pièce défectueuse ou connecter un nouveau matériel ...). Le dialogue avec l'architecture matériel est l'affaire de programmes dédiés.
- Plusieurs couches logicielles existent entre le matériel et l'utilisateur : les *firmwares*, le noyau du système et les outils et programmes du système d'exploitation.
- La plupart des logiciels que vous serez amené à développer n'interagiront qu'indirectement avec le matériel par le filtre des librairies système.

Haut Niveau →

- Logiciel,langages de programmation, ...
- C'est le domaine de l'informatique et des informaticiens

Une interface : Le système → d'exploitation



Bas niveau

- *Firmwares*, exécution des instructions machine, ...
- C'est le domaine de la physique et des électroniciens.

Plan

Le système d'exploitation

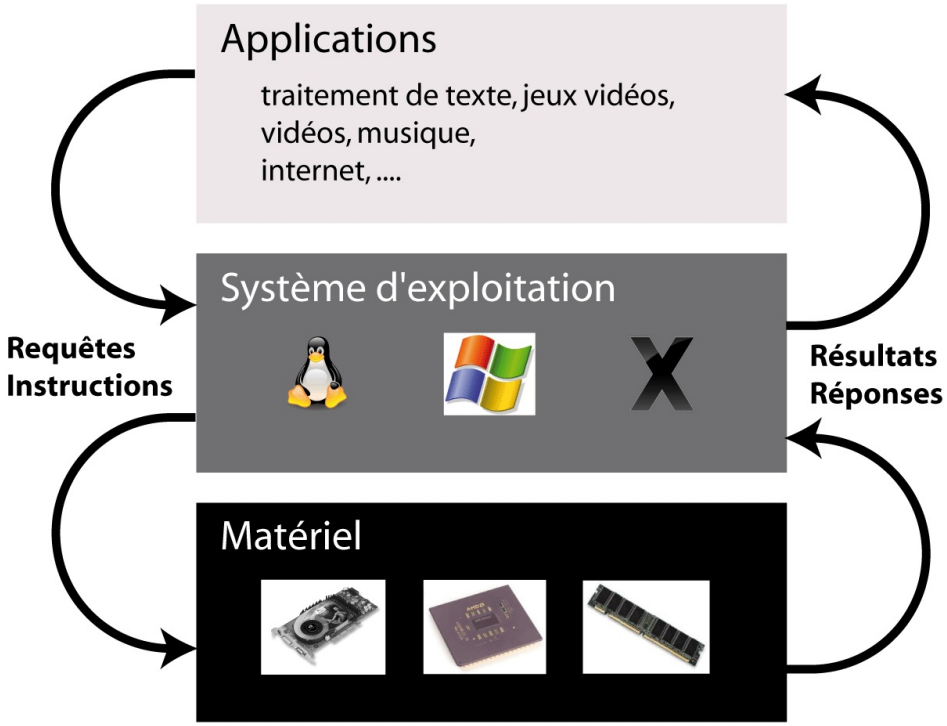
Le système d'exploitation permet de développer des programmes sans tenir compte de la complexité physique de la machine. Le programme utilise des fonctionnalités standardisées d'accès aux ressources matériel.

Côté Système

- coordonne l'utilisation de ces ressources (ex. : temps CPU accordé à chaque processus, allocation mémoire, ...),
- assure la maintenance et la fiabilité du système (ex. : gestion des fichiers, de la sécurité informatique, ...)
- ...

Côté utilisateur

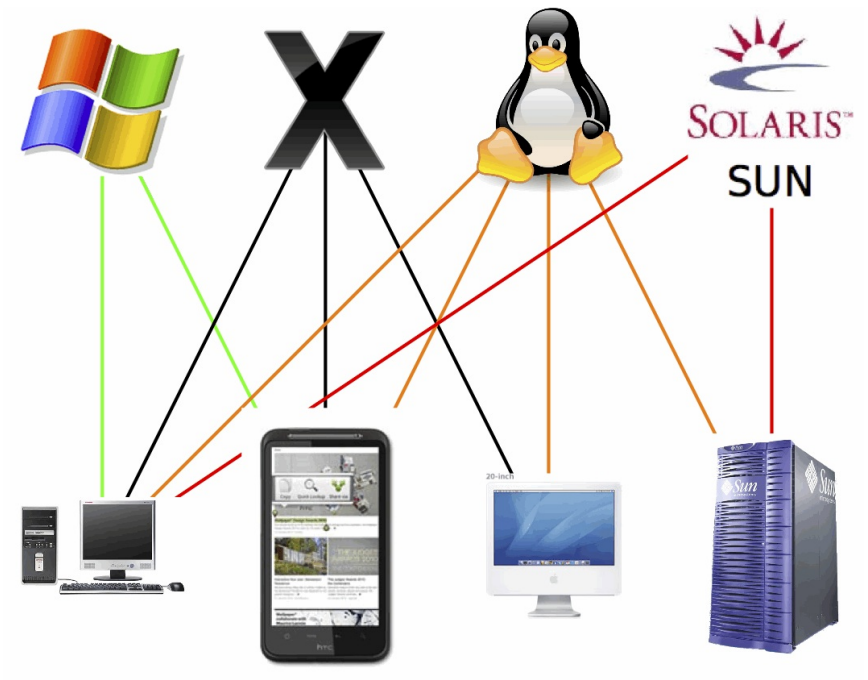
- facilite l'accès et l'utilisation des ressources matériel,
- propose un interface de programmation permettant d'utiliser ces matériels
- ...



Les différents systèmes d'exploitation

Beaucoup d'OS différents existent :

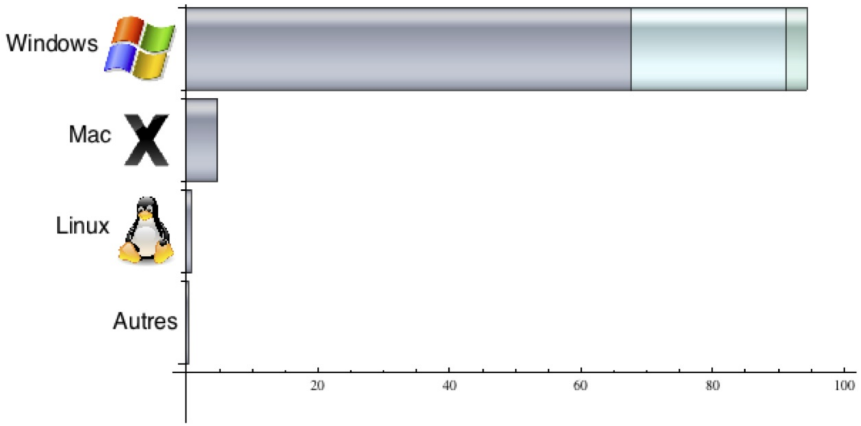
Chaque architecture matériel demande un système d'exploitation adapté. Certain systèmes d'exploitation sont plus souples et prennent en charge des architectures matériel multiples.



Deux OS se distinguent :

Windows est le système d'exploitation le plus utilisé, et Linux est le système d'exploitation le plus souple.
Statistiques au 5 janvier 2011 :
`http://gs.statcounter.com/`

- 95% des ordinateur utilisent Windows,
- il existe plus de 600 Système Linux...



Les différents systèmes d'exploitation

Linux

- Non propriétaire : Gratuit le plus souvent
- Ouvert : sources disponibles
- Flexible : sources modifiables
- Puissant : Programmable
- Communauté active : entraide des utilisateurs
- Plus complexe : pour les informaticiens (interface de programmation optimisées)

Windows©

- Propriétaire : Payant
- Sources non disponibles
- Sources non modifiables
- Plus difficilement programmable
- Communauté active : nombreux utilisateurs
- Plus ergonomique : pour les utilisateurs (interfaces d'utilisation optimisées)

Linux un système puissant en constante évolution

Depuis une dizaine d'année, Linux a beaucoup évolué. La plupart des distributions proposent des systèmes d'installation automatisés, des outils de bureautique ressemblant aux suites commerciales. Il bénéficie en outre d'une sécurité accrue à l'heure des virus et autres failles de sécurité.

Plan

Un peu d'histoire

GNU-Linux

- Le système GNU-Linux est la rencontre d'une technologie, le noyau Linux et d'une philosophie de développement et de diffusion. C'est un système au développement collaboratif (par une communauté) qui est distribué librement et permet l'utilisation de tous les logiciels libres développés pour son architecture.
- Le noyau Linux est historiquement une version libre du système UNIX développé initialement par le Finlandais Linus Torvalds à partir du début des années 1990.
- Le projet GNU est celui du développement collaboratif et libre d'un système d'exploitation libre initié par Richard Stallman en 1983.

Ahjourd'hui

- C'es un système très largement diffusé et utilisé sur lequel ont été développées plusieurs distributions (qui sont des suites logicielles qui accompagnent le noyau).
- Initialement confidentiel et réservé à des spécialistes avec des interfaces rudimentaires, il est aujourd'hui toujours plus ergonomique et automatisé pour les non spécialistes, mais laisse les outils et interfaces de bas niveau disponibles au plus grand nombre.
- On notera par exemple l'existence de nombreuses interfaces graphiques *Bureaux* (GNOME, KDE, ...) de nombreux paquetages pré-compilées, de nombreux outils d'administration et de services (protocoles, ...)

À l'IUT : Gentoo

Une distribution téléchargeable

`http://www.gentoo.org/`
`http://fr.gentoo-wiki.com/wiki/Accueil`

Pour ce cours

- Les concepts abordés dans ce module sont généraux.
- Il pourront être testés sur tous les systèmes Linux (avec de très faibles variantes).
- Il vous est possible d'installer une version de Linux sur votre ordinateur personnel (installation ou version Live) pour votre pratique personnelle et la préparation de l'examen.
- Une pratique régulière devrait vous assurer une bonne note à peu de frais. . .

Pour vous préparer à l'examen

Il vous est possible :

- d'utiliser Linux dans les salles machines,
- d'utiliser Linux via le service de bureaux virtuels *via* le portail de l'université : `https://portail.cevif.univ-paris13.fr/`
- d'installer une version de Linux sur votre ordinateur personnel (installation ou version Live).

Un système Multi-Utilisateurs

Des utilisateurs et des droits

- Chaque personne accédant au système est identifiée par un **nom d'utilisateur** (*i.e. login*) et un mot de passe (*i.e. password*).
- Chaque utilisateur bénéficie de permissions : exécution de certains programmes, lecture de certaines données, écriture de fichiers dans une limite de taille et dans seulement certains répertoires.
- Chaque utilisateur bénéficie d'un espace de travail réservé sur le disque. Cet espace de travail est un répertoire de l'arborescence dans lequel l'utilisateur à tous les droits : il peut y créer des sous-répertoires, y écrire des fichiers, y installer des programmes et applications. Toutes ses données et préférences personnelles y sont regroupées.
- Ce répertoire est appelé "Répertoire Personnel" ou *"Home Directory"*. Il est en général placé dans un répertoire qui s'appelle `/home/` et porte le nom de l'utilisateur : `/home/nom_utilisateur/`.

Superutilisateur - Root

- certains utilisateurs ont des permissions étendues pour administrer le système et effectuer des opérations interdites à l'utilisateur normal.
- l'utilisateur `root` a tous les droits dans le système (ex. : il peut changer les permissions de n'importe quel fichier, il fixe les noms d'utilisateur et les mots de passe, il peut installer des programmes et librairies dans les répertoires système, ...)

Identification en 2 étapes

Étape #1

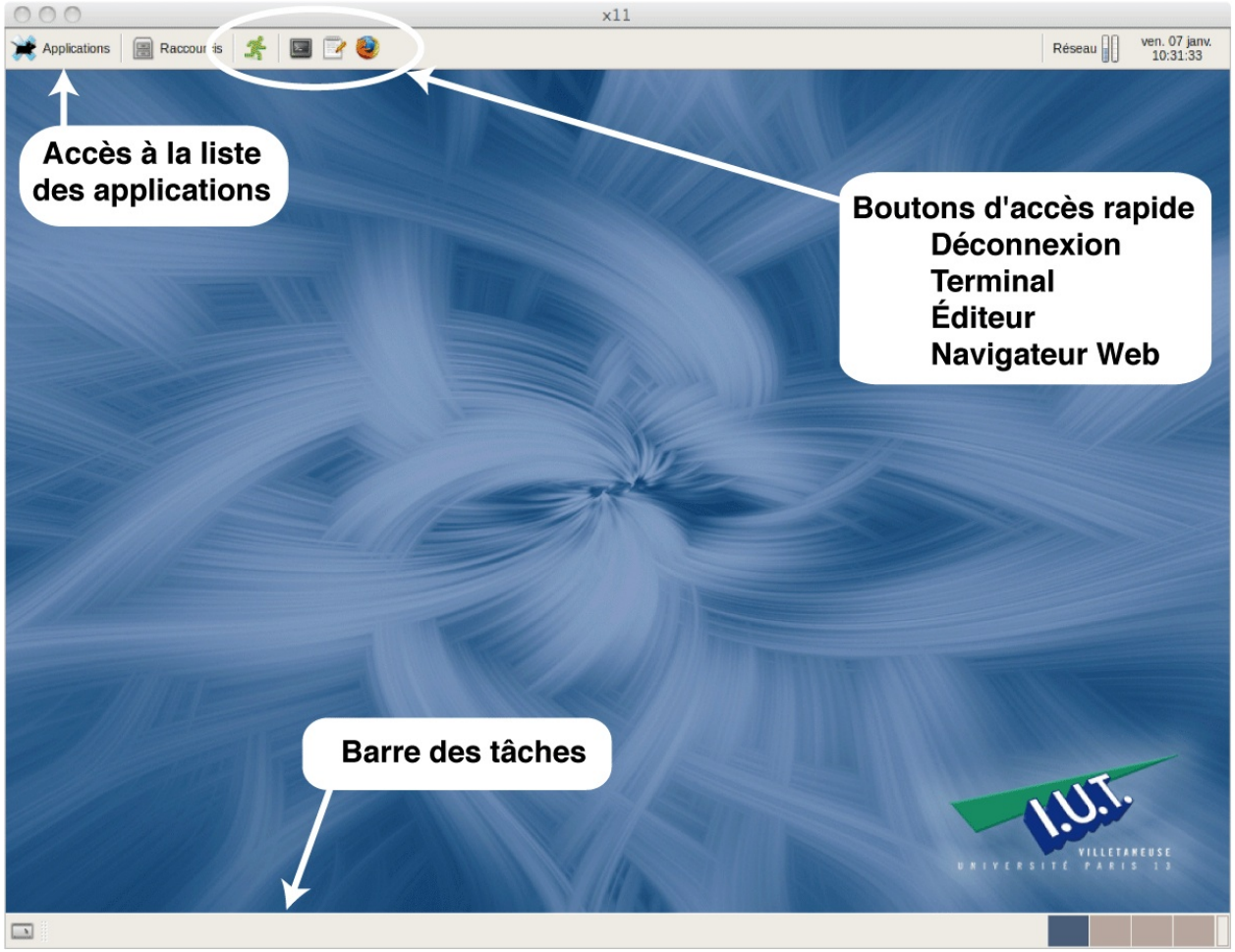
S'identifier en donnant au système son nom d'utilisateur

Identification en 2 étapes

Étape #2

Valider son identité avec le mot de passe

Accès au système



Le bureau GNOME

Parmi les différents environnements graphiques existants, vous utiliserez l'environnement GNOME (`http://www.gnomefr.org/`).

Les logiciels disponibles

Les suites bureautiques

- Les suites bureautiques proposent les fonctionnalités grand public de traitement de texte, de tableur, de présentation, de dessin.
- Plusieurs suites gratuites existent en libre accès sous linux
  - CalligraSuite (`http://www.calligra-suite.org/`)
  - OpenOffice (`http://fr.openoffice.org/`)
  - ...

Les programes dédiés

- Navigateur Web, Client de messagerie, comme sous d'autres OS, de nombreuses solutions existent.
  - Firefox, Opera, Konqueror, ...
  - Thunderbird, KMail, ...
- Des logiciels parmi les plus puissants :
  - Manipulation et création d'images : GIMP, ImageMagick, ...
  - Modélisation 3D : Blender, ...

De nombreuses micro-application ou programmes

- De nombreux programmes de conversion de format, de communication et de téléchargement existent en ligne de commande ...

Distribution et accès aux logiciels

Licences libres (open source)

Elles permettent de :

- d'utiliser le logiciel,
- d'étudier et de modifier les sources,
- de redistribuer les sources, modifiées ou non.

Licences Propriétaires

Elles restreignent un ou plusieurs des droits listés *supra*.

Gratuit ne signifie pas libre

Certains logiciel gratuits sont des logiciels propriétaires).

Copyleft©vs Copyleftↄ

Elles permettent de : Distribué en Copyleftↄ, les sources modifiées préservent les droits précédents. ⇒ Les logiciels qui dérivent des sources Copyleft ne peuvent être distribués avec un Copyright©.

Tout logiciel a un coût de développement

En général :

- Propriétaire est payant : On paie un coût de développement, un service de support, un service de mise à jour, ... Les sources sont protégées et seuls les propriétaires y on accès.
- Libre est gratuit : Le coût est supporté par une communauté (utilisateurs, subventions publiques, subventions ou sociétés privées, ...).

La ligne de commande

Interface de communication avec le système (IHM)

- Interface historique en mode texte,
- Interface privilégiée sous Linux : de nombreux programmes ne peuvent être appelés qu'à partir de la ligne de commande,
- Interface puissante et programmable.

Principes de fonctionnement

1. L'utilisateur tape des commandes sous forme de texte
2. Le texte est évalué par un interpréteur,
3. L'interpréteur lance l'exécution des commandes.

Utilité

- Permet de lancer des programmes ou des applications,
- Permet d'interroger le système et d'interagir avec lui.
- Basé sur un interpréteur, un langage de programmation permet de construire des scripts pour effectuer des tâches complexes de gestion ou d'administration.

La ligne de commande

```
[ login@localhost ~ ] ■
```

La fenêtre de terminal ou Shell

La ligne de commande est un programme fenêtré simple qui permet de taper du texte.

- La ligne de commande comporte une partie non interprétée [user@localhost ~] appelée le *prompt*. Ici le prompt est configuré pour afficher le nom de l'utilisateur, le nom de la machine, et le nom du répertoire courant.
- Le caractère ■ symbolise la position du curseur. C'est la position où sera inséré le texte frappé par l'utilisateur.
- Le texte tapé par l'utilisateur sera évalué comme une commande ou une suite de commandes par un interpréteur.

L'interpréteur

- L'interpréteur parcourt le texte tapé par l'utilisateur, identifie les commandes et les paramètres, et si la syntaxe est correcte, lance un processus.
- Plusieurs interpréteurs existent : csh, tcsh, bash. Dans ce cours nous utiliserons le **bash**.
- Bash acronyme de Bourne-Again shell, est l'interpréteur du projet GNU. Il est le plus utilisé sous linux.

La ligne de commande

```
[ login@localhost ~ ] ls
public_html/
[ login@localhost ~ ] ■
```

Exécution d'une commande

- La commande (ici `ls`) est évaluée (lancée, interprétée) dès que l'utilisateur presse la touche ⏎ (Entrée). L'ensemble du texte partant du prompt jusqu'à la fin de la ligne est interprété comme une commande.
- Si la commande est valide, un programme est lancé.
- Durant l'exécution du programme, la ligne de commande est indisponible. L'utilisateur doit attendre la fin de l'exécution du programme avant de pouvoir taper une nouvelle commande.
- Si le programme produit un affichage (ici `ls` affiche le nom des fichiers et répertoires), celui-ci est affiché par défaut dans la fenêtre du Shell.
- Une fois la commande exécutée, le Shell propose une nouvelle ligne de commande où l'utilisateur peut taper une nouvelle instruction.

La ligne de commande

```
[ login@localhost ~ ] nom_commande options paramètres
affichage
...
[ login@localhost ~ ] ■
```

Interpretation de la commande

nom_commande Le premier mot doit correspondre au nom d'une commande connue du système,

options Comme leur nom l'indique les options ne sont pas obligatoires. Si il n'y en a pas la commande s'exécute selon un mode "par défaut". L'ajout d'une option pourra modifier ce comportement par défaut.

paramètres Certaines commandes peuvent fonctionner sans paramètre.

Se documenter sur le fonctionnement de Linux

Ressource sur le Web

- Les forums d'utilisateurs :
  - `http://www.gentoo.fr/forum/`
  - `http://www.lea-linux.org/`
  - `http://www.linux-france.org/`
- Les pages Wikipedia pour les commandes, les concepts.
  - `http://fr.wikipedia.org/`
- De nombreux sites de description du système Linux
  - `http://www.linux-france.org/article/man-fr/`

Les pages de `man`

- La ligne de commande intègre une aide pour les commandes les plus courantes. La consultation des pages de `man` est essentielle pour avancer dans la maîtrise des commandes bash. Cela doit devenir un reflexe.
- Les pages de `man` détaillent les syntaxes, options et arguments des commandes. Ces options peuvent être très nombreuses.
- Les pages de `man` sont rédigées en anglais (une version française en ligne est disponible pour certaines commande *cf. supra*). Mais l'anglais est omniprésent en informatique, alors il faut vous faire une raison . . .

man

Syntaxe

```
man  nom_de_la_commande
```

Description

- permet d'accéder à la documentation d'utilisation d'une commande (*i.e.* les pages de `man`).
- Les pages de `man` décrivent les syntaxes, les options, les arguments des commandes.
- Elles décrivent les résultats des évaluations et le format de ces résultats.

Exemple d'utilisation:

```
[ login@localhost ~ ] man ls
```

affiche :

```
LS(1) BSD General Commands Manual LS(1)

NAME
ls -- list directory contents

SYNOPSIS
ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]
```

Plan

Noms et contenu des fichiers

La décomposition d'un nom de fichier

Traditionnellement un nom de fichier se décompose en deux parties séparées par un point :

- La 1ère partie informe sur la nature du contenu du fichier,
- La 2ème partie informe sur le format utilisé pour enregistrer les données.

nom.extension
prefix.suffix
description.format

Exemples de formats de fichiers

Extension	Contenu
.c	Sources C
.html	Document Web
.pdf	Document Mis en page
.txt	Texte brut
.mp3	Fichier Multimedia

Exemples de noms de fichiers

Enigmatique	Informatif
e3.c	teste_boucle_for.c
New.pdf	2011_IntroSys_cours_1.pdf
toto.sh	test_boucle_for.sh

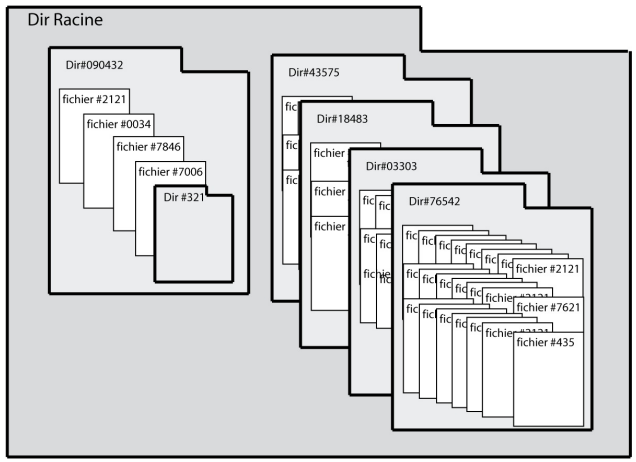
Le choix des noms des fichiers et répertoires

- Ils doivent être choisis minutieusement pour être informatifs,
- Choisir un nom peut être long, mais ce sera un grand gain de temps lorsqu'il s'agira de retrouver le fichier ou le répertoire concerné.

Organisation des données enregistrées

De très nombreux fichiers et répertoires

- Le nombre de fichiers enregistrés sur un disque dur peut aisément dépasser 100.000 fichiers,
- Chaque fichier est identifié par un nom,
- Les fichiers sont regroupés dans des répertoires et sous-répertoires.
- Chaque répertoire est identifié par un nom.



Une organisation en arborescence

- Cette organisation arborescente permet de faciliter la recherche d'un fichier,
- Les fichiers sont regroupés par application, par thème, par format, par fonction, ...

Remarque

- Si tous les fichiers étaient au même *"endroit"*, il serait très difficile de les afficher dans un navigateur. Leur affichage n'apporterait rien car il y en aurait trop à lire.
- L'organisation en arborescence est une organisation hiérarchique, qui permet d'organiser les données et de faciliter leur accès.

Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

15 février 2012

Partie #2

Plan

L'organisation arborescente

Exemple d'arborescence Linux

```
/ ..................................................  Répertoire Racine ou Root Directory
├── bin/
│   └── (...)
├── home/
│   ├── chez_moi/ ..................................  Répertoire Personnel ou User Directory
│   │   ├── Mes_Documents/
│   │   │   ├── ListeDesCourses.txt
│   │   │   └── Exercice_1.sh
│   │   └── (...)
│   └── anonymous/
│       ├── LisezMoi.txt
│       └── Telechargements/
│           └── (...)
└── (...)
```

Les répertoires importants

- Le Repertoire Racine (*Root directory*) contient tous les répertoires et fichiers accessibles depuis le système.
- Le Répertoire Personnel (*User Directory* ou *Home Directory*) est le répertoire dans lequel l'utilisateur peut faire ce qu'il veut (écrire, modifier, supprimer, installer ...).

La notion de Chemin

Le chemin défini un nom unique

- Deux fichiers ou répertoires ne peuvent pas porter le même nom si ils sont dans un même répertoire.
- Les noms des fichiers et répertoires différencient les caractères Majuscules et minuscule. Les fichiers Essai.txt et essai.txt peuvent donc être dans le même répertoire.

Exemples de chemins absolus

```
/ ..................................................  Un chemin absolu part de la racine /
└── home/ ..........................................................  /home/
    └── chez_moi/ ..................................................  /home/chez_moi/
        ├── Etoiles/ ...............................................  /home/chez_moi/Etoiles/
        │   ├── SOLEIL.jpg .........................  /home/chez_moi/Etoiles/SOLEIL.jpg
        │   └── Soleil.jpg .........................  /home/chez_moi/Etoiles/Soleil.jpg
        └── Systeme_Solaire/ .......................  /home/chez_moi/Systeme_Solaire/
            └── SOLEIL.jpg .....................  /home/chez_moi/Systeme_Solaire/SOLEIL.jpg
```

Syntaxe d'un chemin absolu

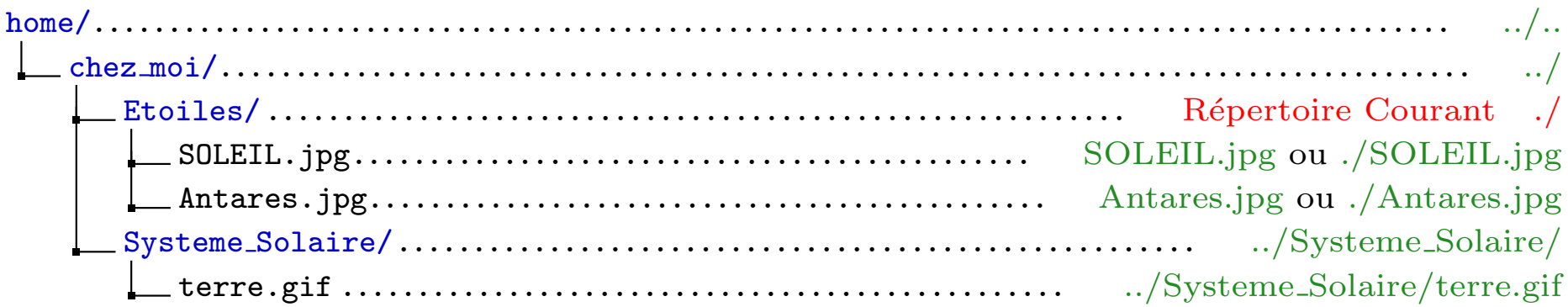
Le chemin *absolu* d'un fichier ou d'un répertoire est unique. Il donne la liste des répertoires et sous-répertoires en partant de la racine / (la référence *absolue* de l'arborescence) jusqu'à la cible.

Répertoire courant et Chemins relatifs

Le répertoire courant

- Le répertoire courant est un répertoire de référence d'où sont lancées les commandes.
- Par défaut, le répertoire courant est le répertoire personnel de l'utilisateur,
- Naviguer dans l'arborescence équivaut à modifier le répertoire courant.

Exemples de chemins relatifs



Syntaxe d'un Chemin Relatif

- Le chemin *relatif* d'un fichier ou d'un répertoire donne la liste des répertoires et sous-répertoires en partant du répertoire courant (la référence *relative* dans l'arborescence) jusqu'à la cible.
- Il est relatif, car lorsque le répertoire courant change, le chemin relatif change.

Notation Spéciales

Les chemins des répertoires de référence

Répertoire	Notation
Répertoire Racine	/
Répertoire Personnel	~

Répertoire	Notation
Répertoire Courant	.
Répertoire Parent	..

Remarques

- La notation ~ correspond à un chemin absolu. Elle est remplacée lors d'une évaluation par le chemin absolu du répertoire personnel de l'utilisateur.

Exemple de chemins valides pointant le fichier cible

```
/ ..........................  Répertoire Racine
└── home/
    └── chez_moi/ .....  Répertoire Personnel
        └── Etoiles/ ....  Répertoire Courant
            └── Soleil.jpg ......  Fichier cible
```

Chemins Absolus
/home/chez_moi/Etoiles/Soleil.jpg
~/Etoiles/Soleil.jpg
/home/chez_moi/../chez_moi/Etoile/Soleil.jpg
/home/chez_moi/../../home/chez_moi/Etoile/Soleil.jpg
Chemins Relatifs
Soleil.jpg
../Etoile/Soleil.jpg
../../chez_moi/Etoile/Soleil.jpg

Tout est Fichier

Gestion des fichiers

Lors de la création du système de fichier une table des i-nodes est créée. Celle-ci fixe le nombre maximum de fichiers.

Fichiers

- Chaque fichier est décrit comme un i-node.
- L'i-node contient un certain nombre de *métadonnées* concernant le fichier :
  - adresse sur le disque et taille du fichier en nombre d'octets,
  - identification du propriétaire (UID et GID) et permissions (lecture, écriture et exécussion),
  - dates de dernière modification et de dernier accès,
  - ...
- Le nom du fichier n'est pas stocké dans son i-node !

Répertoire

Un répertoire est un fichier spécial listant les références des fichiers qu'il contient sous forme de couples (nom_du_fichier, i-node).

Fichiers spéciaux

Les fichiers de périphériques sont des fichiers spéciaux mis en place par le système pour assurer le lien avec un périphérique.

Conventions

Noms et Chemins

- Par convention, le nom d'un fichier ou d'un répertoire est identifié avec son chemin (sauf mention contraire explicite).
- Par convention, un chemin peut être absolu, relatif. Il peut utiliser les notations spéciales.
- Par convention la notion de fichier sera comprise dans son sens large. Par exemple, le chemin d'un fichier devra être interprété sans distinction comme le chemin vers un fichier ordinaire ou comme le chemin vers un répertoire (sauf mention contraire explicite).

Commandes, options, paramètres

Commande c'est le nom d'un programme qui exécute une action.

Options ce sont des paramètres optionnels. Ils peuvent être omis. L'ajout d'options modifie le comportement de la commande (le résultat). Les options sont encadrées par les caractères `< options >`.

Paramètres ce sont des arguments que la commande évalue.

Sources et Cible

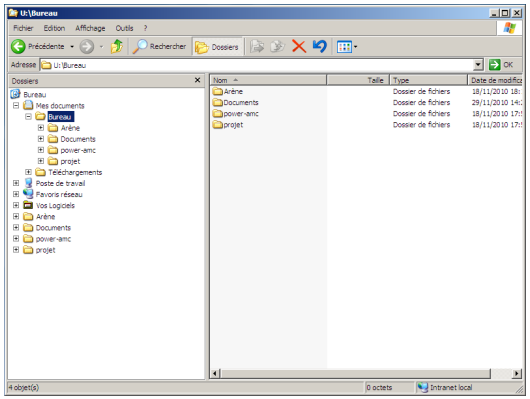
Source c'est un fichier ou un répertoire utilisé en entrée d'une commande,

Cible c'est un fichier ou un répertoire utilisé en sortie d'une commande.

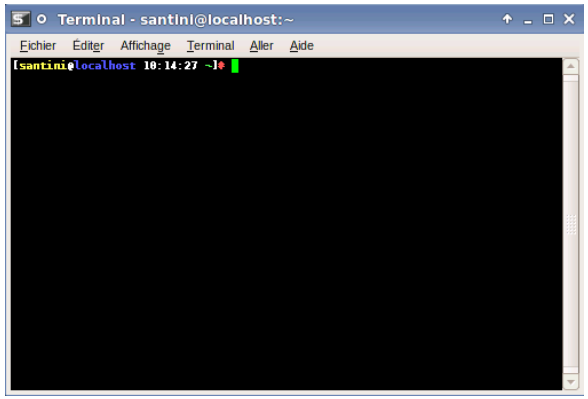
Manipulation de l'arborescence en ligne de commande

Alternatives pour naviguer dans l'arborescence et manipuler les fichiers

Interface Graphique



Ligne de Commande



Principales commandes

Commande	Fonction principale
`pwd`	Afficher le nom du répertoire courant
`ls`	Afficher le contenu d'un répertoire
`cd`	Changer de répertoire courant
`mkdir`	Créer un répertoire
`rm`	Supprimer fichier(s) ou répertoire(s)
`cp`	Copier fichier(s) ou répertoire(s)
`mv`	Déplacer/Renommer fichier(s) ou répertoire(s)

pwd

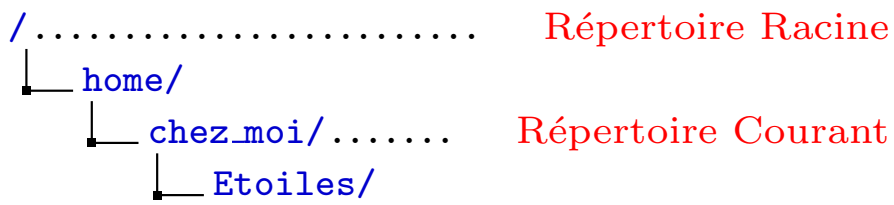
Syntaxe

```
pwd
```

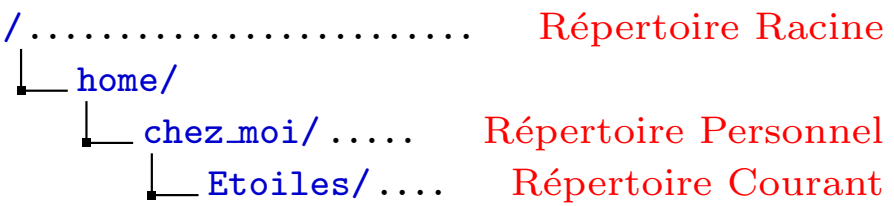
Description

- Affiche le nom du répertoire courant.

Exemple d'utilisation:




```
[ login@localhost ~ ] pwd
/home/chez_moi
```




```
[ login@localhost ~/Etoiles ] pwd
/home/chez_moi/Etoiles/
```

ls

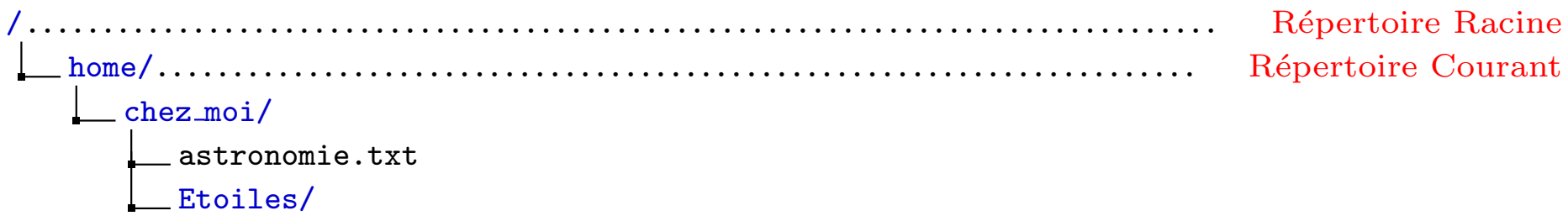
Syntaxe

```
ls  <source>
```

Description

- Affiche le contenu d'un répertoire.
- Par défaut si aucune source n'est indiquée, la commande affiche le contenu du répertoire courant.

Exemple d'utilisation:




```
[ login@localhost /home/ ] ls
chez_moi/
```

```
[ login@localhost /home/ ] ls chez_moi/
Etoiles/ astronomie.txt
```

ls(bis)

Syntaxe

```
ls -a <source>
```

Description

- Affiche le contenu d'un répertoire y compris les fichiers et répertoires cachés.
- Les fichiers et répertoires cachés ont un nom dont le premier caractère est un point.
- Les fichiers et répertoires cachés sont utilisés par le système ou certaines applications.

Exemple d'utilisation:

```
chez_moi/....    Rép. Courant
├── ./ssh/
│   ├── id_rsa
│   ├── id_rsa.pub
│   └── known_hosts
├── .bashrc
├── astronomie.txt
└── Etoiles/
    └── soleil.jpg
```

Sans option -a

```
[ login@localhost ~ ] ls
astronomie.txt
Etoiles/

[ login@localhost ~ ] ■
```

Avec option -a

```
[ login@localhost ~ ] ls -a
.
..
.ssh/
.bashrc
astronomie.txt
Etoiles/

[ login@localhost ~ ] ■
```

cd

Syntaxe

```
cd  <cible>
```

Description

- Change le répertoire courant (permet de naviguer dans l'arborescence).
- Si le chemin du répertoire cible est omit, le répertoire courant redevient par défaut le répertoire personnel.

Exemple d'utilisation:

```
/ ..................................................... Répertoire Racine
└── home/ ............................................. Répertoire Courant Initial
    └── chez_moi/ ..................................... Répertoire Courant Final #1
        ├── astronomie.txt
        └── Etoiles/ .................................. Répertoire Courant Final #2
```

Commande #1 :

```
[ login@localhost /home ] cd

[ login@localhost ~ ] ■
```

Commande #2 :

```
[ login@localhost /home ] cd chez_moi/Etoile

[ login@localhost ~/Etoile ] ■
```

mkdir

Syntaxe

```
mkdir  chemin <chemin_2 ...>
```

Description

- Création d'un ou de plusieurs répertoires aux endroits spécifiés par les chemins.
- Si le chemin est occupé par un fichier ou un répertoire, il y a un message d'erreur.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt
├── Systeme_Solaire/ ..... Création Commande #1
├── Etoiles/
│   ├── Rouges/ ..... Création Commande #2
│   └── Bleues/ ..... Création Commande #3
└── Galaxies/ ..... Création Commande #3
```

Commande #1 :

```
[ login@localhost ~ ] mkdir Systeme_Solaire
```

Commande #2 :

```
[ login@localhost ~ ] mkdir Etoiles/Rouges
```

Commande #3 :

```
[ login@localhost ~ ] mkdir Galaxies Etoiles/Bleues
```

rm

Syntaxe

```
rm  chemin <chemin_2 ...>
```

Description

- La commande supprime le fichier pointé par le(s) chemin(s).
- Si le chemin pointe sur un répertoire, la commande affiche un message d'erreur.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt ..... Supprimé par la Commande #1
├── Etoiles/
│   └── soleil.jpg ..... Supprimé par la Commande #2
└── aldebaran.gif ..... Supprimé par la Commande #2
```

Commande #1 :

```
[ login@localhost ~ ] rm astronomie.txt
```

Commande #2 :

```
[ login@localhost ~ ] rm aldebaran.gif Etoiles/soleil.jpg
```

rm(bis)

Syntaxe

```
rm -r chemin <chemin_2 ...>
```

Description

- L'option -r (**R**écursif) permet de supprimer un répertoire et tout son contenu.

Exemple d'utilisation:

```
chez_moi/ ..................................................... Répertoire Courant
├── astronomie.txt
├── Etoiles/ ..................................... Supprimé par la Commande #1
│   ├── soleil.jpg ............................... Supprimé par la Commande #1
│   └── Galaxie/ ................................. Supprimé par la Commande #1
│       └── Andromede.pdf ........................ Supprimé par la Commande #1
└── aldebaran.gif
```

Commande #1 :

```
[ login@localhost ~ ] rm -r Etoiles
```

cp

Syntaxe

```
cp  source cible
```

Description

- Copie le fichier source vers la cible.
- La source doit être un fichier ordinaire (pas un répertoire),
- Si la source est un répertoire la commande produit un message d'erreur.
- Si la cible :
  - est le chemin d'un répertoire existant, le fichier sera copié dans ce répertoire et conservera son nom,
  - ne correspond pas à un répertoire existant, le fichier sera copié avec le nom `cible`.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt ..... Fichier Source Commande #1
├── Etoiles/ ..... Répertoire Cible Commande #1
│   └── astronomie.txt ..... Copié/Créé par la Commande #1
└── cv.pdf
```

Commande #1 :

```
[ login@localhost ~ ] cp astronomie.txt Etoiles
```

cp(bis)

Syntaxe

```
cp  source <source_2 ...> cible
```

Description

- Copie plusieurs fichiers sources vers la cible.
- Les sources doivent être des fichiers ordinaires, et la cible un répertoire.

Exemple d'utilisation:

```
chez_moi/ ..................................................... Répertoire Courant
├── cv.pdf ..................................................... Fichier Source Commande #2
├── motivations.pdf ............................................ Fichier Source Commande #2
└── Candidature/ ............................................... Répertoire Cible Commande #2
    ├── cv.pdf ................................................. Copié/Créé par la Commande #2
    └── motivations.pdf ........................................ Copié/Créé par la Commande #2
```

Commande #2 :

```
[ login@localhost ~ ] cp cv.pdf motivations.pdf Candidature
```

cp(ter)

Syntaxe

```
cp -r source <source_2 ...> cible
```

Description

- L'option -r (**R**écursif) permet de copier un répertoire et son contenu si il apparait dans le(s) source(s).

Exemple d'utilisation:

```
chez_moi/ ..................................................... Répertoire Courant
├── astronomie.txt
├── Galaxie/ ..................................... Fichier Source Commande #3
│   └── Andromede.pdf
├── Etoiles/ ..................................... Répertoire Cible #3
│   ├── soleil.jpg
│   └── Galaxie/ ..................... Copié/Créé par la Commande #3
│       └── Andromede.pdf ..... Copié/Créé par la Commande #3
└── aldebaran.gif
```

Commande #3 :

```
[ login@localhost ~ ] cp -r Galaxies Etoiles
```

mv

Syntaxe

```
mv  source cible
```

Description

Déplace/Renomme un fichier ou répertoire.

- modifie le chemin d'accès à la `source` qui devient le chemin `cible`.
- Le chemin `source` disparait et le chemin `cible` est créé.
- Le fichier ou répertoire pointé reste le même.
- La cible doit être un chemin non occupé ou un répertoire.

Exemple d'utilisation: Renommer un fichier

État Initial de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── AstroNomIe.TXT.....  Fichier Source
```

État Final de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── astronomie.txt .....  Fichier Renommé
```

```
[ login@localhost ~ ] mv AstroNomIe.TXT astronomie.txt
```

mv(bis)

Exemple d'utilisation: Déplacer un Répertoire

État Initial de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt ..... Fichier Source
└── Etoiles/ ..... Répertoire Cible
```

État Final de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
└── Etoiles/ ..... Répertoire Cible
    └── astronomie.txt ..... Fichier Déplacé
```

```
[ login@localhost ~ ] mv astronomie.txt Etoiles
```

Exemple d'utilisation: Renommer un Répertoire

État Initial de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
└── Etoiles/ ..... Répertoire Source
    └── astronomie.txt
```

État Final de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
└── Relativite/ ..... Répertoire Renommé
    └── astronomie.txt
```

```
[ login@localhost ~ ] mv Etoiles Relativite
```

mv(ter)

Exemple d'utilisation:

État Initial de l'arborescence :

```
chez_moi/.............  Répertoire Courant
├── astronomie.txt.....  Fichier Source
├── relativite.pdf.....  Fichier Source
└── Etoiles/...........  Répertoire Cible
```

État Final de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── Etoiles/...........  Répertoire Cible
    ├── astronomie.txt.  Fichier Déplacé
    └── relativite.pdf.  Fichier Déplacé
```

```
[ login@localhost ~ ] mv astronomie.txt relativité.pdf Etoiles
```

Exemple d'utilisation:

État Initial de l'arborescence :

```
chez_moi/.............  Répertoire Courant
├── relativite.pdf.....  Fichier Source
├── Etoiles/...........  Répertoire Source
│   └── astronomie.txt
└── Espace/............  Répertoire Cible
```

État Final de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── Espace/............  Répertoire Cible
    ├── relativite.pdf.  Fichier Déplacé
    └── Etoiles/.......  Répertoire Déplacé
        └── astronomie.txt  Fichier Déplacé
```

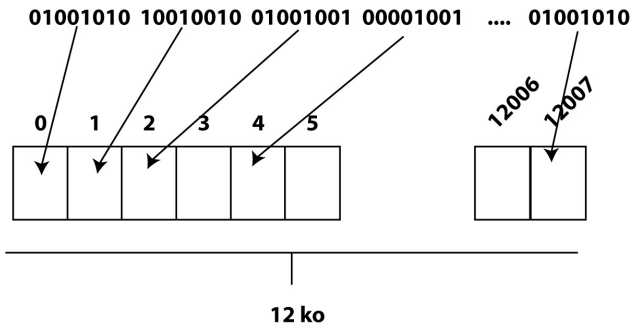
```
[ login@localhost ~ ] mv relativité.pdf Etoiles Espace
```

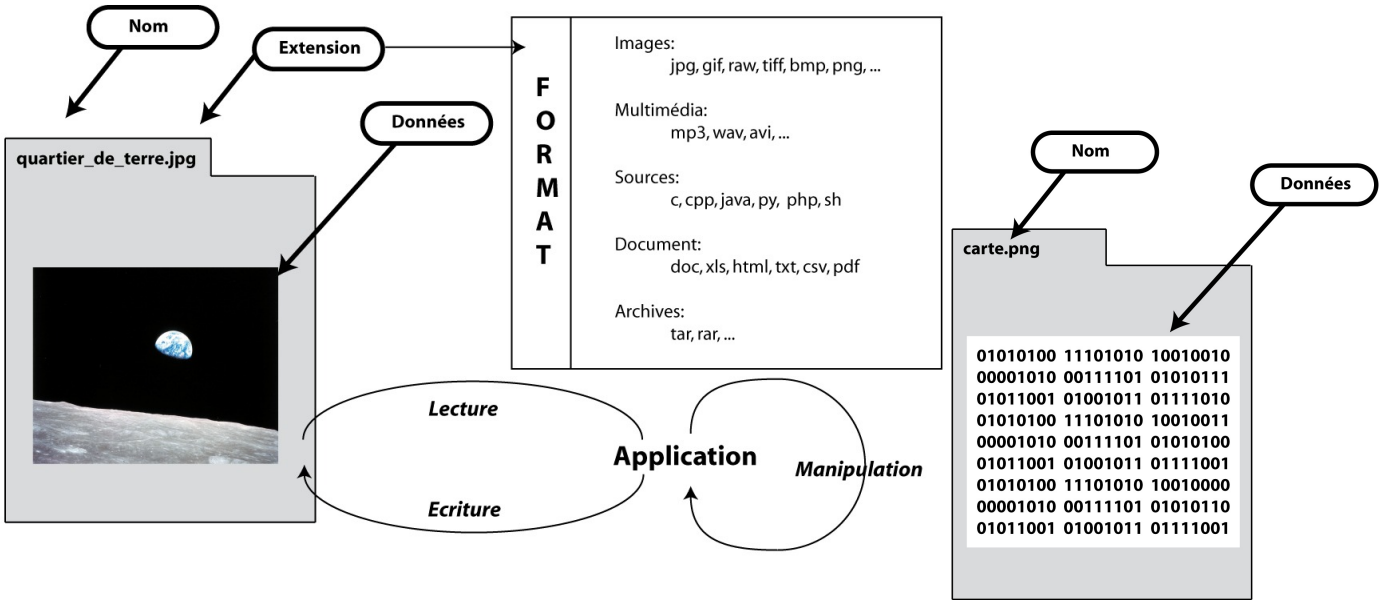
Plan

Fichier binaire et fichier texte

Les données numériques

Tout fichier enregistré sur un support numérique est encodé sous forme binaire.





Accès aux données

Lors de son utilisation un fichier est *lu* par un programme. Pour cela il doit décoder les informations binaires et les traiter.

Fichier binaire et fichier texte

Deux grands types de fichiers : Binaire Vs Numérique

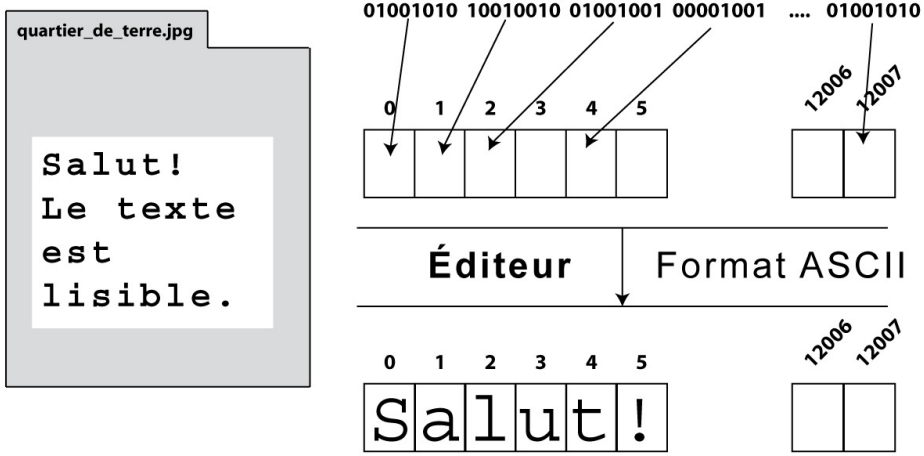
De façon générale un fichier binaire ne peut être *"lu"* que par un programme informatique, alors qu'un fichier texte peut être *"lu"* par être humain.

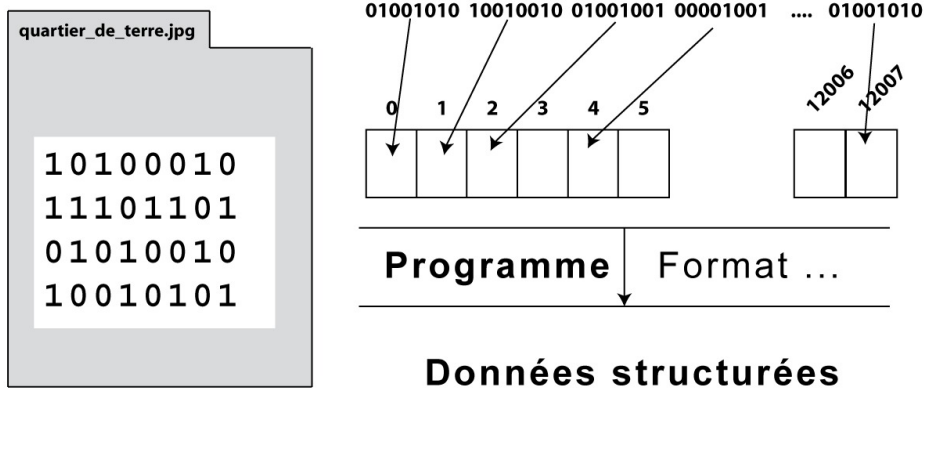
Les fichiers textes

C'est un fichier qui peut être *"lu"* par un éditeur de texte brut. Les données sont encodées au format ASCII (une norme). Chaque octet correspond à un caractère (256 possibles).

Les fichiers binaires

Ce n'est pas un fichier texte . . .Il peut contenir des instructions machines, des données compressées, des données binaires brutes nécessitant un programme pour être lues.





Fichiers sources → Exécutable → Processus

Les sources : Une *"recette de cuisine"*

- Exprime un ensemble de tâches à réaliser pour accomplir le programme (le plat cuisiné).
- Utilise un langage de programmation.
- C'est un fichier texte.

L'exécutable

- Exprime les mêmes tâches dans un langage machine.
- Ce fichier ne fonctionne que sur des ordinateurs qui ont la même architecture.
- C'est un fichier binaire.

Les processus

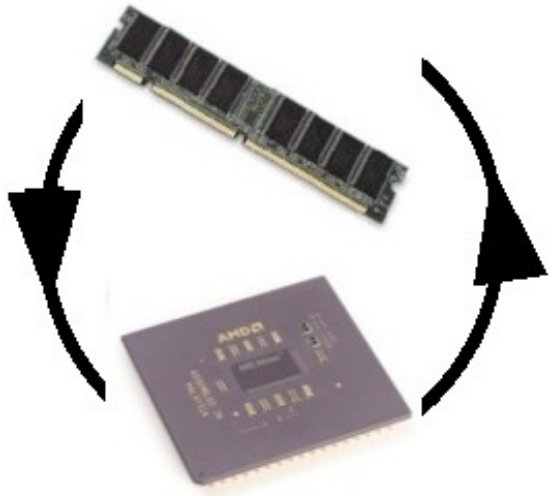
- L'évaluation des instructions machines engendre des processus.
- Ces processus sont exécutés par le matériel.
- Les instructions machine doivent donc être adaptées au matériel.

dessine.c

```
(...)
float r, x, y;
r=3.0;
x=0.0;
y=7.1;
cercle([0.,0.],r)
segment([0.,0.],[x,y])
```

dessine

```
10100101 11101001
10001001 00100101
00101010 00100010
01111011 10110101
01000010 00110011
00101101 11010100
(...)
```



Identification des processus par le système d'exploitation

Système multi-utilisateur

- Plusieurs utilisateurs partagent les mêmes ressources matériel (RAM, CPU, disques, ...),
- Chaque utilisateur lance des processus liés à ses activités sur la machine et il utilise les résultats de ces processus.

Système multi-tâches

- Plusieurs programmes en cours d'exécution partagent les mêmes ressources matériel (mémoire vive, CPU, disques, ...). Ils peuvent provenir d'un seul ou de plusieurs utilisateurs,
- Chaque programmes lance des processus et il utilise les résultats de ces processus.

Il faut partager les ressources ! ! !

- Chaque programme doit être exécuté éventuellement *"en même temps"*. Il faut donc gérer le partage des ressources de calcul (accès à la mémoire vive, au CPU),
- Chaque programme ou utilisateur doit pouvoir retrouver les résultats de ses calculs. Il faut donc pouvoir identifier qui a lancé les processus et qui doit récupérer les résultats.

La gestion des processus est réalisée par le système d'exploitation. C'est une de ses tâches principales. Pour cela il a besoin de pouvoir identifier chaque processus.

PID et PPID

PID - Process IDentifier

- C'est un numéro unique attribué à chaque processus lors de son lancement.
- Il permet d'identifier de façon unique chaque processus.
- La liste des processus en cours d'exécution est accessible en ligne de commande par les commandes `ps` et `top`.

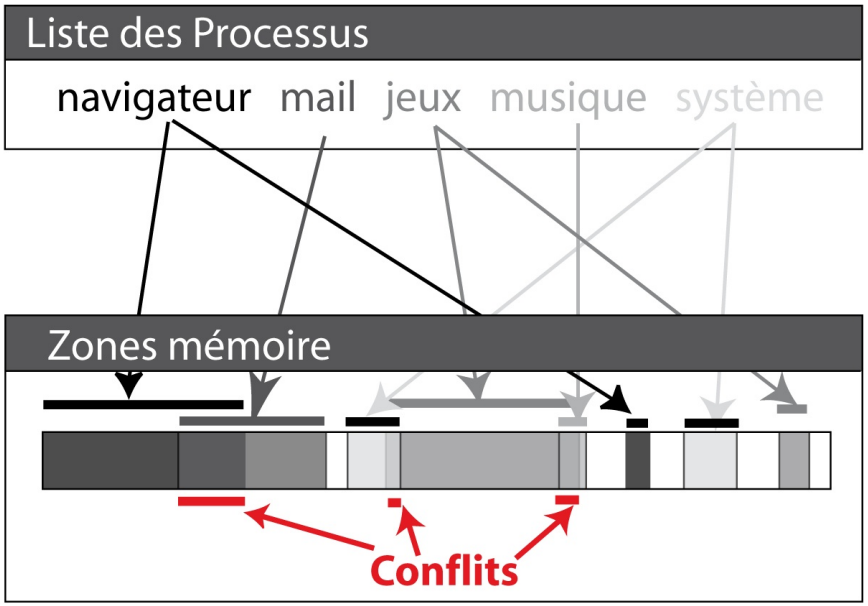
PPID - Parent Process IDentifier

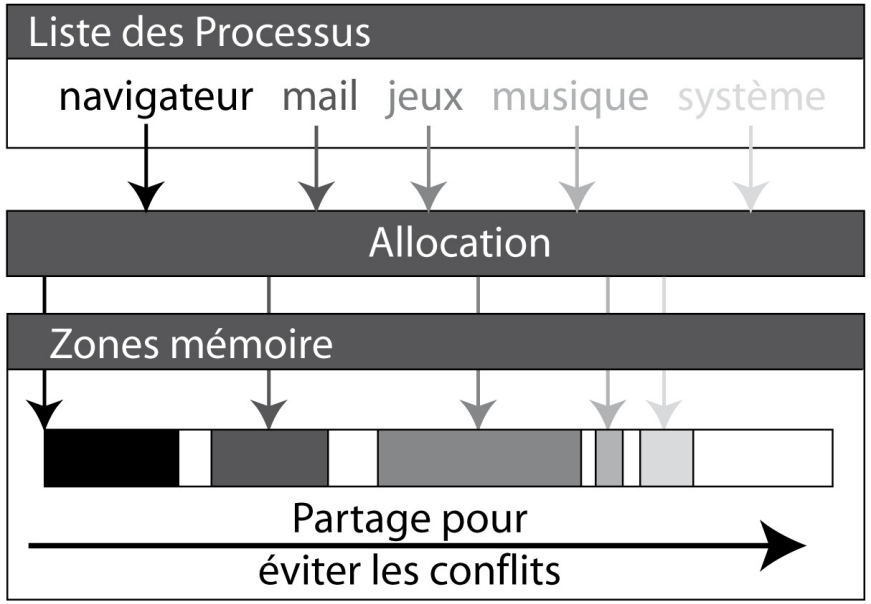
- Le premier processus lancé porte le numéro de PID 1. Les processus suivants sont des processus issus de ce processus parent.
- Chaque processus est lancé par un processus parent *via* l'appel système `fork`.
- Le PPID est le PID du processus Parent.

Utilités

- L'utilisateur peut suivre un processus, le suspendre temporairement, le relancer ou le tuer (interruption définitive).
- Le système s'en sert pour lui affecter des ressources matériel.

Gestion de la mémoire vive





Chaque processus a besoin de mémoire

Pour stocker et travailler sur :

- les données,
- les instructions,
- les résultats.

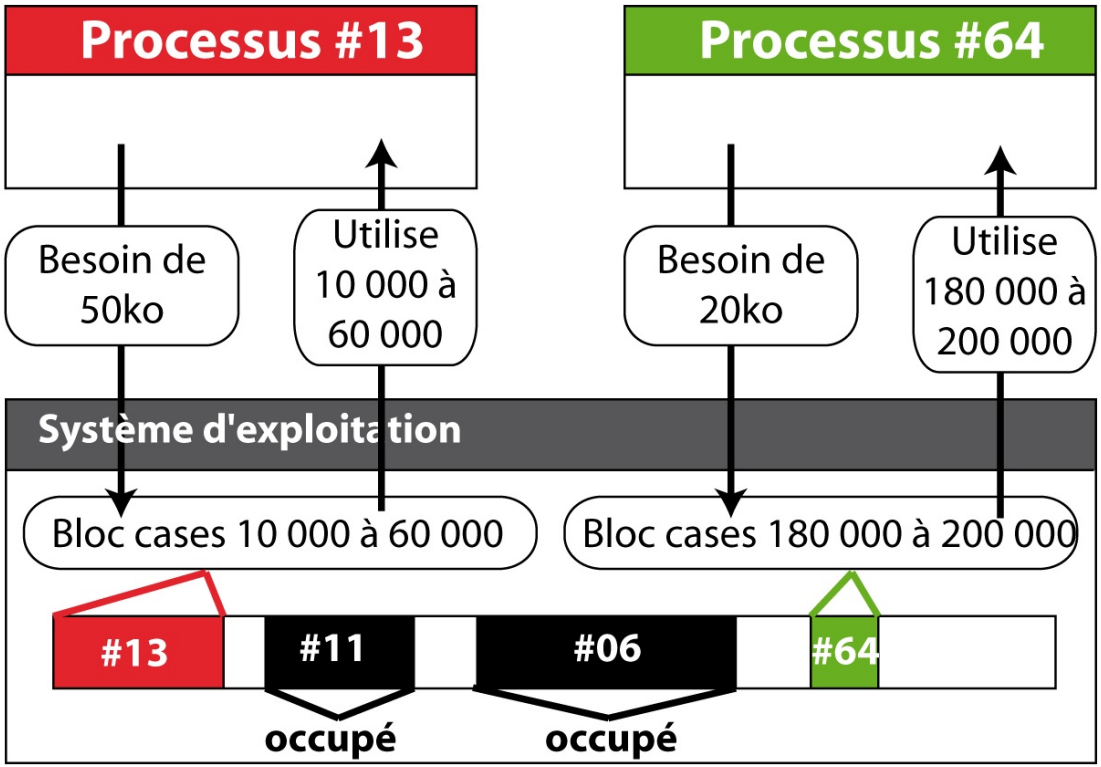
Il faut assurer l'intégrité des données !

Allocation de zone mémoire

L'allocation permet :

- d'attribuer à chaque processus un espace de travail en mémoire,
- le système contraint le programme à écrire dans sa zone mémoire et ainsi,
- évite qu'un programme modifie les données d'un autre programme.

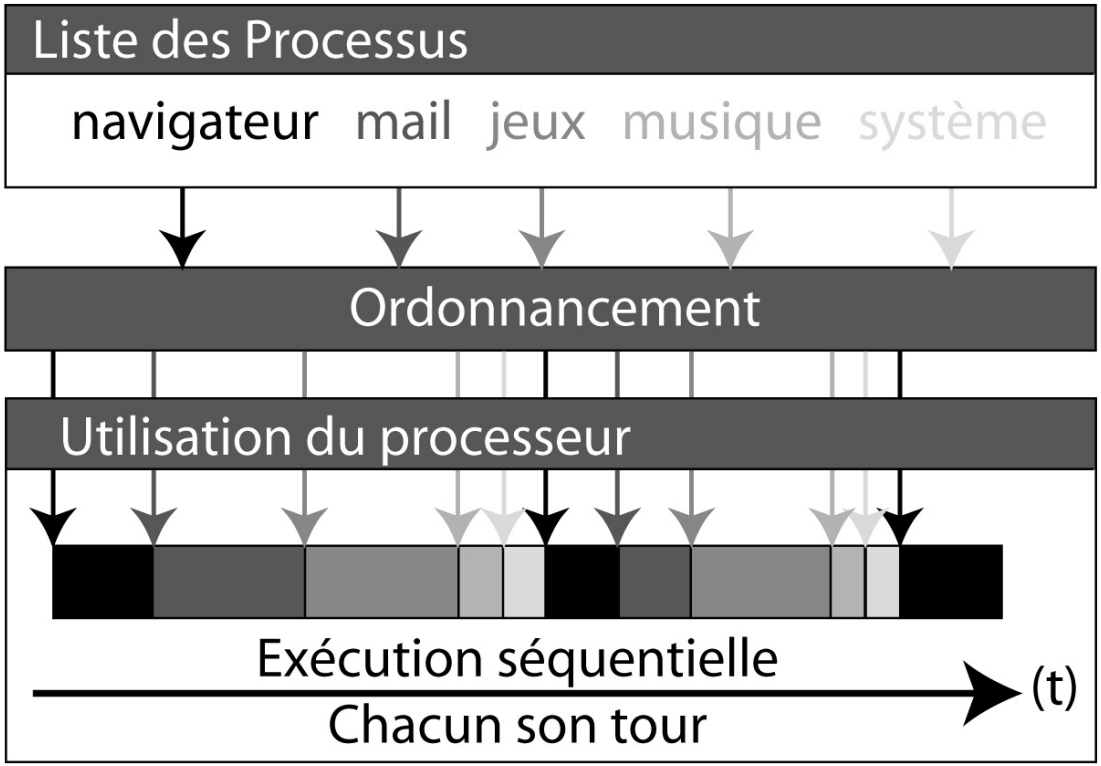
Gestion de la mémoire vive



Principes généraux de l'allocation

- L'OS maintient une table des zones mémoires allouées à chaque processus. Ces zones sont réservées et ne peuvent être utilisées que par le processus parent.
- Lorsqu'il a besoin de mémoire, un processus demande à l'OS quelle zone il peut utiliser,
- L'OS lui attribue, en fonction de l'espace libre, un certain nombre de blocs mémoire.
- Les blocs mémoire attribués sont alors réservés.

Gestion de l'accès au CPU



Le planificateur gère le temps CPU attribué à chaque processus

- Le CPU ne traite qu'un seul processus à la fois,
- Le planificateur permet l'alternance d'accès au CPU en attribuant une priorité à chaque processus.
- L'illusion d'exécution simultanée de plusieurs processus est donnée par une alternance rapide d'attribution de temps de calcul à chaque processus.

ps

Syntaxe

ps <-*eu*>

Description

- Affiche les processus en cours d'exécution.
- L'option <-e> indique que tous les processus doivent être affichés,
- L'option <-u> restreint l'affichage aux processus de l'utilisateur.

Exemple d'utilisation:

```
[ login@localhost ~ ] ps -eu
  Warning : bad ps syntax, perhaps a bogus '-'? See http ://procps.sf.net
   USER  PID %CPU %MEM  VSZ  RSS   TTY STAT  START  TIME COMMAND
santini 5905  0.0  0.2 4824 1656 pts/1   Ss 09 :27 0 :00 -bash LC_ALL=fr_FR.UTF
santini 5962  0.0  0.1 3884  896 pts/1   R+ 09 :48 0 :00 ps -eu MANPATH=/etc/jav

[ login@localhost ~ ] ■
```

top

Syntaxe

```
top
```

Description

- Permet de suivre dynamiquement (temps réel) les ressources matériel utilisées par chaque processus.
- Ouvre un interface dans la ligne de commande qui peut être quittée en pressant la touche Q
- Donne pour chaque processus en autres choses, le PID, le nom du propriétaire, la date de lancement du processus, les %CPU et %MEM utilisés.

Exemple d'utilisation:

```
  Tasks : 85 total, 1 running, 84 sleeping, 0 stopped, 0 zombie
  Cpu(s) : 5.7%us, 0.0%sy, 0.0%ni, 93.6%id, 0.0%wa, 0.7%hi, 0.0%si, 0.0%st
  Mem : 772068k total, 231864k used, 540204k free, 2412k buffers
  Swap : 995992k total, 0k used, 995992k free, 161316k cached

 PID    USER PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+             COMMAND
5116    root 20  0 33832  22m 6576 S  5.7  3.0 0 :19.49                   X
5879 santini 20  0 16060 7344 6116 S  0.3  1.0 0 :01.06    xfce4-netload-p
   1    root 20  0  1664  568  496 S  0.0  0.1 0 :02.95                init
   2    root 20  0     0    0    0 S  0.0  0.0 0 :00.00           kthreadd
   3    root RT  0     0    0    0 S  0.0  0.0 0 :00.00        migration/0
```

Processus en ligne de commande

Occupation de la ligne de commande

- Lorsque l'on tape une commande, la ligne de commande est bloquée (plus de prompt) jusqu'à la fin de l'exécution.
- La ligne de commande est à nouveau disponible ensuite.

```
[ login@localhost ~ ] sleep 20
(il faut attendre 20 secondes avant l'apparition du
nouveau prompt)
...
...
[ login@localhost ~ ] ■
```

```
[ login@localhost ~ ] gedit
(Il faut quitter l'application ou tuer le processus gedit
pour avoir un nouveau prompt)
...
...
```

Libération de la ligne de commande

Deux façons possibles de lancer une instruction en tâche de fond :

Lancement en tâche de fond

- Les commandes qui prennent beaucoup de temps peuvent être lancées en tâche de fond pour libérer la ligne de commande du shell.
- Pour lancer directement la commande en tâche de fond il suffit de faire suivre la commande du caractère &. On retrouve immédiatement un nouveau prompt.

```
[ login@localhost ~ ] gedit &

[ login@localhost ~ ] ■
```

Relégation en tâche de fond

- Si une tâche déjà lancée occupe la ligne de commande, il est possible de suspendre son exécution en pressant la combinaison de touches Ctrl + z . La tâche est alors interrompue et on retrouve un nouveau prompt.
- Il est possible de relancer le processus en tâche de fond au moyen de la commande bg.

```
[ login@localhost ~ ] gedit
^Z
[1]+ Stopped gedit

[ login@localhost ~ ] bg
[1]+ gedit &

[ login@localhost ~ ] ■
```

Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

15 février 2012

Partie #3

Plan

Propriété des fichiers

Identifications des utilisateurs dans un environnement multi-utilisateurs

UID (**U**ser **ID**entifier) numéro unique associé à chaque utilisateur lors de la création de son compte.

GID (**G**roup **ID**entifier) numéro unique d'un groupe d'utilisateurs. Chaque utilisateur peut être associé à un ou plusieurs groupes.

Utilité

- Chaque fichier (ou répertoire) et chaque processus du système est associé à un utilisateur : cet utilisateur est le propriétaire du fichier (ou répertoire) ou celui qui a lancé le processus.
- Être propriétaire d'un fichier ou d'un processus confère des droits sur ceux-ci.

Connaitre l'identité du propriétaire d'un processus ou d'un fichier

- Les commandes `top` et `ps` affichent le nom du propriétaire des processus.
- La commande `ls` avec l'option `-l` affiche le nom et le groupe du propriétaire d'un fichier ou d'un répertoire.

ls(ter)

Syntaxe

```
ls -l <source>
```

Description

- Affiche le contenu d'un répertoire en format long.
- Le format long donne le nom du propriétaire et son groupe, ainsi que les droits des différentes classes d'utilisateurs sur les fichiers et répertoires.

Exemple d'utilisation:

```
chez_moi/.....................................................  Répertoire Courant
├── public_html/
│   └── index.html
└── astronomie.txt
```

```
[ login@localhost ~ ] ls -l
 total 32
drwxr-xr-x 2 santini ensinfo 4096 20 jui 15 :50     public_html
-rw-r--r-- 1 santini ensinfo   25 20 jui 15 :49 telluriques.txt
```

Ici, le nom de l'utilisateur est santini, nom du groupe est ensinfo et les droits sont colorés en vert.

Les droits sur les fichiers et répertoires

3 catégories d'utilisateurs

-	r w x	r w x	r w x
Type de Fichier	Doits du propriétaire (**U**ser)	Doits du groupe (**G**roup)	Doits des autres (**O**ther)

Types de fichiers

	Types
-	Fichier ordinaire
d	Répertoire
l	lien symbolique

Droits/Permissions

		Fichier	Répertoire
`r`	(**R**ead)	lire	lister le contenu
`w`	(**W**rite)	écrire et modifier	modifier le contenu
`x`	(e**X**ecute)	exécution	traverser

Types d'utilisateurs

		Cible
u	(**U**)ser	Propriétaire du fichier/répertoire
g	(**G**)roup	Membre du même groupe que le propriétaire
o	(**O**)ther	Tous les autres
a	(**A**)ll	Tous les utilisateurs (réunion de 'u', 'g' et 'o').

chmod

Syntaxe

```
chmod  droit fichier
```

Description

- Modifie les droits et permissions accordés par le propriétaire aux différents utilisateurs du système.

Exemple d'utilisation:

Retire au propriétaire le droit d'écriture sur le fichier `cv_2011.pdf`.

```
[ login@localhost ~ ] chmod u-w cv_2011.pdf
```

Retire aux utilisateurs qui ne sont ni le propriétaire ni membre de son groupe les droits de lecture, d'écriture et d'exécution.

```
[ login@localhost ~ ] chmod o-rwx listing.bash
```

Ajoute au propriétaire et aux membres de son groupe le droit d'exécution sur le fichier `listing.bash`.

```
[ login@localhost ~ ] chmod ug+x listing.bash
```

Ajoute à tous les utilisateurs, tous les droits.

```
[ login@localhost ~ ] chmod a+rwx listing.bash
```

chmod(bis)

Description

Il existe plusieurs notations des droits.

- La notation alphanumérique : `(ugoa) (+/-) (rwx)`
- La notation octale :

Droit	`---`	`--x`	`-w-`	`-wx`	`r--`	`r-x`	`rw-`	`rwx`
Binaire	000	001	010	011	100	101	110	111
Octale	0	1	2	3	4	5	6	7

Alphabétique	`r`	`w`	`x`	`r`	`-`	`x`	`-`	`-`	`x`
Binaire	1	1	1	1	0	1	0	0	1
Octale	7			5			1		

Exemple d'utilisation:

Alph.	Oct.
`--- --- ---`	000
`rw- --- ---`	600
`rw- r-- r--`	644
`rw- rw- rw-`	666

Alph.	Oct.
`rwx --- ---`	700
`rwx r-x r-x`	755
`rwx rwx rwx`	777

```
[ login@localhost ~ ] chmod 700 dir_parano

[ login@localhost ~ ] chmod 644 fichier_pub
```

Plan

Compression

Réduire la taille des fichiers

- Pour économiser de la place sur les supports de stockage,
- Pour réduire la quantité de données à transférer sur un réseau.

Algorithmes

On distingue les algorithmes de compression avec perte ou sans perte :

- Sans perte, signifie que l'algorithme de décompression **permettra** de retrouver les données telles qu'elles étaient avant la compression. Il s'agit juste d'une ré-écriture des données sous forme plus concise.
- Avec perte, signifie que l'algorithme de décompression **ne permettra pas** de retrouver les données telles qu'elles étaient avant la compression. Les données initiales sont modifiées pour prendre moins de place.

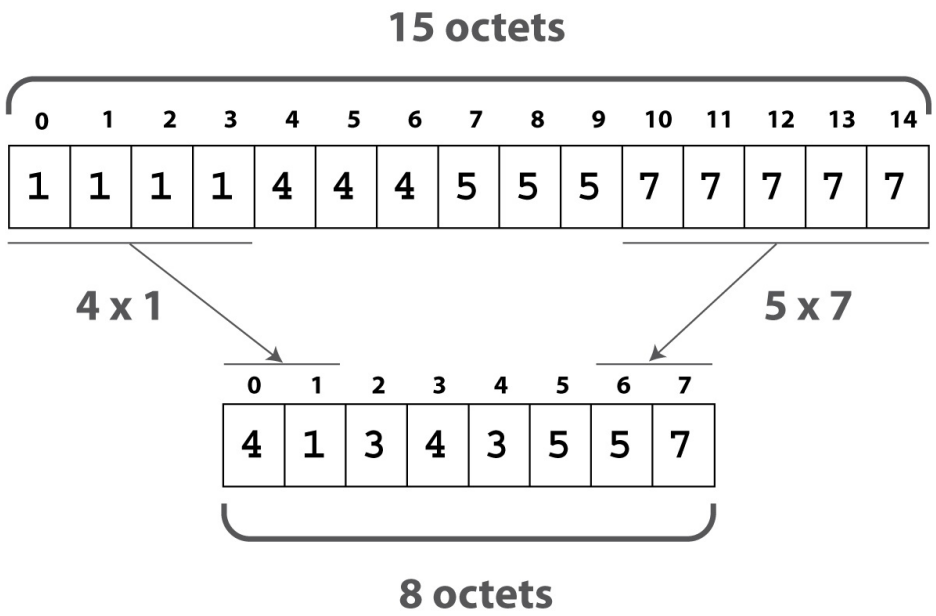
Formats compressés

Les fichiers compressés sont des fichiers binaires

- Formats issus d'un programme de compression : `.gz`, `.bz2`, `.Z`, `.tgz`, `.zip`, ...
- Formats spécialisés : `.jpeg`, `.mpeg`, ...

Compression

Principe de l'algorithme RLE (Run-Length Encoding)



Description

- Une séquence de n répétitions du motif m est ré-écrite comme un couple (n,m).
- L'ordre des couples permet de retrouver les données initiales,
- C'est un algorithme de compression sans perte.

gzip

Syntaxe

```
gzip  fichier <fichier_2 ...>
```

Description

- Compresse un ou plusieurs fichiers dont le nom est passé en paramètre.
- Le fichier source (initial non compressé) est supprimé et seul subsiste le fichier compressé.
- Le fichier compressé qui apparaît porte le même nom que le fichier initial avec l'extension .gz ajoutée à la fin.

Exemple d'utilisation:

```
chez_moi/.............  Répertoire Courant
├── tellurique.tsv
└── astronomie.txt.............  Avant gzip
```

```
chez_moi/.............  Répertoire Courant
├── tellurique.tsv
└── astronomie.txt.gz .........  Après gzip
```

```
[ login@localhost ~ ] ls
astronomie.txt telluriques.tsv

[ login@localhost ~ ] gzip astronomie.txt


[ login@localhost ~ ] ls
astronomie.txt.gz telluriques.tsv
```

gunzip

Syntaxe

```
gunzip  fichier <fichier_2 ...>
```

Description

- Décompresse un ou plusieurs fichiers dont le nom est passé en paramètre.
- Le fichier source (compressé) est supprimé et seul subsiste le fichier décompressé.
- Le fichier décompressé qui apparaît porte le même nom que le fichier initial sans l'extension .gz ajoutée à la fin.

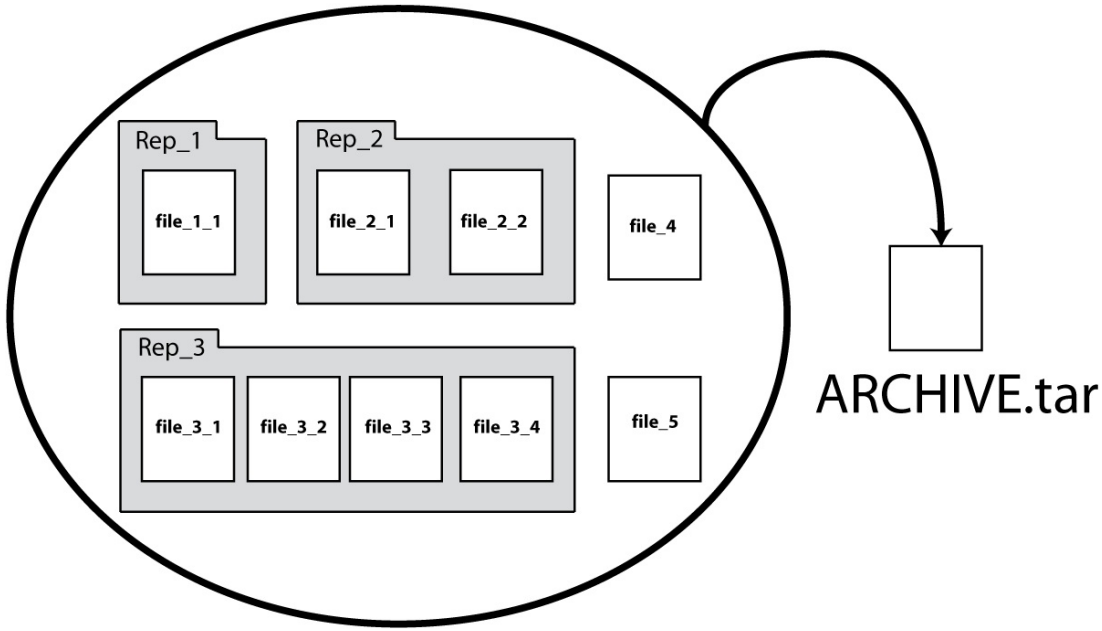
Exemple d'utilisation:

```
chez_moi/.............. Répertoire Courant
├── tellurique.tsv
└── astronomie.txt.gz..... Avant gunzip
```

```
chez_moi/.............. Répertoire Courant
├── tellurique.tsv
└── astronomie.txt ..... Après gunzip
```

```
[ login@localhost ~ ] ls
astronomie.txt.gz telluriques.tsv

[ login@localhost ~ ] gunzip astronomie.txt.gz


[ login@localhost ~ ] ls
astronomie.txt telluriques.tsv
```

Archivage



Regrouper les fichiers

- Pour simplifier le transfert des données, un groupe de fichiers et de répertoires sont concaténés dans un seul fichier qui peut être compressé.
- Pour archiver des données non utilisées (sauvegardes intermédiaires).

tar

Syntaxe

```
tar cv nom_archive fichier_ou_repertoire <autres_sources>
```

Description

- Crée un fichier archive dont le nom (chemin) est donné en premier argument et porte classiquement l'extension `.tar`.
- Les fichiers sources qui servent a créer l'archive sont préservés par la commande `tar`.
- L'option `c` (**C**reate), indique que la commande `tar` doit utiliser un algorithme d'archivage.
- L'option `v` (**V**erbose), permet d'afficher le déroulement de l'archivage.

Exemple d'utilisation:

Regroupe dans la même archive `espace.tar` le fichier `astronomie.txt` et le répertoire `Images/` et son contenu :

```
[ login@localhost ~ ] tar cv espace.tar astronomies.txt Images/
```

tar(bis)

Syntaxe

`tar` *xv* `nom_archive`

Description

- Extrait les fichiers et répertoires d'une archive.
- Les fichiers sont placés dans le répertoire courant.
- L'option `x` (e**X**tarct) indique que la commande `tar` doit utiliser un algorithme de désarchivage.

Exemple d'utilisation:

Extrait le contenu de l'archive `espace.tar` :

```
[ login@localhost ~ ] tar xv espace.tar
```

tar(ter)

Syntaxe

```
tar cvz nom_archive fichier_ou_repertoire <autres_sources>
```

Syntaxe

```
tar xvz nom_archive
```

Description

- L'option z permet de créer ou d'extraire une archive compressée.
- L'extension donnée aux fichiers contenant une archive compressée par ce moyen est classiquement : `.tgz`

Exemple d'utilisation:

Crée une archive compressée `espace.tgz` avec le fichier `astronomie.txt` et le répertoire `Images/` et son contenu :

```
[ login@localhost ~ ] tar cvz espace.tar astronomie.txt Images/
```

Extrait le contenu d'une archive compressée `espace.tgz` :

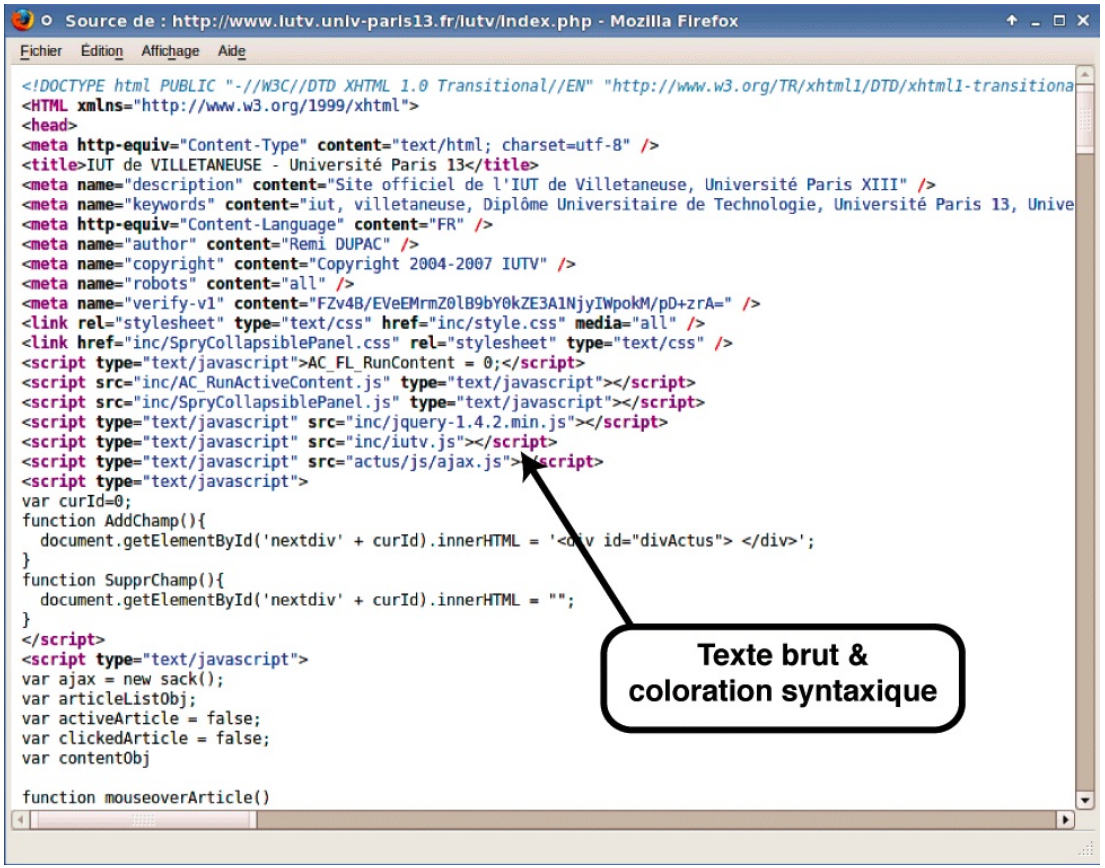
```
[ login@localhost ~ ] tar xvz espace.tar
```

Plan

L'éditeur de texte

Caractéristiques

- Affiche le contenu de fichiers textes bruts,
- Le texte n'est pas mis en forme : pas de notion de titre ni de paragraphe, de taille ou de police de caractères . . .
- Le texte ne contient que des caractères alpha-numériques.



La coloration syntaxique

- L'éditeur reconnait le format du fichier et applique des règles de coloration sur certain mots clefs, délimiteurs, . . .,
- Contrairement aux documents mis en forme (pdf, doc), la coloration des caractères n'est pas sauvée dans le fichier, c'est le résultat d'un calcul de l'éditeur.

L'éditeur de texte

Les racourcis claviers utiles

Combinaisons de touches	Action
Ctrl + A	Tout sélectionner
Ctrl + C	Copier la sélection
Ctrl + X	Couper la sélection
Ctrl + V	Coller la sélection la où se trouve le curseur
Ctrl + Z	Annuler la dernière opération
Ctrl + S	Enregistrer le document
Shift ⇑ + Ctrl + S	Enregistrer le document sous ...
Ctrl + O	Ouvrir un document
Ctrl + N	Créer un nouveau document

Utilisation

Les fichiers textes sont utilisés pour :

- Écrire les sources des scripts et des programmes,
- Écrire les sources de certains documents (HTML, LaTeX, ...),
- Enregistrer/Manipuler des données, ...

Prenez l'habitude de sauver régulièrement votre travail

- Une version précédente du document pourra toujours être obtenue en répétant la combinaison Ctrl + Z ,
- Si il y a un problème avec votre ordinateur (coupure d'alimentation électrique, plantage du système, ...), vous aurez ainsi une version récente de votre travail.

L'édition en ligne de commande

De nombreux outils

Visualisation de Contenu	
`more`	Affiche le contenu page par page
`less`	Ouvre un environnement permettant de naviguer dans le document
Extraction de Parties	
`head`	Affiche les premières lignes
`tail`	Affiche les dernières lignes
`cut`	Affiche une colonne
`grep`	Affiche les lignes comportant un motif particulier
Manipulation de textes	
`cat`	Concatène plusieurs fichiers
`sed`	Opère des substitutions
`sort`	Trie les lignes selon un ordre
`uniq`	Supprime les lignes consécutives identiques
Analyse de contenu	
`wc`	Compte le nombre de lignes, de mots et d'octets

more

Syntaxe

```
more  fichier <fichier_2 ...>
```

Description

- Affiche le contenu du (des) fichier(s) page par page,
- L'affichage s'adapte à la taille du shell,
- Pour passer à la ligne suivante, l'utilisateur presse la touche Enter .
- Pour passer à la page suivante, l'utilisateur presse la touche Space .
- Une fois que tout le contenu du fichier a défilé, l'utilisateur retrouve un nouveau prompt.

Exemple d'utilisation:

- Cette commande est utilisée pour parcourir des documents dont l'affichage dépasse la taille de la fenêtre du terminal.
- Utilisée avec un tube (*cf.* Partie sur les Redirections) elle permet de visualiser tous les résultats d'une commande qui dépasserait la taille de la fenêtre du terminal. Par exemple, si un répertoire contient de très nombreux fichiers, la commande `ls` qui affiche le contenu du répertoire peut produire un affichage très long. Si l'on souhaite passer en revue tous les fichiers il faut alors utiliser la commande suivante :

```
[ login@localhost ~ ] ls Ma_Musique | more
```

less

Syntaxe

```
less  fichier
```

Description

- Affiche le contenu d'un fichier,
- Permet de naviguer en avant et en arrière dans le fichier.
- Permet d'effectuer des recherches de mot(if)s.

La commande ouvre une interface dans la fenêtre du terminal. Contrairement à la commande `more`, on ne revient pas à la ligne de commande lorsqu'on atteint la fin du fichier, pour cela il faut quitter l'application.

Exemple d'utilisation:

Pour avoir une description complète des commandes de navigation dans l'interface de visualisation `less`, reportez-vous aux pages de `man`. Les commandes les plus utilisées sont :

Combinaison de touches	Action
H	Affiche l'aide (abrégé des commandes)
F	Avancer d'une page (forward)
B	Reculer d'une page (backward)
E	Avancer d'une ligne
Y	Reculer d'une ligne

Combinaison de touches	Action
Q	Quiter
G	Aller à la première ligne
Shift ⇑ + G	Aller à la dernière ligne
num + G	Aller à la ligne numéro `num`

head

Syntaxe

```
head < -int > fichier
```

Description

- Affiche par défaut les 10 premières lignes d'un fichier.
- Si un entier `n` précède le nom du fichier, la commande affiche les n premières lignes du fichier.

Exemple d'utilisation:

Soit le fichier `planetes.txt` contenant les lignes suivantes :

planetes.txt

```
# Premier groupe
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique
# Deuxième groupe
1  Jupiter  Gazeuse
2  Saturne  Gazeuse
3  Uranus   Gazeuse
4  Neptune  Gazeuse
```

La commande suivante affiche les 5 premières lignes du fichier :

```
[ login@localhost ~ ] head -5 planetes.txt
# Premier groupe
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique

[ login@localhost ~ ]
```

tail

Syntaxe

```
tail < -int > fichier
```

Description

- Affiche par défaut les 10 dernières lignes d'un fichier.
- Si un entier `n` précède le nom du fichier, la commande affiche les n dernières lignes du fichier.

Exemple d'utilisation:

Soit le fichier `planetes.txt` contenant les lignes suivantes :

planetes.txt

```
# Premier groupe
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique
# Deuxième groupe
1  Jupiter  Gazeuse
2  Saturne  Gazeuse
3  Uranus   Gazeuse
4  Neptune  Gazeuse
```

La commande suivante affiche les 4 dernières lignes du fichier :

```
[ login@localhost ~ ] tail -4 planetes.txt
1  Jupiter  Gazeuse
2  Saturne  Gazeuse
3  Uranus   Gazeuse
4  Neptune  Gazeuse

[ login@localhost ~ ] ■
```

cut

Syntaxe

```
cut -d 'sep' -f n fichier
```

Description

- Affiche une colonne du fichier.
- L'option <-d 'sep'> permet de changer le séparateur par défaut qui est la tabulation. Le séparateur est donné entre guillement simples.
- L'option <-f n> indique que la commande doit afficher la n^{ème} colonne.

Exemple d'utilisation:

Cas#1 : les mots (les champs) sont séparés par des tabulations :

tellur.tsv

```
1	Mercure	Venus
2	Terre	Mars
```

Commande #1

```
[ login@localhost ~ ] cut -f 2 tellur.tsv
Mercure
Terre
[ login@localhost ~ ] ■
```

Cas#2 : les mots (les champs) sont séparés par le caractère = :

jov.txt

```
1=Jupiter=Saturne
1=Uranus=Neptune
```

Commande #2

```
[ login@localhost ~ ] cut -d '=' -f 3 jov.txt
Saturne
Neptune
[ login@localhost ~ ] ■
```

grep

Syntaxe

```
grep  "motif" fichier
```

Description

- Affiche les lignes du fichier qui comportent le "motif".
- Les lignes sont affichées dans leur ordre d'apparition dans le fichier.

Exemple d'utilisation:

Soit le fichier planetes.txt contenant les lignes suivantes :

planetes.txt

```
# Premier groupe
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique
# Deuxième groupe
1  Jupiter  Gazeuse
2  Saturne  Gazeuse
3  Uranus   Gazeuse
4  Neptune  Gazeuse
```

Commandes :

```
[ login@localhost ~ ] grep 'Tellurique' planetes.txt
1  Mercure  Tellurique
2  Venus    Tellurique
3  Terre    Tellurique
4  Mars     Tellurique

[ login@localhost ~ ] grep '1' planetes.txt
1  Mercure  Tellurique
1  Jupiter  Gazeuse

[ login@localhost ~ ] ■
```

cat

Syntaxe

```
cat  fichier <fichier_2 ...>
```

Description

- Affiche le contenu des fichiers les uns à la suite des autres.
- Les fichiers sont concaténés dans l'ordre des paramètres.

Exemple d'utilisation:

Cette commande est en générale utilisée pour concaténer des fichiers textes. On l'utilise avec une commande de redirection (*cf.* Partie Redirections) pour enregistrer le résultat de la concaténation dans un nouveau fichier.

Soient les deux fichiers suivants :

tellur.txt

```
Mercure, Venus
Terre, Mars
```

jov.txt

```
Jupiter, Saturne
Uranus, Neptune
```

La commande :

```
[ login@localhost ~ ] cat tellur.txt jov.txt
Mercure, Venus
Terre, Mars
Jupiter, Saturne
Uranus, Neptune

[ login@localhost ~ ] ■
```

sort

Syntaxe

```
sort <-r> fichier
```

Description

- Affiche les lignes du fichier triées par ordre croissant.
- L'option `-r` inverse l'ordre de tri.

Exemple d'utilisation:

Soit le fichier :

donnees.txt

```
a
A
1
7
8
71
```

```
[ login@localhost ~ ] sort donnees.txt
1
7
71
8
A
a

[ login@localhost ~ ] ■
```

uniq

Syntaxe

```
uniq  fichier
```

Description

- Affiche les lignes du fichier en supprimant les lignes consécutives identiques.

Exemple d'utilisation:

Soit le fichier :

donnees.txt

```
1
lune
Terre
Terre
lune
```

```
[ login@localhost ~ ] uniq donnees.txt
1
lune
Terre
lune

[ login@localhost ~ ] ■
```

sed

Syntaxe

```
sed  's/motif/new/g' fichier
```

Description

La commande sed est une commande qui permet de faire de nombreuses opérations. Nous ne verrons ici que la syntaxe permettant de substituer un motif dans un texte.

- Affiche le contenu du fichier après avoir remplacé les occurrences du `motif` par `new`.

Exemple d'utilisation:

Soit le fichier :

dialogue.txt

```
- C'est par ici !!!
-Où ça, "ici" ?
```

```
[ login@localhost ~ ] sed 's/ici/là/' dialogue.txt
- C'est par là !!!
-Où ça, "là" ?

[ login@localhost ~ ] ■
```

wc

Syntaxe

```
wc  fichier <fichier_2 ...>
```

Description

- Affiche des statistiques sur le nombre de lignes, de mots et de caractères (comptés en nombre d'octets) contenus dans le fichier dont le chemin est donné en paramètre.

Exemple d'utilisation:

Soit le fichier suivant :

tellur.tsv

```
1 Mercure Venus
2 Terre Mars
```

Commande #1 :

```
[ login@localhost ~ ] wc tellur.tsv
2       6       29  tellur.tsv
[ login@localhost ~ ] ■
```

L'affichage produit indique que le fichier `tellur.tsv` comporte :

- 2 lignes,
- 6 mots et
- 29 caractères. La taille du fichier texte est donc de 29 octets ...

wc(bis)

Syntaxe

```
wc -l fichier <fichier_2 ...>
```

Description

- L'option `-l` indique que l'on affiche que le nombre de lignes.

Exemple d'utilisation:

Soit le fichier suivant :

tellur.tsv

```
1 Mercure Venus
2 Terre Mars
```

L'affichage produit indique que le fichier `tellur.tsv` comporte :

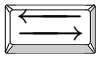
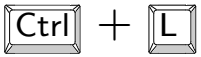
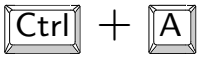
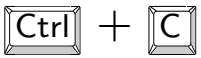
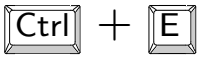
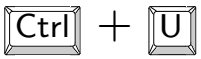
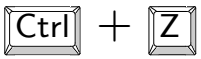
- 2 lignes.

Commande #1 :

```
[ login@localhost ~ ] wc -l tellur.tsv
2       tellur.tsv
[ login@localhost ~ ] ■
```

Se faciliter la vie en ligne de commande

Raccourcis clavier

Combinaisons de Touches	Fonction
	Complétion automatique
	Vide la fenêtre : place la ligne de commande en haut de la fenêtre
	Curseur au début de la ligne
	Tue le processus en cours d'exécution
	Curseur à la fin de la ligne
	Efface ce qui précède le curseur
	Interrompt l'exécution d'un processus
	Affiche la commande précédante dans l'historique
	Affiche la commande suivante dans l'historique

wget

Syntaxe

wget *chemin*

Description

- Client HTTP, HTTPS et FTP .
- Permet de récupérer du contenu d'un serveur serveur Web ou FTP (télécharger).

Exemple d'utilisation:

```
[ login@localhost ~ ] wget http ://www-lipn.univ-paris13.fr/~santini/intro_syste
me/2011_2012_S1D_cours_1.pdf .
Résolution de www-lipn.univ-paris13.fr... 10.10.0.68
Connexion vers www-lipn.univ-paris13.fr|10.10.0.68| :80... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Longueur : 4568618 (4,4M) [application/pdf]
Sauvegarde en : «2011_2012_S1D_cours_1.pdf»

100%[=====================================>] 4 568 618 10,4M/s ds 0,4s

2012-01-02 16 :02 :59 (10,4 MB/s) - «2011_2012_S1D_cours_1.pdf» sauvegardé
[4568618/4568618]

[ login@localhost ~ ] ls -l ./2011_2012_S1D_cours_1.pdf
-rw-r--r-- 1 santini users 4,4M 2011-12-14 10 :33 ./2011_2012_S1D_cours_1.pdf
```

diff

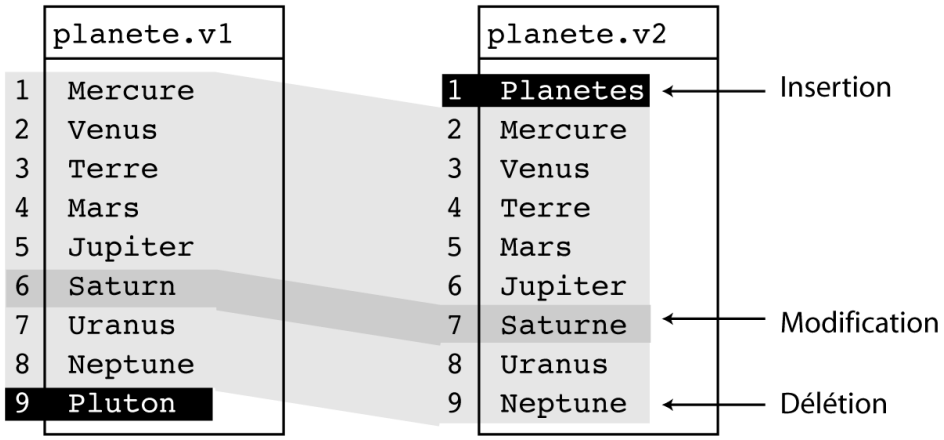
Syntaxe

```
diff  fichier_1 fichier_2
```

Description

- Compare deux fichiers, localise et affiche les différences (très utile pour suivre l'évolution d'un code).
- identifie les insertions, délétions et modifications.

Exemple d'utilisation:




```
[ login@localhost ~ ] diff planete.v1 planete.v2
0a1
> Planetes
6c7
< Saturn
---
> Saturne
9d9
< Pluton

[ login@localhost ~ ] ■
```

Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

15 février 2012

Partie #4

Plan

Métacaractère et Chemins ciblés

Le caractère *

- Le cataractère * est utilisé comme un *jocker* pour remplacer une chaîne de caractères,
- Il est utilisé pour pointer plusieurs fichiers ou répertoires dont le nom partage un motif commun.
- Le caractère * peut être placé en début, en fin ou au milieu d'une chaîne de caractères,
- Le caractère * peut être répété.

Exemple de manipulation avec la commande mv

```
[ login@localhost ~ ] mv *.jpg Images/
```

Ici, le chemin `*.jpg` pointe tous les fichiers du répertoire courant dont le nom se fini par l'extension `.jpg`. Il pointe donc les fichiers `etacentauri.jpg` et `aldebaran.jpg` et exclue les autres fichiers (ici le fichier `alphacentauri.gif`).

```
chez_moi/.............. Répertoire Courant
├── aldebaran.jpg ..... Fichier ciblé
├── alphacentauri.gif
├── etacentauri.jpg ..... Fichier ciblé
└── Images/ ..... Répertoire final
```

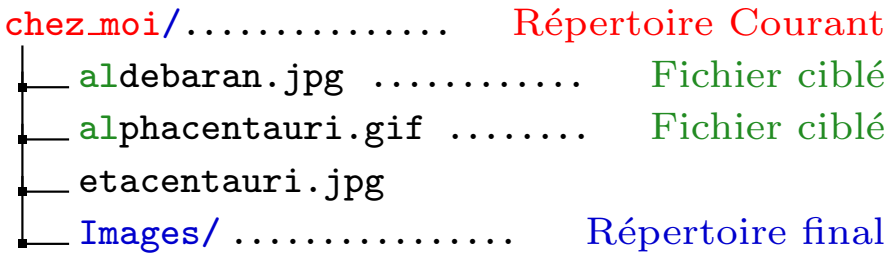
```
chez_moi/.............. Répertoire Courant
├── alphacentauri.gif
└── Images/ ..... Répertoire final
    ├── aldebaran.jpg ..... Fichier déplacé
    └── etacentauri.jpg ... Fichier déplacé
```

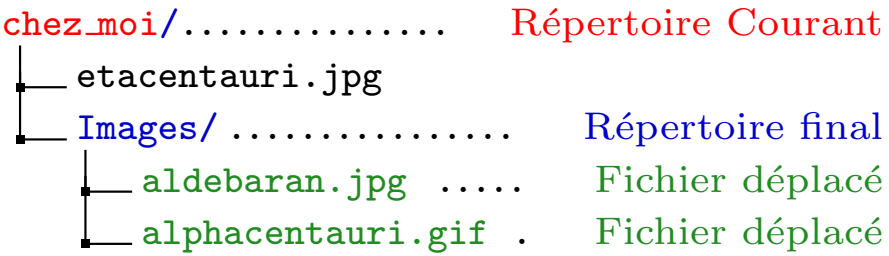
Métacaractère et Chemins ciblés

Exemple de manipulation avec la commande mv

```
[ login@localhost ~ ] mv al* Images/
```

Ici, le chemin `al*` pointe tous les fichiers du répertoire courant dont le nom commence par les caractères `al`. Il pointe donc les fichiers `aldebaran.jpg` et `alphacentauri.gif` et exclue les autres fichiers (ici le fichier `etacentauri.jpg`).

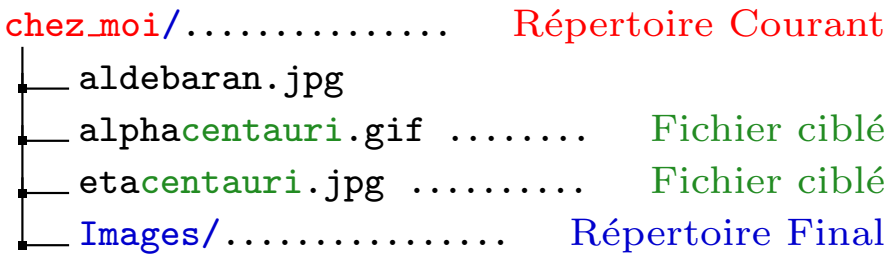


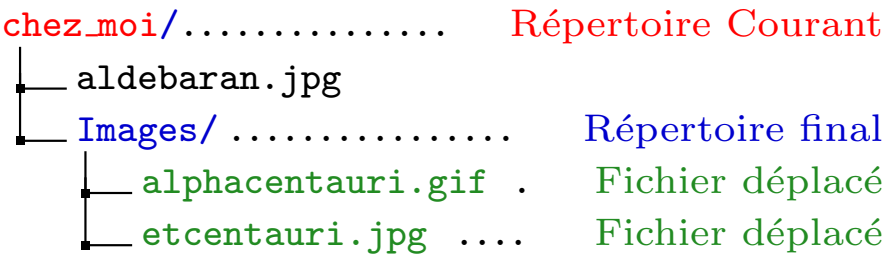


Exemple de manipulation avec la commande mv

```
[ login@localhost ~ ] mv *centauri* JPG/
```

Ici, le chemin `*centauri*` pointe tous les fichiers du répertoire courant dont le nom contient la chaîne de caractères `centauri`. Il pointe donc les fichiers `alphacentauri.gif` et `etacentauri.jpg` et exclue les autres fichiers (ici le fichier `aldebaran.jpg`).

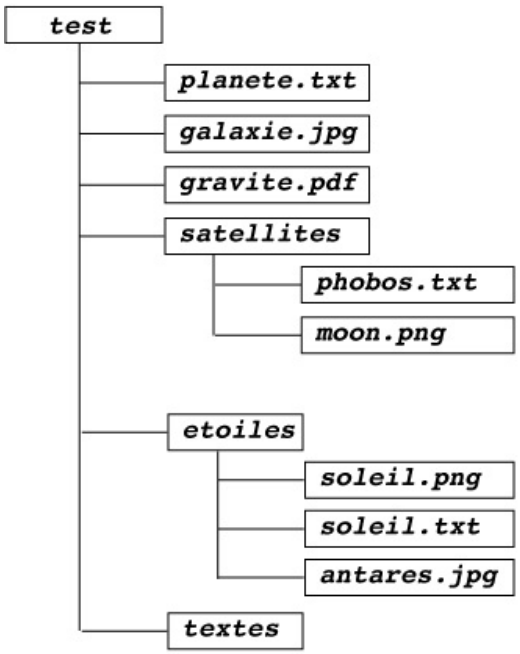


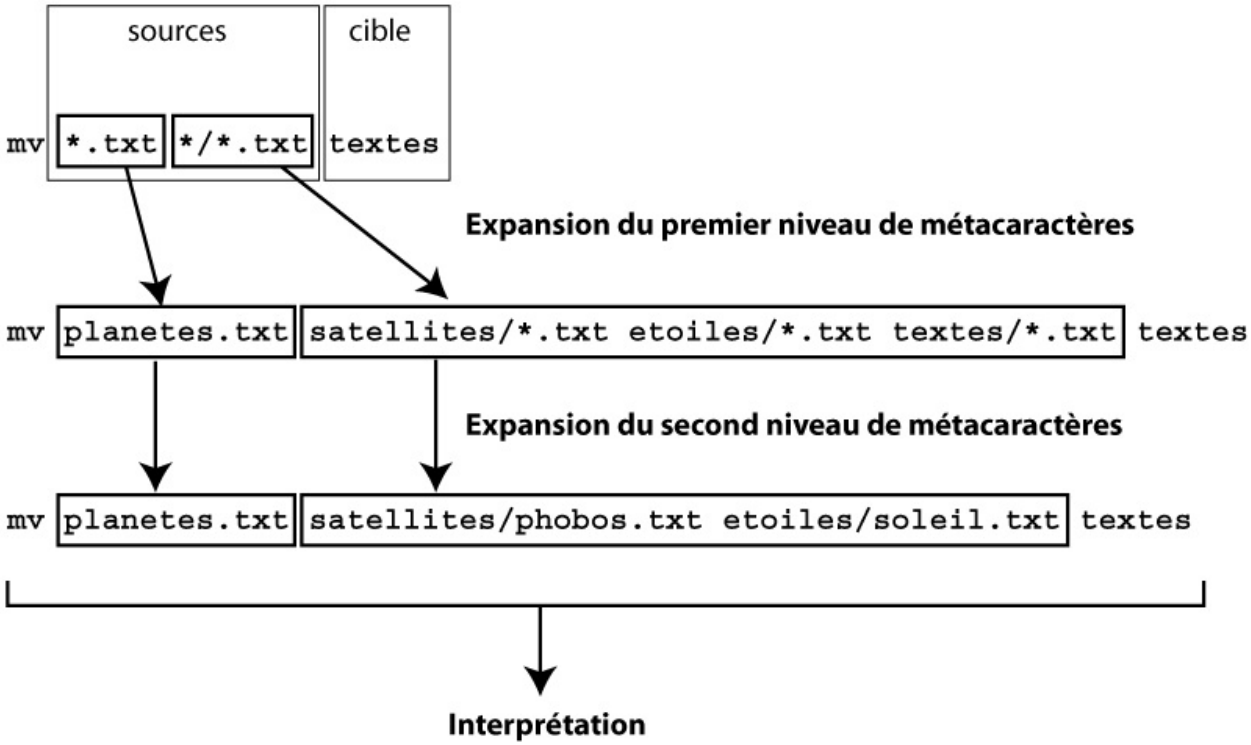


Métacaractère et Chemins ciblés

Exemple plus complexe et Détails de l'interprétation

- Le cararctère ∗ est développé lors de l'interprétation.





find

Syntaxe

```
find  depart -iname "motif"
```

Description

- Recherche dans les répertoires et sous-répertoires les fichiers dont le nom correspond au motif en partant du point de l'arborescence spécifié par le depart.
- L'option -iname indique que le motif sera recherché sans tenir compte des majuscules et minuscules.

Exemple d'utilisation:

```
chez_moi/ . Répertoire courant
├── Mes_Images/
│   ├── JPG/
│   │   ├── soleil.jpg
│   │   └── lune.jpg
│   └── GIF/
│       ├── alphacentauri.gif
│       └── terre.gif
└── Mes_Photos/
    ├── etacentauri.jpg
    └── phobos.gif
```

```
[ login@localhost ~ ] find . -iname *.gif
./Mes_Images/GIF/alphacentauri.gif
./Mes_Images/GIF/terre.gif
./Mes_Photos/phobos.gif

[ login@localhost ~ ] find . -iname *centauri*
./Mes_Images/GIF/alphacentauri.gif
./Mes_Photos/etacentauri.jpg

[ login@localhost ~ ] find Mes_Images/ -iname *e.*
Mes_Images/GIF/terre.gif
Mes_Images/JPG/lune.jpg

[ login@localhost ~ ] ■
```

find(bis)

Syntaxe

```
find  depart -iname "motif" -exec commande \;
```

Description

- Exécute la commande sur la liste des fichiers identifiés par `find`,
- Dans la rédaction de la commande, la liste des fichiers est symbolisée par les caractères {}.

Exemple d'utilisation:

```
[ login@localhost ~ ] find ./ -iname *centauri* -exec mv {} Mes_Photos \;
```

chez_moi/.....État Initial Répertoire courant

```
Mes_Images/
    JPG/
        etacentauri.jpg...Trouvé par find
        soleil.jpg
    GIF/
        alphacentauri.gif.Trouvé par find
        terre.gif
Mes_Photos/
```

chez_moi/.....État Final Répertoire courant

```
Mes_Images/
    JPG/
        soleil.jpg
    GIF/
        terre.gif
Mes_Photos/
    alphacentauri.jpg.....Déplacé par mv
    etacentauri.jpg.......Déplacé par mv
```

Liens symboliques

Définition

- C'est une entrée spéciale contenue dans la liste des références (fichiers ou répertoires) d'un répertoire qui pointe vers une autre référence (fichier ou répertoire) déjà existante dans l'arborescence du système de fichiers.
- Autrement dit, c'est un *alias* placé dans un répertoire qui fait référence à un autre fichier ou répertoire de l'arborescence (où qu'il soit).
- C'est un *"raccourci"* placé dans l'arborescence.

Manipulation

- Lors de la création d'un lien symbolique, seule la référence est copiée. Les données pointées par la référence n'existent toujours qu'en un seul exemplaire.
- Le même fichier sera accessible par son chemin d'origine ou par le chemin de l'*alias* (*i.e.* le chemin du lien symbolique).
- La destruction du lien symbolique n'entraine pas la destruction du fichier original.
- Plusieurs liens symboliques peuvent pointer le même fichier ou le même répertoire.

ln

Syntaxe

```
ln -s source cible
```

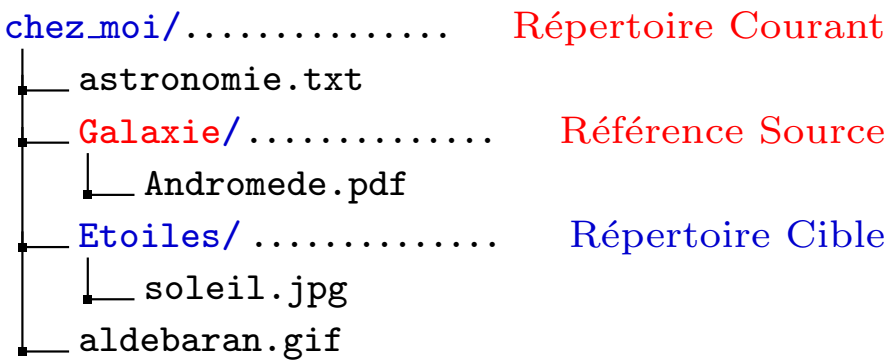
Description

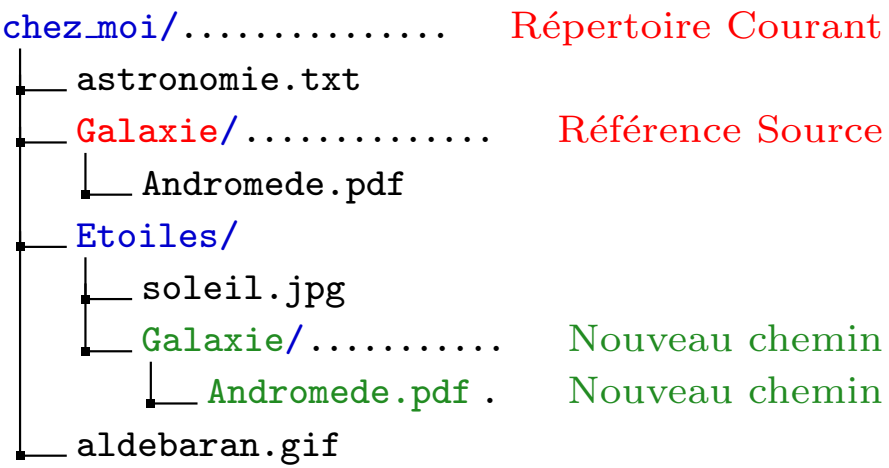
- Crée un lien symbolique entre la référence source et le chemin cible..

Exemple d'utilisation:

```
[ login@localhost ~ ] ln -s Galaxies Etoiles/Galaxies
```

Le lien symbolique sur un répertoire donne également accès à toutes les références contenues dans le répertoire pointé par le lien. Ainsi, le fichier `~/Galaxie/Adromede.pdf` est aussi accessible par le chemin `~/Etoiles/Galaxie/Andromede.pdf`.



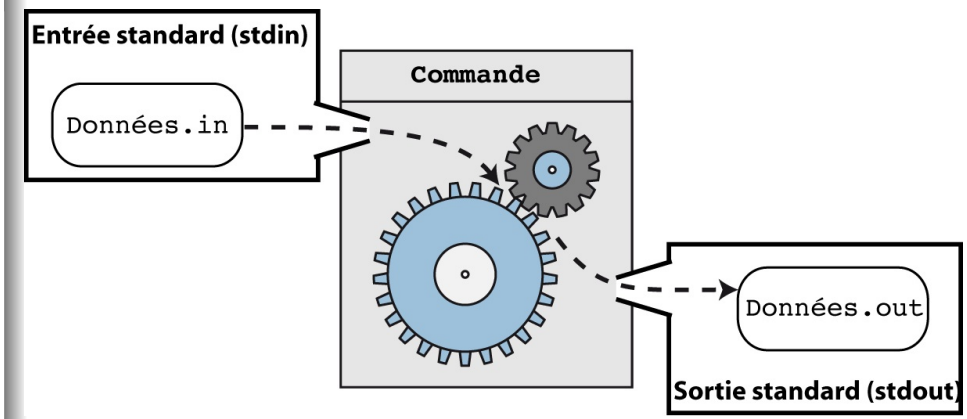


Plan

Entrée et sortie standard

Rappel : Les programmes informatiques

- Un programme prend des données en entrée. Ces données peuvent être lues dans un fichier ou fournies par un flux du système.
- Le programme manipule ces données.
- Le programme fournit un résultat en sortie (des données). Ces données peuvent être écrites dans un fichier ou exportées comme un flux vers le système.



Les flux de données

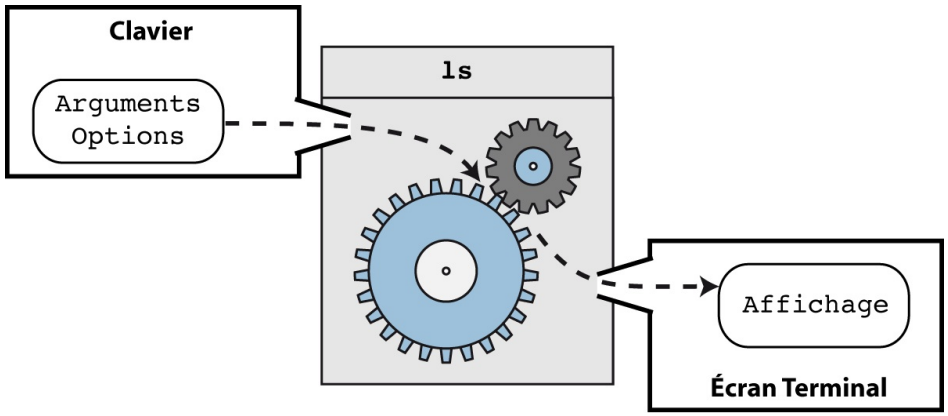
Pour fonctionner, un programme a donc besoin de lire des données (flux d'entrée : input) et d'écrire les résultats de ses évaluations (flux de sortie : output). On distingue 3 types de flux de données :

- **STDIN** : entrée standard (là où sont lues les données),
- **STDOUT** : sortie standard (là où sont écrits les résultats),
- **STDERR** : sortie erreur (là où sont écrit les messages d'erreur).

Entrée et sortie standard

Les commandes qui lisent sur l'entrée standard

- Certaines commandes Linux qui traitent les données d'un fichier (dont le chemin est passé en paramètre) peuvent alternativement, si aucun chemin fichier n'est spécifié, travailler directement avec les données lues sur l'entrée standard.
- Parmi les commandes déjà vue : `cat`, `head`, `tail`, `cut`, `sed`, `grep`.
- **Par défaut, l'entrée standard est le clavier**.



Les commandes qui écrivent sur la sortie standard

- Les affichages produits par les commandes Linux sont le résultat de leur évaluation. Ce résultat est écrit sur la sortie standard.
- **Par défaut, la sortie standard est l'écran**.

Redirection des Entrée/Sorties

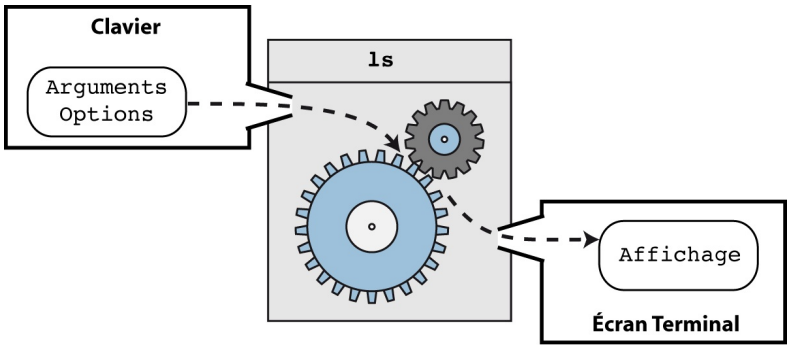
Commandes de Redirection

Il est possible de modifier le comportement par défaut des commandes et de donner une entrée et/ou une sortie standard différente des entrées/sorties standards.

- `command > fichier.out`
  - **Redirige la sortie standard** de la commande `command` vers le fichier `fichier.out`.
  - Si le fichier `fichier.out` n'existe pas, il est créé avec comme contenu les affichages produits par la commande `command`.
  - **Si le fichier `fichier.out` existe, son contenu est écrasé** et remplacé par les affichages produits par la commande `command`.
- `command >> fichier.out`
  - **Redirige la sortie standard** de la commande `command` vers le fichier `fichier.out`.
  - Si le fichier `fichier.out` n'existe pas, il est créé avec comme contenu les affichages produits par la commande `command`.
  - Si le fichier `fichier.out` existe, les affichages produits par la commande `command` sont **ajoutés à la fin du contenu du fichier**.
- `command 2> fichier.err`
  - **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec écrasement du contenu** si le fichier de sortie existe déjà.
- `command 2>> fichier.err`
  - **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec préservation du contenu** si le fichier de sortie existe déjà.

Exemple de redirection

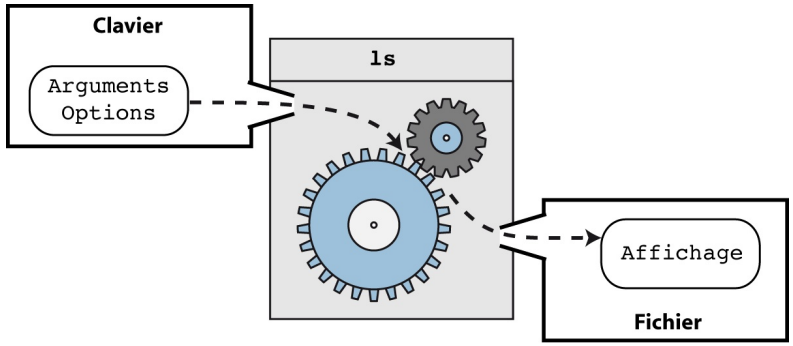
Comportement par défaut de la commande `ls`




```
[ login@localhost ~ ] ls
aldenaran.jpg alphacentauri.gif etacentauri.jpg

[ login@localhost ~ ] ls
aldenaran.jpg alphacentauri.gif etacentauri.jpg

[ login@localhost ~ ] ■
```

La sortie standard de la première commande `ls` est l'écran. La liste du contenu du répertoire courant est affichée à l'écran.

Redirection de la sortie de la commande `ls`



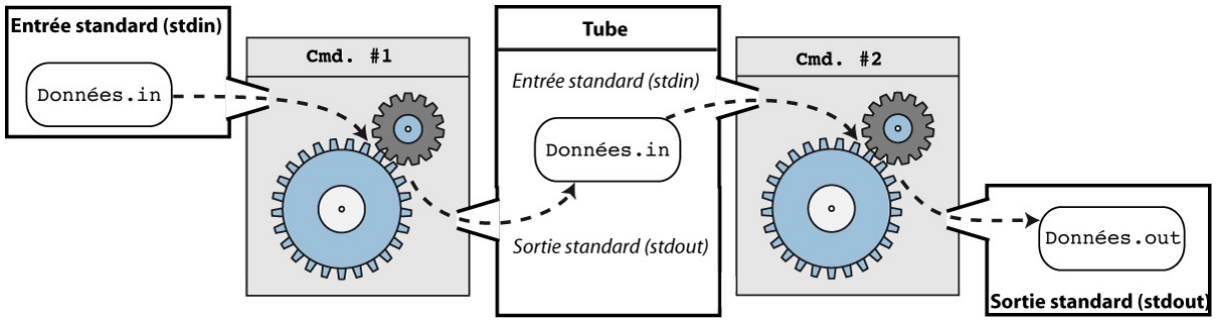

```
[ login@localhost ~ ] ls > 1.out


[ login@localhost ~ ] ls
1.out aldenaran.jpg alphacentauri.gif etacentauri.jpg

[ login@localhost ~ ] ■
```

La sortie standard de la première commande `ls` est redirigée vers le fichier `1.out`. La liste du contenu du répertoire courant est écrite dans le fichier `1.out`.
La deuxième commande `ls`, montre qu'un fichier portant le nom `1.out` a été créé.

Tubes

Principes de fonctionnement des Tubes (Pipe en anglais)

- A la différence des redirections simples qui permettent de rediriger la sortie standard d'une commande vers un fichier,
- Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.



Syntaxe

- Le tube est symbolisé par le caractère |.
- `cmd1 | cmd2`
  - La sortie standard de la première commande (`cmd1`) est redirigée vers l'entrée standard de la deuxième commande (`cmd2`).
  - L'entrée standard de la commande `cmd1` et la sortie standard de la commande `cmd2` ne sont pas modifiées.

Exemple de Tubes avec les commande `ls` et `more`

Rappel des commandes :

- `ls` affiche à l'écran (stdout) la liste des fichiers contenus dans un répertoire.
- `more` affiche page par page le contenu des données passée sur son entrée standard.

Exemple #1

- Si de très nombreux fichiers sont contenus dans un répertoire, la commande `ls` peut produire un affichage qui ne tient pas dans l'écran, rendant impossible le parcours de la liste des fichiers (seuls les derniers sont visibles).

```
[ login@localhost ~ ] ls
```

Défilement de tous les fichiers

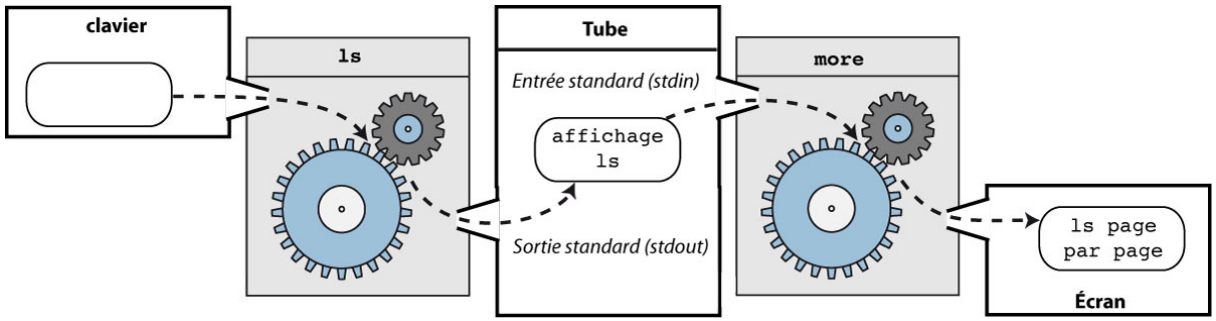
```
betelgeuse.jpg    etacentauri.jpg
soleil.jpg        syrius.gif
vega.png
 [ login@localhost ~ ] ■
```

```
Images/ ..... Répertoire courant
├── aldebaran.jpg ..... Hors de la fenetre
├── alphacentauri.gif ..... Hors de la fenetre
├── betelgeuse.jpg ..... Dans la fenetre
├── etacentauri.jpg ..... Dans la fenetre
├── soleil.jpg ..... Dans la fenetre
├── syrius.gif ..... Dans la fenetre
└── vega.png ..... Dans la fenetre
```

Exemple de Tubes avec les commande `ls` et `more`

Exemple #1 (suite) :

- La redirection de la sortie standard de la commande `ls` vers l'entrée standard de la commande `more` permet de passer en revue l'affichage de la commande `ls` page par page.




```
[ login@localhost ~ ] ls | more
aldebaran.jpg      alphacentauri.gif
betelgeuse.jpg     etacentauri.jpg
soleil.jpg         syrius.gif
```

Affichage d'une première page puis
Presser la touche Enter pour la page suivante

```
soleil.jpg         syrius.gif
vega.png
 [ login@localhost ~ ] ■
```

```
Images/ ..... Répertoire courant
├── aldebaran.jpg ..... Page 1
├── alphacentauri.gif ..... Page 1
├── betelgeuse.jpg ..... Page 1
├── etacentauri.jpg ..... Page 1
├── soleil.jpg ..... Page 1&2
├── syrius.gif ..... Page 1&2
└── vega.png ..... Page 2
```

Exemple de Tubes avec les commande ls et grep

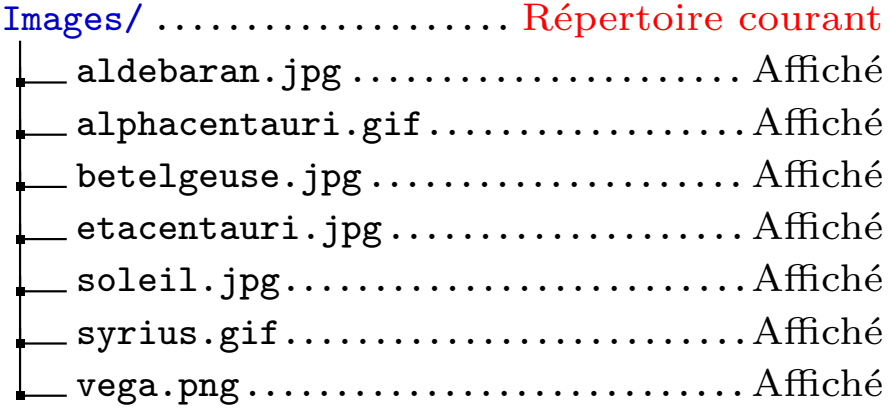
Rappel des commandes :

- `ls` affiche à l'écran (stdout) la liste des fichiers contenus dans un répertoire.
- grep affiche les lignes d'un texte qui comportent un certain motif.

Exemple #2 :

- Si de très nombreux fichiers sont contenus dans un répertoire, la commande `ls` peut produire un affichage qui ne tient pas dans l'écran, rendant compliqué l'identification de certain type de fichier (fichiers au format `gif` par exemple).

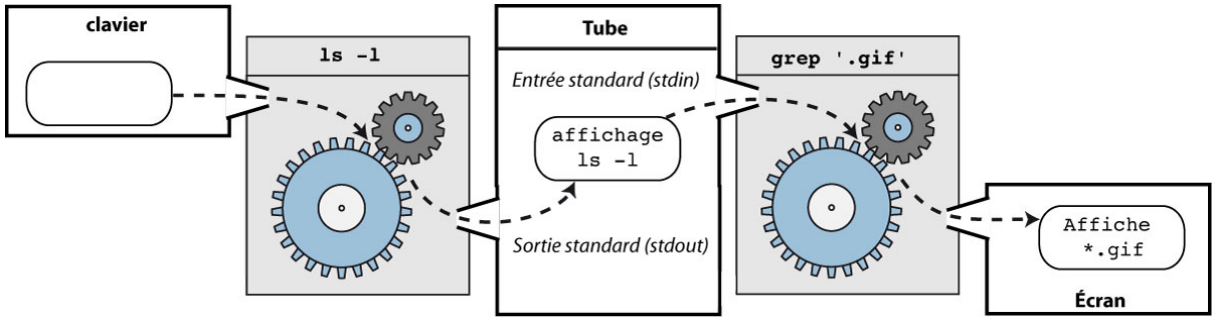
```
[ login@localhost ~ ] ls
aldebaran.jpg
alphacentauri.gif
betelgeuse.jpg
etacentauri.jpg
soleil.jpg
syrius.gif
vega.png

[ login@localhost ~ ] ■
```



Exemple de Tubes avec les commande `ls` et `more`

Exemple #2 (suite) :

- La redirection de la sortie standard de la commande `ls` vers l'entrée standard de la commande `grep` permet d'effectuer un filtrage des fichiers présents dans le répertoire sur la base d'un motif présent dans leur nom (par exemple l'extension `.gif`).




```
[ login@localhost ~ ] ls | grep '.gif'
alphacentauri.gif
syrius.gif

[ login@localhost ~ ] ■
```

```
Images/ ..... Répertoire courant
├── aldebaran.jpg ..... Retenu par le filtre
├── alphacentauri.gif ..... Affiché
├── betelgeuse.jpg ..... Retenu par le filtre
├── etacentauri.jpg ..... Retenu par le filtre
├── soleil.jpg ..... Retenu par le filtre
├── syrius.gif ..... Affiché
└── vega.png ..... Retenu par le filtre
```

Exemple de Tubes avec la commande cut

Rappel des commandes :

- cut permet d'afficher une colonne donnée d'un texte. Pour cela on spécifie le numéro de la colonne et le séparateur de colonnes.

Exemple #3 :

- Extraction d'une colonne d'un fichier formaté dans lequel les séparateurs de colonne ne sont pas homogènes.

Par exemple soit le fichier suivant dont on souhait extraire la liste des planètes :

comparatif.txt

```
Planetes<Nom>Masse(10E+18t)
Tellurique<Mercure>330
Tellurique<Venus>4871
Jovienne<Uranus>86760
Jovienne<Neptune>103000
```

Premier essai :
choisir le séparateur '>' et sélectionner la première colonne :

```
[ login@localhost ~ ] cut -d '>' -f 1 comparatif.txt
Planetes<Nom
Tellurique<Mercure
Tellurique<Venus
Jovienne<Uranus
Jovienne<Neptune

[ login@localhost ~ ] ■
```

C'est un echec ! ! !

Exemple de Tubes avec la commande cut

Rappel des commandes :

- cut permet d'afficher une colonne donnée d'un texte. Pour cela on spécifie le numéro de la colonne et le séparateur de colonnes.

Exemple #3 :

- Extraction d'une colonne d'un fichier formaté dans lequel les séparateurs de colonne ne sont pas homogènes.

Par exemple soit le fichier suivant dont on souhait extraire la liste des planètes :

comparatif.txt

```
Planetes<Nom>Masse(10E+18t)
Tellurique<Mercure>330
Tellurique<Venus>4871
Jovienne<Uranus>86760
Jovienne<Neptune>103000
```

Deuxième essai :
choisir le séparateur '<' et sélectionner la deuxième colonne :

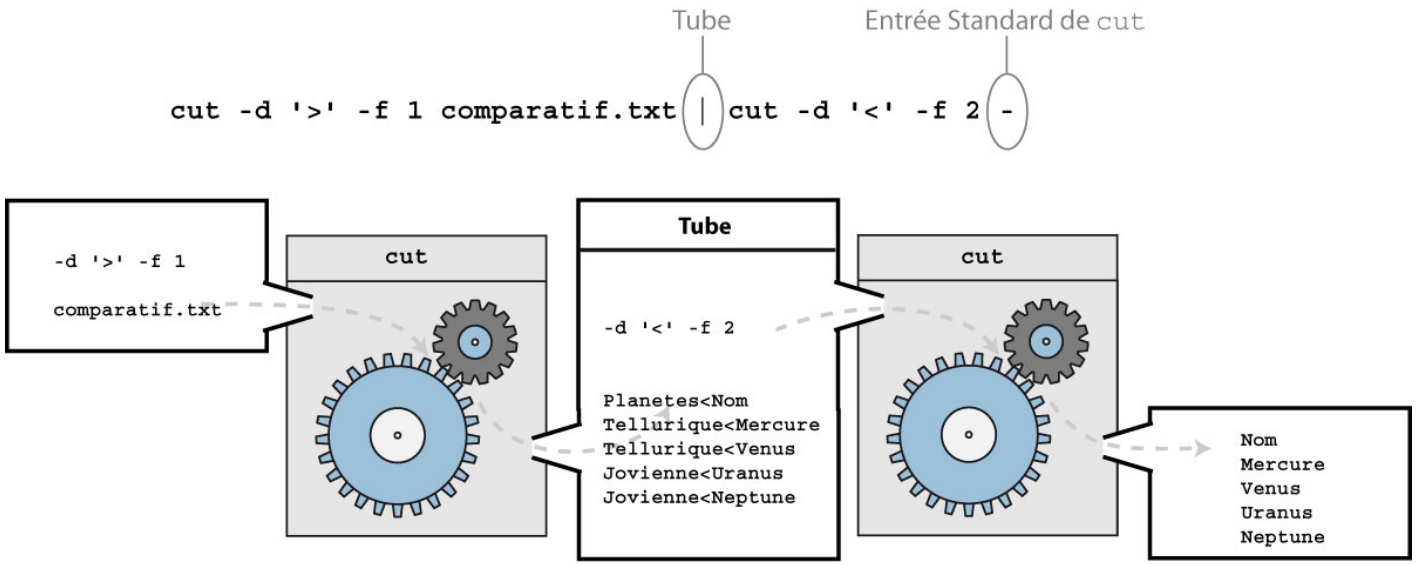
```
[ login@localhost ~ ] cut -d '<' -f 2 comparatif.txt
Nom>Masse(10E+18t)
Mercure>330
Venus>4871
Uranus>86760
Neptune>103000

[ login@localhost ~ ] ■
```

Encore un echec ! ! !

Exemple de Tubes avec la commande cut

La solution : Décomposer en étapes simples et utiliser un tube

- Lors d'une première passe, prendre la première colonne en utilisant le séparateur >, puis
- Lors de la deuxième passe, prendre la deuxième colonne en utilisant le séparateur <.




```
[ login@localhost ~ ] cut -d '>' -f 1 comparatif.txt | cut -d '<' -f 2 -
Nom
Mercure
Vebus
Uranus
Neptune

[ login@localhost ~ ] ■
```

Plan

Introduction

Rappels sur le langage bash

- Langage interprété permettant d'adresser des commandes au système.
- Comme tout langage il est caractérisé par un vocabulaire (le nom des commandes), une grammaire (enchaînement des commandes, des options et des paramètres) et d'une syntaxe (caractères de séparation).
- Il permet à l'utilisateur de définir des programmes pour les besoins spécifiques de certains utilisateurs.

La programmation

Écrire un programme c'est définir un ensemble d'instructions pour réaliser une tâche complexe qu'une seule commande ne peut faire seule. Pour cela il faut pouvoir :

- Regrouper ensemble plusieurs instructions (script),
- Séparer les différentes instructions (séparateurs),
- Définir des variables pour définir des instructions génériques dont le résultat de l'interprétation sera paramétré par les différentes valeurs prises par les variables,
- Exécuter l'ensemble de ces instructions . . .

Les variables

Définition

- Une variable est une chaîne de caractères à laquelle est associée une valeur.
- La chaîne de caractères est le nom de la variable.
- Pour faire appel à la valeur il suffit de taper le nom de la variable.
- Lorsque l'on donne un nom de variable comme paramètre à une instruction, celle-ci est remplacée lors de l'interprétation de la commande par sa valeur.
- On appelle cela *une variable*, car la valeur associée à un nom de variable peut changer ...
- On appel *affectation* le fait d'associer une valeur à un nom de variable.

Utilisation

- Les variables sont utilisées pour stocker des valeurs qui doivent être utilisée par la suite, souvent plusieurs fois.
- Elles permettent de définir des instructions dont le comportement sera différent selon la valeur de la variable à l'instant de l'évaluation.

Exemple d'utilisation de variables

Définition et appel d'une variable

- Le signe = permet de réaliser l'affectation selon la syntaxe suivante (Attention : il ne faut pas mettre de caractère espace autour du signe d'affectation '=') :

```
nom_de_la_variable=valeur
```

- Pour accéder au contenu d'un variable, on fait précéder son nom du caractère $.

```
$nom_de_la_variable
```

Se placer dans un répertoire souvent utilisé

Pour sélectionner comme répertoire courant un répertoire dans lequel on se place souvent et dont le nom est long a taper, on peut définir une variable contenant le nom du répertoire, et utiliser le nom de la variable en lieu et place du nom complet du répertoire :

```
[ login@localhost ~ ] DirTP1=/home/chez_moi/TP_Intro_Systeme/TP_1

[ login@localhost ~ ] cd $DirTP

[ login@localhost ~/TP_Intro_Systeme/TP_1 ] ■
```

echo

Syntaxe

```
echo  expression
```

Description

- Affiche sur la sortie standard l'expression après interprétation.

Exemple d'utilisation:

Affiche 'Bonjour' :

```
[ login@localhost ~ ] echo Bonjour
Bonjour
[ login@localhost ~ ] ■
```

Définie une variable puis affiche sa valeur :

```
[ login@localhost ~ ] Astre=Terre
[ login@localhost ~ ] echo $Astre
Terre
[ login@localhost ~ ] echo La planete $Astre
La planete Terre
[ login@localhost ~ ] ■
```

Exemple d'utilisation de variables

Modification de la valeur d'une variable

- Pour modifier la valeur d'une variable il faut lui affecter une nouvelle valeur.
- La syntaxe est la même que celle de la définition de la valeur d'une variable (on utilise le signe d'affectation '=').

Afficher des messages personnalisés

Cet exemple utilise deux variables `Cible` et `Entete`. La même instruction `echo $Entete $Cible` affiche des messages différents car la valeur de la variable `Entete` a été modifiée entre les deux évaluations. Cet exemple montre :

- comment on modifie la valeur d'une variable et
- comment une même instruction paramétrée par des variables peut conduire à des résultats différents selon la valeur des variables.

```
[ login@localhost ~ ] Cible='A tous'

[ login@localhost ~ ] Entete='Bonjour'

[ login@localhost ~ ] echo $Entete $Cible
Bonjour A tous
```

```
[ login@localhost ~ ] Entete='Salut'

[ login@localhost ~ ] echo $Entete $Cible
Salut A tous
[ login@localhost ~ ] ■
```

; Le séparateur de commandes

Écrire plusieurs instructions sur la même ligne de commande

- Il est possible d'écrire plusieurs instructions sur la même ligne de commande.
- Pour séparer les différentes instructions on utilise le caractère de ponctuation ;
- Les instructions seront évaluées dans l'ordre d'écriture, et une instruction ne sera évaluée qu'après que la précédente ait terminée sont évaluation.

Exemple d'instruction multiples sur la même ligne de commande

```
[ login@localhost ~ ] Dir='Mes_Images' ; cd $Dir
~/Mes_Images

[ login@localhost ~/Mes_Images ] echo $Dir ; Dir='~/Mes_Photos'
~/Mes_Images

[ login@localhost ~/Mes_Images ] cd $Dir ; echo $Dir
~/Mes_Photos

[ login@localhost ~/Mes_Photos ] ■
```

Les scripts bash : Présentation

Les scripts bash

- Ce sont des fichiers textes qui regroupent un ensemble d'instructions.
- Ce sont des programmes rédigés et interprétables par le langage bash.
- Ils regroupent dans un même fichier (réutilisable) un ensemble d'instructions qui pourraient être tapées sur la ligne de commande.
- Ces programmes peuvent admettre des paramètres.
- Ils peuvent être lancés depuis la ligne de commande.

mon_premier_script.sh

```
#!/bin/bash

instruction_1
instruction_2
...
...
instruction_n
```

Le nom des scripts

- Le nom donné au fichier texte contenant le script doit être *expressif*.
- Traditionnellement, l'extension donnée à un fichier bash est `.sh`

Alternatives possibles

Le bash est l'interpréteur de commande développé dans le cadre du projet GNU. Il existe d'autres interpréteurs historiques qui sont également disponibles dans la plupart des distributions Linux.

- csh, tcsh, ksh, zsh, ...

Les scripts

Structure d'un script

- Sur la première ligne du script on place une entête indiquant le chemin de l'interpréteur `#!/bin/bash` suivit d'une ligne vide.
- Dans le script, chaque ligne correspond à une instruction,
- Sur une ligne, tout texte placé après le caractère # est considéré comme un commentaire et n'est pas interprété.

mon_premier_script.sh

```
#!/bin/bash

message="Aller au répertoire : "      # Définition de la variable message
dir=~/Intro_Systeme/                  # Définition de la variable dir
echo $message $dir                    # Affichage de l'action
cd $dir                               # Changement de répertoire courant
```

Évaluation des commandes du script

- Les instructions d'un script sont évaluées dans leur ordre d'écriture. Par exemple, ici, la commande `echo $message $dir` est évaluée avant la commande `cd $dir`.
- Une commande d'un script n'est évaluée qu'après la fin de l'exécution de la commande précédente - *i.e.* évaluation séquentielle des commandes.

Les scripts

Cas d'étude

Soit le script `mon_premier_script.sh` et son emplacement dans l'arborescence des fichiers :

mon_premier_script.sh

```
#!/bin/bash

message="Aller au répertoire : "
dir=~/Intro_Systeme/
echo $message $dir
cd $dir
```

```
chez_moi/ ..... Répertoire Courant
├── mes_scripts/
│   └── mon_premier_script.sh ..... Le script
└── Intro_Systeme/
    ├── TP_1/
    │   └── ...
    └── TP_2/
        └── ...
```

Exécuter un script (1)

- Pour rendre un script *exécutable*, il faut donner les droits d'exécution sur le fichier (`chmod`).

```
[ login@localhost ~ ] chmod u+x ~/mes_scripts/mon_premier_script.sh

[ login@localhost ~ ] ls -l ~/mes_scripts/mon_premier_script.sh
 -rwxr--r-- 2 santini ensinfo 93 18 jul 12 :59 mon_premier_script.sh

[ login@localhost ] ■
```

Les scripts

Cas d'étude

Soit le script `mon_premier_script.sh` et son emplacement dans l'arborescence des fichiers :

mon_premier_script.sh

```
#!/bin/bash

message="Aller au répertoire : "
dir=~/Intro_Systeme/
echo $message $dir
cd $dir
```

```
chez_moi/.....Répertoire Courant Initial
├── mes_scripts/
│   └── mon_premier_script.sh ..... Le script
└── Intro_Systeme/..Répertoire Courant Final
    ├── TP_1/
    │   └── ...
    └── TP_2/
        └── ...
```

Exécuter un script (2)

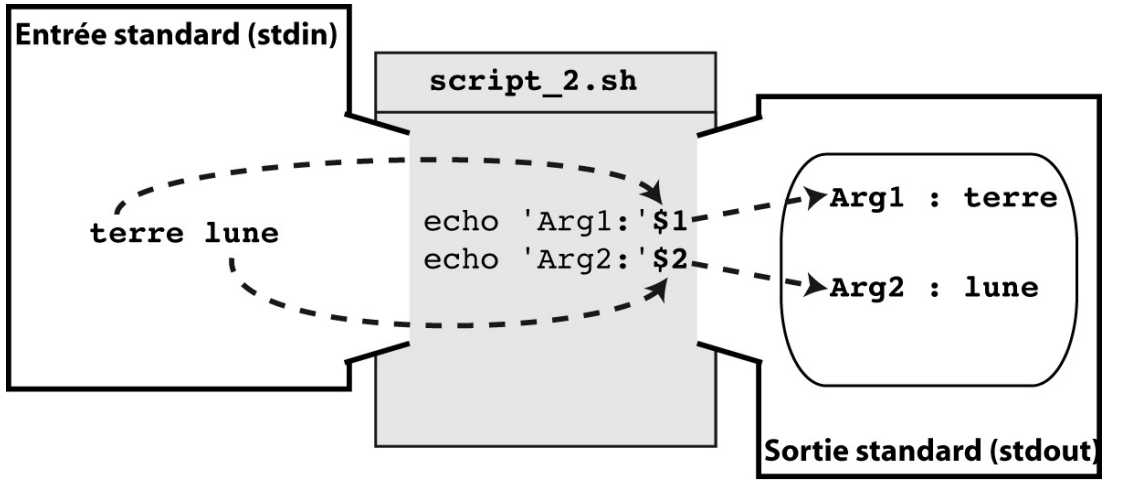
- Pour exécuter un script il suffit ensuite de taper son chemin -*i.e.* le chemin vers le fichier contenant le script-.

```
[ login@localhost ~ ] mes_scripts/mon_premier_script.sh
Aller au répertoire :~/Intro_Systeme

[ login@localhost ~/Intro_Systeme ] pwd
~/Intro_Systeme

[ login@localhost ~/Intro_Systeme ] ■
```

Les paramètres des scripts

Passage des paramètres

- Dans un script, il est possible de récupérer des informations passées sur la ligne de commande.
- Ces informations sont passées au script comme des variables.
- Ces variables peuvent être utilisées dans les instructions.




```
[ login@localhost ~ ] ./script_2.sh terre lune
Arg1 : terre
Arg2 : lune

[ login@localhost ~ ] ■
```

Appel de la variable	Valeur de la variable
$0	Le nom du script
$1 à $9	Les (éventuels) premiers arguments passés à l'appel du script
$#	Le nombre d'arguments passés au script
$*	La liste des arguments à partir de $1

Les paramètres des scripts

Exemple de scripts avec passage de paramètres

Soit le script suivant :

affiche_param.sh

```
#!/bin/bash

echo "Nom du script :" $0
echo "Nombre de parametres :" $#
echo "Premier parametre :" $1
echo "Deuxieme parametre :" $2
```

Son exécution produit les affichages suivants :

```
[ login@localhost ~ ] ./affiche_param.sh terre lune soleil
Nom du script : affiche_param.sh
Nombre de parametres : 3
Premier parametre : terre
Deuxieme parametre : lune

[ login@localhost ~ ] ■
```

Plan

Caractères spéciaux

Définition

Ce sont des caractères qui :

- peuvent apparaitre sur la ligne de commande,
- prennent un sens particulier lors de l'interprétation

Caractères spéciaux déjà vus

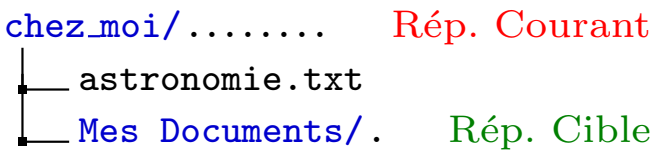
Caractère	Interprétation
>	Redirection de la sortie standard d'une commande.
<	Redirectiond e l'entrée standard d'une commande.
|	Tube : redirection de la sortie standard d'une commande vers l'entrée standard d'une autre commande.
*	Complétion d'un nom de fichier
$	Appel de la valeur d'une variable
;	Fin de commande (permet de taper plusieurs commandes sur la même ligne)
␣	Séparateur de paramètres
"	Délimiteur de texte
#	Commentaires

Caractères spéciaux

L'échappement

- Permet d'échapper au sens donné aux caractères spéciaux par l'interpréteur.
- Redonne leur sens littérale aux caractères spéciaux.

Exemple : Taper un nom de répertoire contenant un espace




```
[ login@localhost ~ ] cd Mes Documents
-bash : cd : Mes : No such file or directory

[ login@localhost ~ ] ■
```

Problème

- Le répertoire `'Mes Documents'` n'est pas trouvé,
- Le caractère 'espace' compris dans le nom du répertoire est interprété comme un séparateur de paramètre.
- Le répertoire `'Mes'` n'existe pas dans le répertoire courant...

Échapper à l'interprétation

Plusieurs modes d'échappement

Échappement	Caractère	syntaxe
Ponctuel	Back Slash	`cd Mes\ Documents`
Complet	Guillemets Simples	`cd 'Mes Documents'`
Partiel	Guillemets Doubles	`cd "Mes Documents"`

Fonctionnement

Ponctuel Le caractère qui suit le Back Slash (\) échappe à l'interprétation (reprend son sens littérale).

Complet Tous les caractères compris entre les Guillemets Doubles échappent à l'interprétation (reprennent leur sens littérale).

Partiel Tous les caractères compris entre les Guillemets Doubles échappent à l'interprétation (reprennent leur sens littérale), sauf le caractère $. Du coup, les noms de variables sont interprétés et remplacé par leur valeur.

Échapper à l'interprétation

Exemples d'échappement Ponctuel

```
[ login@localhost ~ ] PLANETE=Terre

[ login@localhost ~ ] echo $PLANETE # $PLANETE
Terre
[ login@localhost ~ ] echo \$PLANETE \# $PLANETE
$PLANETE # Terre
```

Exemples d'échappement Partiel et Complet

```
[ login@localhost ~ ] PLANETE=Terre

[ login@localhost ~ ] echo "$PLANETE # $PLANETE"
Terre # Terre
[ login@localhost ~ ] echo '$PLANETE # $PLANETE'
$PLANETE # $PLANETE
```

Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

15 février 2012

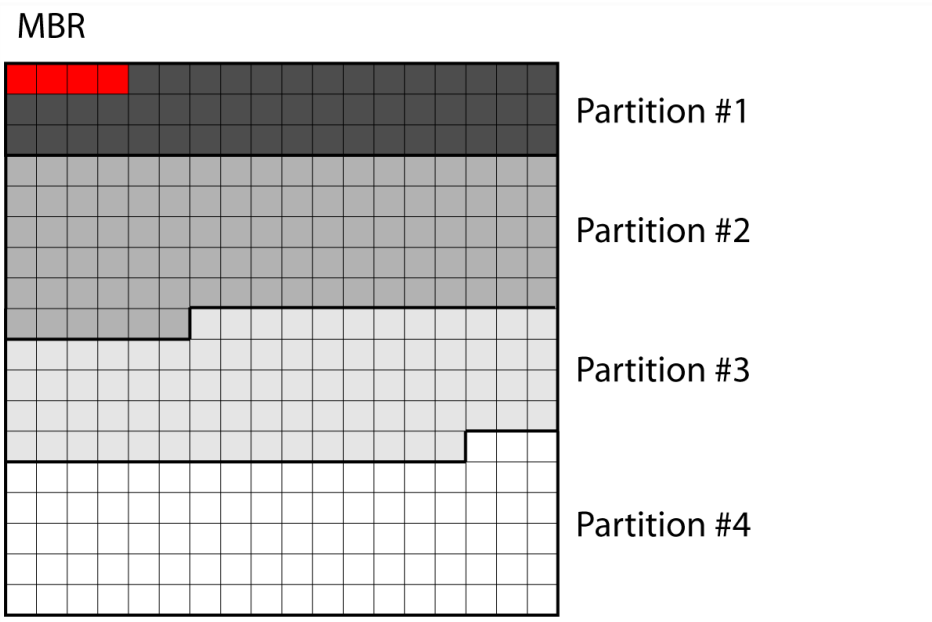
Partie #5

Plan

Le découpage de l'espace physique d'un disque dur

Les blocs mémoire : le découpage élémentaire

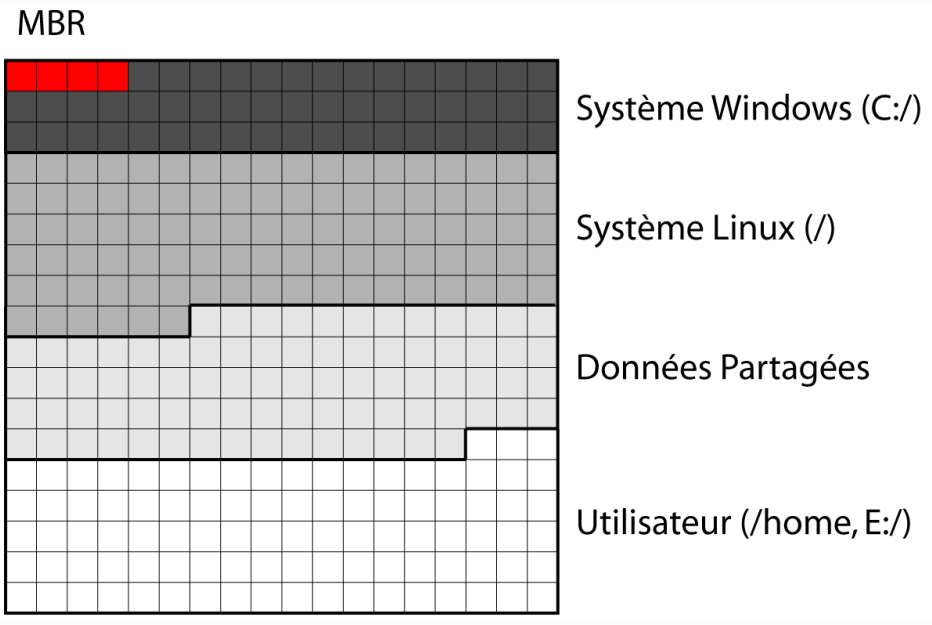
- Il s'agit d'une unité physique de stockage magnétique,
- Il est "découper" en "blocs" qui correspondent aux plus petites unités de stockage d'un système de fichier,
- La taille du bloc dépend du système de fichier.



Les partitions : un découpage intermédiaire

- L'espace mémoire disponible d'un disque dur peut être subdivisé en plusieurs parties,
- Chaque partie peut alors accueillir un système de fichier différent.
- Cela peut permettre de
  - faire cohabiter plusieurs systèmes d'exploitation sur un même disque dur,
  - définir des zones dédiées à certains types de fichiers.

La partition : un disque virtuel



La partition : un disque presque comme un autre

Une partition est définie par

- Un nom,
- Une zone physique sur le disque dur,
- Un système de fichier.

Les différents systèmes de fichier

Ils se caractérisent par :

- des tailles de blocs différents,
- des systèmes d'indexation différents permettant de retrouver l'adresse physique d'un fichier.

Les principaux systèmes de fichier

fat	Windows (Linux)	reiserfs	Linux
ntfs	Windows (Linux)	swap	Linux
ext	Linux		

Quelques partitions spéciales

MBR : Master Boot Record

- C'est la première partition lue lors de l'amorçage d'un ordinateur ; placée au début (physique du disque).
- Elle contient une base de registre indiquant les adresses physiques de début et de fin de chaque partitions,
- Elle permet d'indiquer où doit être lu le système d'exploitation à charger.

SWAP dans les systèmes linux

- Elle est dédiée au stockage de l'image mémoire des processus,
- Elle se comporte comme une extension de la mémoire vive sur le disque,
- Elle permet un accès plus rapide à des données stockées temporairement et mises à jour très fréquemment sur le disque dur.

Utiliser les partitions pour faciliter la gestion des données

- Une partition par système d'exploitation (susceptibles de changer)
- Une partition pour les programmes et applications,
- Une partition pour les données utilisateurs (conservées indépendamment des changement d'OS)
- Une partition pour les bases de données (accessible depuis plusieurs OS, plusieurs machines)

Les principaux répertoires et leur contenu

Une structure plus ou moins normalisée

- Les fichiers nécessaires au fonctionnement du système sont organisés en arborescence,
- Cette arborescence est commune à presque toutes les distribution linux,
- Cette organisation rationalisée facilite l'installation de nouveaux programmes qui savent où trouver les fichiers dont ils peuvent avoir besoin.

Une organisation qui permet un cloisonnement

- Les fichiers et les répertoires systèmes sont protégés par des restrictions de droits,
- De nombreux fichiers ne peuvent être modifiés par un utilisateur "normal",
- Seul l'utilisateur `root`, ou les utilisateur faisant partie du groupe `admin` peuvent avoir la permission de modifier certains fichiers.
- Il s'agit d'une protection. Pour réaliser une action susceptible d'affecter le comportement du système il faut montrer "patte blanche" et prendre conscience de ce que l'on fait. Entrer le mot de passe `root` doit être un signal d'alerte.

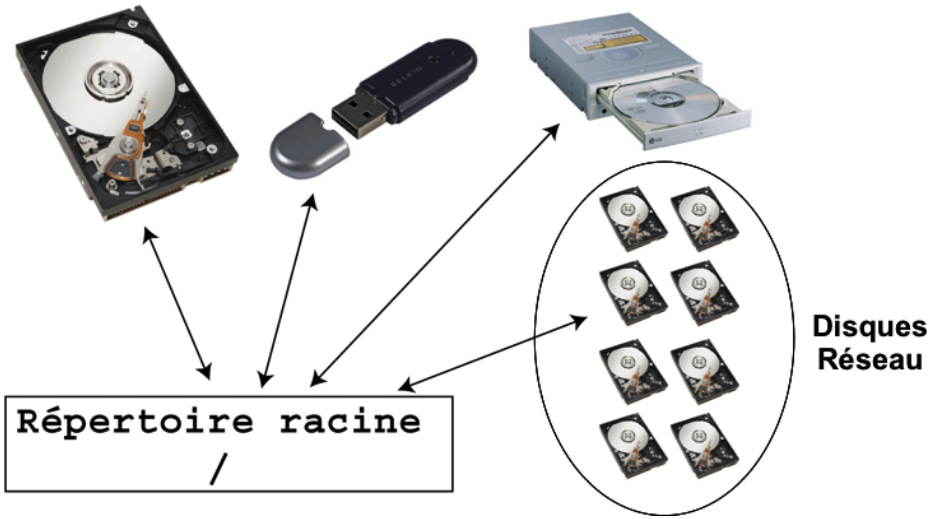
Les principaux répertoires et leur contenu

Répertoire	Contenu
/	Répertoire racine : Toutes les données accessibles par le système
/home	Les répertoires personnels des utilisateurs
/bin	Binaires exécutables des commandes de bases (cd, ls, mkdir, ?)
/lib	Librairies partagées et modules du noyeau
/usr	Ressources accessibles par les utilisateurs
/etc	Fichiers de configuration (profile, passwd,fstab...)
/tmp	Données temporaires
/dev	Fichiers spéciaux correspondants aux périphériques
/mnt	Points de montage des périphériques
/var	Fichiers de log ou fichiers changeant fréquemment
/root	Répertoire personnel de l'administrateur

Accéder aux données stockées sur un autre disque

Notion de disque

- Un disque est une unité de stockage physique ou virtuelle.
- Il peut s'agir d'un disque dur, d'une carte mémoire, d'une clef USB, d'une partition, ou d'un disque accessible *via* un réseau.



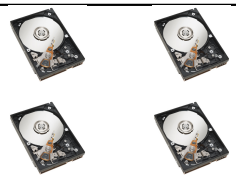
Accès aux données depuis le système de fichier

- Chaque périphérique de stockage à un système de fichier,
- Comme toute donnée accessible est définie par un chemin partant de la racine /, les données enregistrées sur des supports périphériques doivent avoir un chemin d'accès situé dans l'arborescence.

L'accès aux données se fait par des points de montage

Fichiers de périphériques

Lors de l'installation du système, l'os configure des fichiers spéciaux permettant de faire le lien avec des périphériques matériels connectés sur la carte mère.

	`/dev/hda1` `/dev/hdb1`	`/dev/hda1` `/dev/hdb1`
	`/dev/sda1`	
	`/dev/hdc`	
	server :/share	

Fonctionnement

- Il s'agit d'un fichier donnant un accès à un périphérique matériel,
- Pour accéder aux données il faut "monter" le périphérique au moyen de la commande `mount`.

mount

Syntaxe

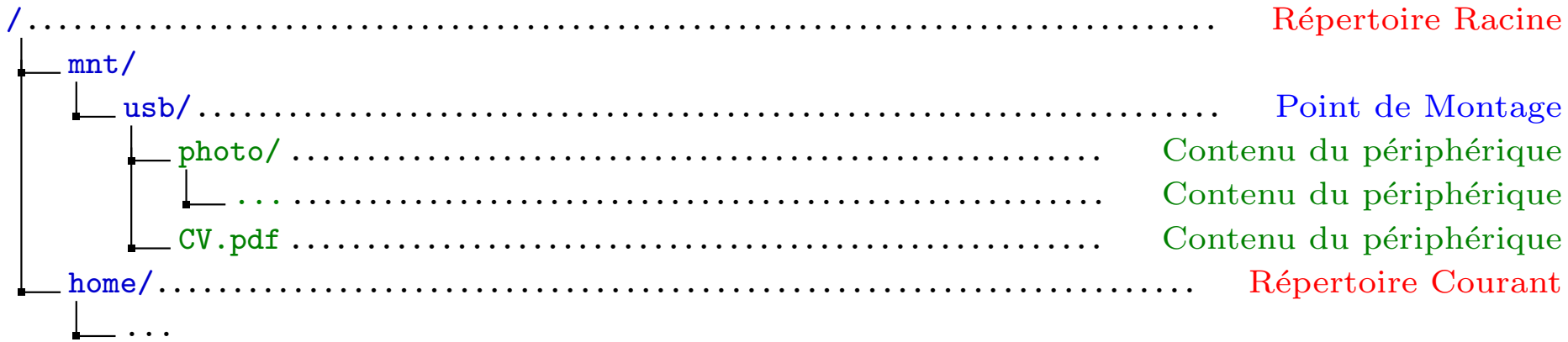
```
mount  périphérique point_de_montage
```

Description

- `périphérique` correspond soit à un fichier de périphérique (`/dev/xxx`), soit à l'adresse d'un disque (`nom_réseau_du_disque :répertoire_du_disque`).
- `point_de_montage` correspond à un nom de répertoire valide dans l'arborescence principale donnant accès au contenu de l'arborescence du périphérique.

Exemple d'utilisation:

```
[ login@localhost /home ] mount /dev/sda1 /mnt/usb
```



Liste des disques montés

Fichiers Systèmes

- /etc/fstab liste des disques à "monter" lors du démarrage du système. Il est modifiable par l'administrateur du système.
- /etc/mtab liste des disque "actuellement montés". Il est mis à jour automatiquement par le système dès qu'un disque est "monté" ou "démonté".

/etc/fstab

```
# /etc/fstab : static file system information.
#
# <file system> <mount point> <type> <options> <dump> <pass>
  proc          /proc             proc       defaults,noexec,nosuid          0    0
  /dev/sda1     /                 ext3       defaults,errors=remount-ro      0    1
  /dev/sda2     none              swap       sw                              0    0
  /dev/hda      /media/cdrom0     iso9660    ro,user,noauto                  0    0
  /dev/fd0      /media/floppy0    auto       rw,user,noauto                  0    0
```

/etc/mtab

```
/dev/sda1 / ext3 rw,errors=remount-ro 0 0
tmpfs /lib/init/rw tmpfs rw,nosuid,mode=0755 0 0
proc /proc proc rw,noexec,nosuid,nodev 0 0
server-xxx :/export4/vol04/santini /users/santini nfs
rw,noatime,rsize=131072,wsize=131072,vers=3,sloppy,addr=10.10.0.191 0 0
...
```

df

Syntaxe

```
df -h
```

Description

- Affiche les disques montés et leur capacité de mémoire.
- L'option `-h` (human readable) convertie l'affichage des tailles mémoires en unités conventionnelles (en nombre de blocs par défaut).

Exemple d'utilisation:

```
[ login@localhost ~ ] df -h
  Sys. de fichiers        Taille    Uti.    Disp.    Uti%    Monté sur
  /dev/sda1               56G       16G     37G      31%     /
  myserver :/home/sant    1,8T      1,6T    192G     90%     /users/santini
  ...                     ...       ...     ...      ...     ...

[ login@localhost ~ ] ■
```

du

Syntaxe

```
du -sh
```

Description

- Affiche l'espace mémoire utilisé par un fichier ou un répertoire.
- L'option `-h` (human readable) convertie l'affichage des tailles mémoires en unités conventionnelles (en nombre de blocs par défaut).
- L'option `-s` (sumurize) n'affiche pas le détail des fichiers et des sous-répertoires.

Exemple d'utilisation:

```
[ login@localhost ~ ] du -sh Documents/
5,2G Documents/

[ login@localhost ~ ] ■
```

Plan

Fichiers sources → Exécutable → Processus

Les sources : Une *"recette de cuisine"*

- Exprime un ensemble de tâches à réaliser pour accomplir le programme (le plat cuisiné).
- Utilise un langage de programmation.
- C'est un fichier texte.

L'exécutable

- Exprime les mêmes tâches dans un langage machine.
- Ce fichier ne fonctionne que sur des ordinateurs qui ont la même architecture.
- C'est un fichier binaire.

Les processus

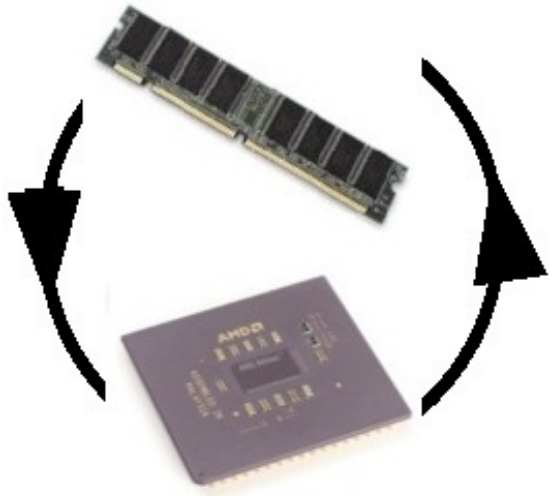
- L'évaluation des instructions machines engendre des processus.
- Ces processus sont exécutés par le matériel.
- Les instructions machine doivent donc être adaptées au matériel.

dessine.c

```
(...)
float r, x, y;
r=3.0;
x=0.0;
y=7.1;
cercle([0.,0.],r)
segment([0.,0.],[x,y])
```

dessine

```
10100101 11101001
10001001 00100101
00101010 00100010
01111011 10110101
01000010 00110011
00101101 11010100
(...)
```



Langages Compilés Vs Langages Interprétés

Caractéristiques des Langages Compilés

- L'ensemble du code source est compilé une seule fois avant l'exécution en instructions machine (contenues dans un fichier : exécutable).
- Le compilateur n'est pas nécessaire lors de l'exécution.
- Le compilateur est spécifique à la machine.
- L'exécutable (code compilé) est spécifique à la machine.

Inconvenients

- Il faut recompiler pour prendre en compte une modification du code.
- L'exécutable n'est pas portable sur d'autres machines.

Avantages

- Plus rapide (spécifique à la machine qui exécute les instructions).
- L 'ensemble des instructions sont regroupées dans un seul fichier.

Exemples de langages Compilés

- C, C++, ADA, Pascal, Fortran, Cobol,

Langages Compilés Vs Langages Interprétés

Caractéristiques des Langages Interprétes

- Les instructions du code source sont converties en instructions machine lors de l'exécution du programme
- L'interpréteur est nécessaire lors de l'exécution.
- L'interpréteur est spécifique à la machine,
- L'exécutable (le code source) n'est pas spécifique à la machine.

Inconvenients

- Moins rapide.
- Plusieurs fichiers (et librairies) servent à l'exécution.

Avantages

- Modifications du code source immédiatement prises en compte lors de la réexécution.
- Le code est portable sur d'autres machine

Exemples de langages Interprétés

- Java, Python, Bash, Lisp, PHP, Prolog, Perl

Lancer un programme/une commande

Cas général

- Pour exécuter un programme il suffit saisir sur la ligne de commande le chemin menant au fichier contenant les instructions,
- Si le fichier présente la permission `"X"` pour exécutable, les instructions qu'il contient sont exécutées.

Cas particulier : les commandes

- Une commande (`cd`, `ls`, `python`, `firefox`, ...) est un programme comme un autre,
- Les instructions qui doivent être évaluées sont écrites dans un fichier (`/bin/cd`, `/bin/ls`, `/usr/bin/python`, `/usr/share/bin/firefox`, ...),
- Poutant ...

Des chemins qui mènent nulle part

- les noms des commandes (`cd`, `ls`, `python`, `firefox` ...) sont toujours saisies comme des chemins relatifs (pas de `/bin/...` devant le nom du fichier), alors que le fichier de commande n'est pas dans le répertoire courant !...
- On donne donc un chemin vers un fichier qui n'existe pas ...

Chemins par défaut et variable d'environnement

Lorsqu'on donne une commande au terminal, on ne spécifie pas le chemin vers le fichier qui contient l'exécutable, on donne juste le nom du fichier...

```
[ login@localhost ~ ] ls
Mes_Documents/ Etoiles/ astronomie.txt cv.pdf
[ login@localhost ~ ] ■
```

...alors, comment le système trouve-t-il le fichier a exécuter correspondant à la commande ?...

Un mécanisme propre aux commandes

- Le premier mot tapé sur la ligne de commande est toujours interprétée comme le nom d'un fichier exécutable,
- Le système recherche donc dans une liste de répertoires contenant les exécutables si un fichier porte le nom de cette commande,
- Dès qu'il trouve dans ces répertoires un tel fichier, il l'exécute ...

Chemins par défaut et variable d'environnement

Les variables d'environnement

- Comme les variables d'un script, les variables d'environnement sont associées à une valeur,
- De telles variables sont définies par le système d'exploitation pour son fonctionnement, ce sont les variables d'environnement,
- ces variables peuvent être utilisées par les programmes.

La variable d'environnement `$PATH`

- Sa valeur est une liste de répertoires séparés par le signe `:`.

```
PATH=/bin:/usr/bin:/usr/local/bin:/usr/X11/bin
```

- Lors de chaque appel de commande, l'interpréteur parcourt cette liste dans l'ordre à la recherche d'un fichier portant le nom de la commande,
- Dès qu'il rencontre un tel fichier, il met fin à sa recherche et exécute le fichier.

Rôle de `$PATH`

- $\Rightarrow$ Il s'agit d'une liste de répertoires que l'interpréteur parcours automatiquement et séquentiellement (par défaut) si aucun chemin n'est donné pour trouver le fichier exécutable.

which

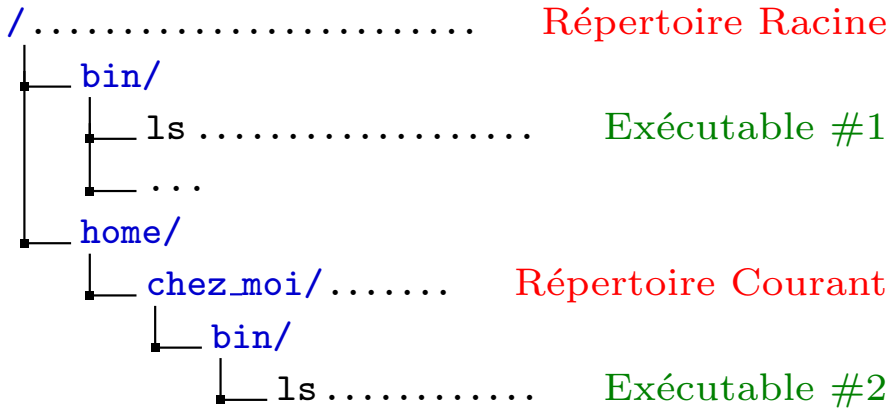
Syntaxe

```
which  nom_de_la_commande
```

Description

- Affiche le chemin du fichier correspondant à une commande.
- Parcours successivement les répertoires de la variable $PATH. Dès qu'il trouve un fichier correspondant au nom de la commande il renvoie son chemin.

Exemple d'utilisation:




```
[ login@localhost /home/chez_mo ] echo $PATH
/bin:/usr/bin:/usr/local/bin:/home/chez_moi/bin

[ login@localhost /home/chez_moi ] which ls
/bin/ls
```

Fichiers de configuration

Fichiers systèmes et utilisateurs

- Les variables d'environnement (et d'autres variables de configuration) sont définis dans divers fichiers.
- On distingue les fichiers système qui définissent des comportements pour tous les utilisateurs (stockés dans le répertoire `/etc/`) des fichiers de configuration propres à un utilisateur (stockés dans le répertoire personnel)

fichier	Propriétaire	Applicable à	Évalué lors
`/etc/profile`	`root`	Tous	Au début de chaque shell de login
`/home/chez_moi/.profile`	`utilisateur`	utilisateur	Au début de chaque shell de loginl
`/etc/bashrc`	`root`	Tous	Au début de chaque shell
`/home/chez_moi/.bashrc`	`utilisateur`	utilisateur	Au début de chaque shell

Configurer son environnement

- Chaque utilisateur peut redéfinir ses variables d'environnement,
- Pour cela il peut modifier le contenu des fichiers `.bashrc` et `.profile` dans son répertoire personnel,
- Ce sont des fichiers cachés (leur nom commence par un point : `.`). Pour voir si ils existent il faut utiliser l'option `-a` de la commende `ls`.

Fichiers de configuration

Contenu d'un fichier .bashrc

- Redéfinition des variables d'environnement,
- Définition des alias,
- Définition des fonctions,
- et de façon générale toutes les instructions que l'on souhaite évaluer lors de l'ouverture d'un nouveau shell.

.bashrc

```
# Mes aliases
alias ll='ls -l'
alias df='df -h'
alias rm='rm -i'
# Mes variables
PATH=$PATH:$HOME/bin
PYTHONPATH=$PYTHONPATH:$HOME/lib/python
```

Autres variables d'environnement

`$HOME` le chemin du répertoire personnel de l'utilisateur,

`$PWD` le chemin du répertoire courant.

alias

Syntaxe

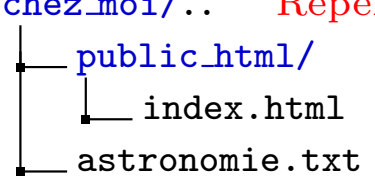
```
alias  nom_de_la_commande=expression
```

Description

- créet un alias entre un nom de commande et une expression.
- l'expression est donnée entre *quotes : 'expression ...'*

Exemple d'utilisation:

chez_moi/.. Répertoire Courant




```
[ login@localhost ~ ] ll
-bash : ll : command not found

[ login@localhost ~ ] alias ll='ls -l'


[ login@localhost ~ ] ls -l
  total 32
 drwxr-xr-x 2 santini ensinfo 4096 20 jui 15 :50     public_html
 -rw-r--r-- 1 santini ensinfo   25 20 jui 15 :49 telluriques.txt
```

dirname

Syntaxe

dirname *chemin*

Description

- Ne conserve que la partie répertoire d'un chemin d'accès.
- Il n'est pas nécessaire que le chemin existe dans l'arborescence. Le chemin est traité comme une chaîne de caractères.

Exemple d'utilisation:

```
[ login@localhost ~ ] dirname Documents
.
[ login@localhost ~ ] dirname Documents/cv.txt
Documents
[ login@localhost ~ ] dirname Documents/Photos/
Documents
[ login@localhost ~ ] dirname Documents/Photos/Soleil.jpg
Documents/Photos
[ login@localhost ~ ] ■
```

basename

Syntaxe

basename *chemin*

Description

- Élimine le chemin d'accès et le suffixe d'un nom de fichier.
- Il n'est pas nécessaire que le chemin existe dans l'arborescence. Le chemin est traité comme une chaîne de caractères.

Exemple d'utilisation:

```
[ login@localhost ~ ] basename curriculum.pdf
curriculum
[ login@localhost ~ ] basename Documents/cv.txt
cv
[ login@localhost ~ ] basename Documents/Photos/Soleil.jpg
Soleil
[ login@localhost ~ ] ■
```

Plan

Les calculs arithmétiques

Bash un langage orienté sur le traitement des chaînes de caractères

Même si ce langage n'est pas fait pour effectuer des opérations de calcul arithmétique il propose des fonctionnalités de base permettant d'effectuer des calculs simples tels que les additions, soustractions, multiplications et divisions.

Syntaxe

`$(( expression_arithmétique ))`

Exemples

```
[ login@localhost ~ ] total=$(( 5 + 3 ))

[ login@localhost ~ ] echo $total
8
[ login@localhost ~ ] echo $(( 5 - 3 ))
2
[ login@localhost ~ ] echo $(( 5 * 3 ))
15
[ login@localhost ~ ] echo $(( 5 / 3 ))
1
```

for

for Boucle itérative

- permet de répéter l'évaluation d'une ou plusieurs instructions,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur atteint une certaine valeurs.

Syntaxe #1

```
for (( init ; test ; incr )) ; do
    expr_1
    expr_2
    ...
done
```

Ici, la condition d'arrêt est sur la valeur numérique de l'itérateur.

Exemple #1

test_for_loop_1.bash

```
#!/bin/bash

echo "test #1"
for (( i = 0 ; i < 3 ; i++ )) ;do
    echo '$i = '$i
done
```

```
[ login@localhost ~ ]
./test_for_loop_1.bash
test #1
$i = 0
$i = 1
$i = 2
```

for

for Boucle itérative

- permet de répéter l'évaluation d'une instruction,
- à chaque tour de boucle une variable appelée itérateur change de valeur,
- la sortie de boucle s'effectue lorsque l'itérateur a parcouru toute la liste.

Syntaxe #2

```
for var in val_1 val_2 ... ; do
    expr_1
    expr_2
    ...
done
```

Ici, la boucle s'arrête lorsque toute la liste des valeurs a été parcourue.

Exemple #2

test_for_loop_2.bash

```
#!/bin/bash

echo "test #2"
for i in hello la terre ;do
    echo '$i = '$i
done
```

```
[ login@localhost ~ ]
./test_for_loop_2.bash
test #2
$i = hello
$i = la
$i = terre
```

if

Branchements conditionnels

- Le `if` permet de mettre en place des alternatives.
- Un test (dont le résultat est Vrai ou Faux) permet de conditionner les expressions qui seront évaluées.

Syntaxe #1

```
if test
then
        expr_1
        expr_2
        ...
fi
```

Comportement

- Ici, les expressions ne sont évaluées que si le test renvoie la valeur Vrai.
- Aucune des expressions ne sont évaluées si le test renvoie la valeur Faux.

if

Syntaxe #2

```
if test
then
        expr_1
else
        expr_2
fi
```

Comportement

- Si le test renvoie la valeur Vrai l'expression `expr_1` est évaluée, et
- sinon le test renvoie la valeur Faux c'est l'expression `expr_2` qui est évaluée.

Syntaxe #3

```
if test_1
then
        expr_1
elif test_2
        expr_2
elif test_3
        expr_3
else
        expr_4
fi
```

Comportement

- Si `test_1` renvoie la valeur Vrai l'expression `expr_1` est évaluée,
- si `test_2` renvoie la valeur Vrai l'expression `expr_2` est évaluée,
- si `test_3` renvoie la valeur Vrai l'expression `expr_3` est évaluée, et
- si aucun des tests ne renvoie la valeur Vrai alors c'est l'expression `expr_4` qui est évaluée.

Les tests

Les tests peuvent prendre plusieurs formes

Il peuvent porter sur :

- l'arborescence (présence, absence, permission sur les répertoires et fichiers),
- les chaînes de caractères,
- les valeurs numériques.

Tests de l'arborescence

Syntaxe	Valeur
[[-d fichier]]	Vrai si fichier est un nom de répertoire valide (si il existe).
[[-f fichier]]	Vrai si fichier est un nom de fichier valide (si il existe).
[[-r fichier]]	Vrai si il y a le droit de lecture sur le fichier.
[[-w fichier]]	Vrai si il y a le droit d'écriture sur le fichier.
[[-x fichier]]	Vrai si il y a le droit d'exécution sur le fichier.

Les tests

Tests sur les chaînes de caractères

Syntaxe	Valeur
[[chaine_1 = chaine_2]]	Vrai si les 2 chaînes sont identiques.
[[chaine_1 != chaine_2]]	Vrai si les 2 chaînes sont différentes.
[[-n chaine]]	Vrai si la chaîne est non vide.
[[-z chaine]]	Vrai si la chaîne est vide.

Tests sur les valeurs numériques

Syntaxe	Valeur
[[nb_1 -eq nb_2]]	Vrai si nb_1 $=$ nb_2 (eq pour equal).
[[nb_1 -ne nb_2]]	Vrai si nb_1 $\neq$ nb_2 (ne pour not equal).
[[nb_1 -gt nb_2]]	Vrai si nb_1 $>$ nb_2 (gt pour greater than).
[[nb_1 -ge nb_2]]	Vrai si nb_1 $\geq$ nb_2 (ge pour greater or equal).
[[[nb_1 -lt nb_2]]	Vrai si nb_1 $<$ nb_2 (ge pour lower than).
[[[nb_1 -le nb_2]]	Vrai si nb_1 $\leq$ nb_2 (ge pour lower or equal).

Les tests

Opérateurs booléens

Syntaxe	Valeur
! [[test]]	NOT : Vrai si le test renvoie Faux (négation).
[[test_1]] || [[test_2]]	OU logique.
[[test_1]] && [[test_2]]	ET logique.

Tables de vérité

ET (&&)	**Vrai**	**Faux**
Vrai	Vrai	Faux
Faux	Faux	Faux

OU (||)	**Vrai**	**Faux**
Vrai	Vrai	Vrai
Faux	Vrai	Faux

NOT (!)	**Vrai**	**Faux**
	Faux	Vra