

UNIX ...

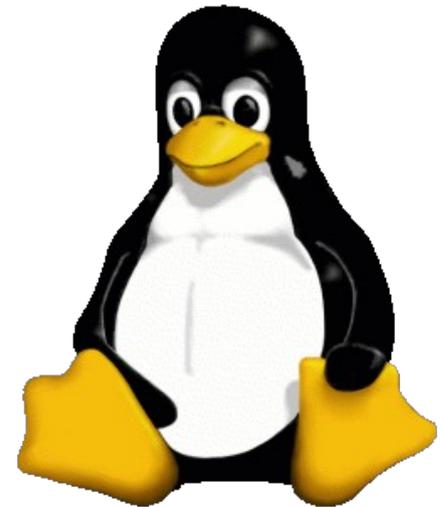
et surtout GNU-Linux !



(Partie 1)

Benoit Da Mota

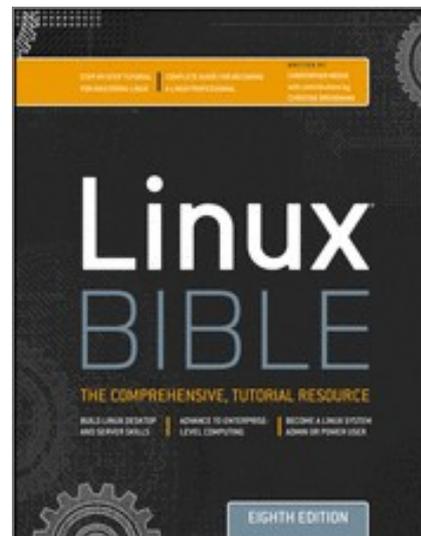
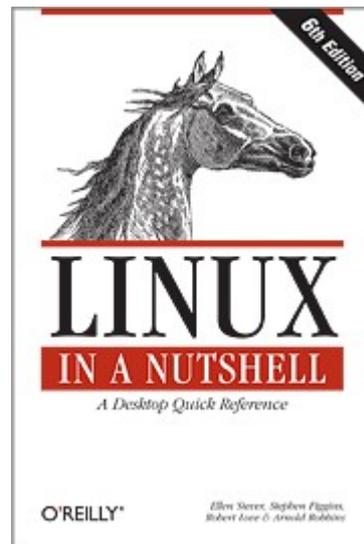
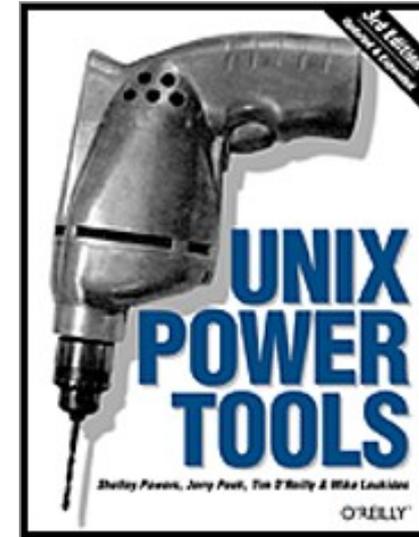
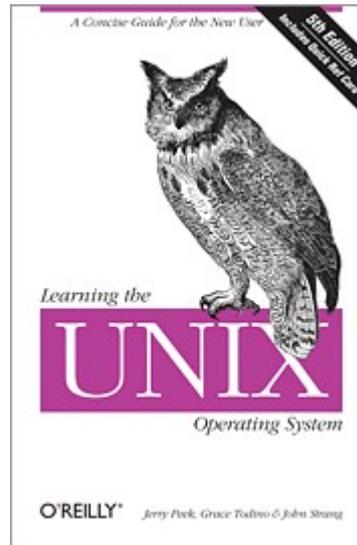
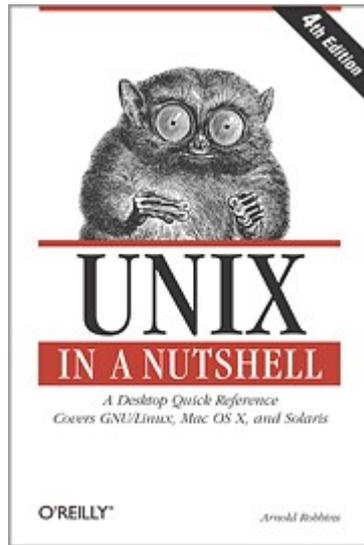
benoit.damota@univ-angers.fr
Bureau H204



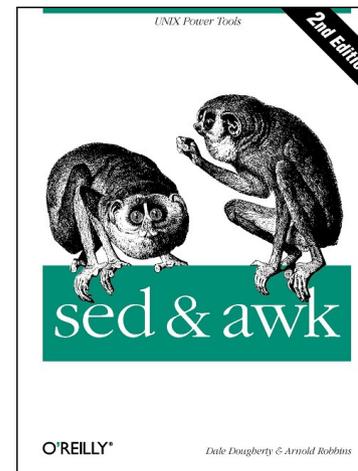
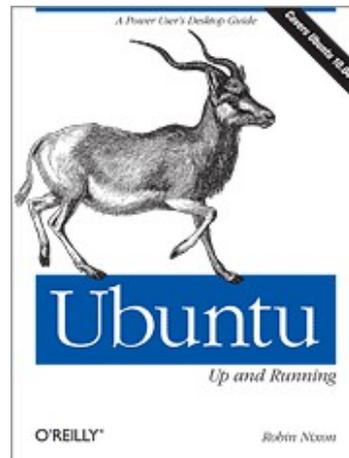
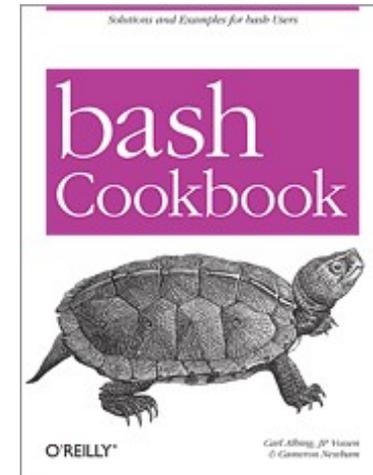
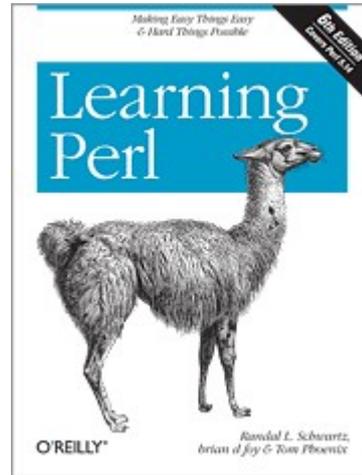
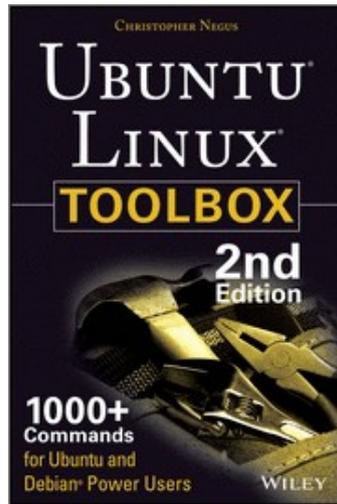
Sources et Références

- Anciens cours d'UNIX de l'Université d'Angers
- Introduction Système par Guillaume Santini (Paris 13)
- Unix Tools 2006, Université de New York

Livres de référence ?



Livres de référence ?



Plan du cours

- Introduction
- Système de fichiers et navigation
- Propriétés et droits sur les fichiers
- Exécutables et processus
- Archivage et compression
- Edition et manipulation de fichiers
- E/S et tubes
- Exécution et configuration `.bashrc`
- Scripts Bash

SECTION 1

Introduction, Historique, Vue
d'ensemble d'une distribution Linux

L'HORIZON MATÉRIEL

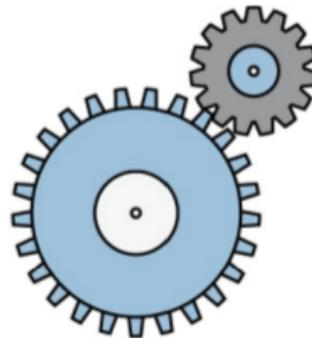
INTERACTION AVEC LE MATÉRIEL

- 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.

LE SYSTÈME D'EXPLOITATION

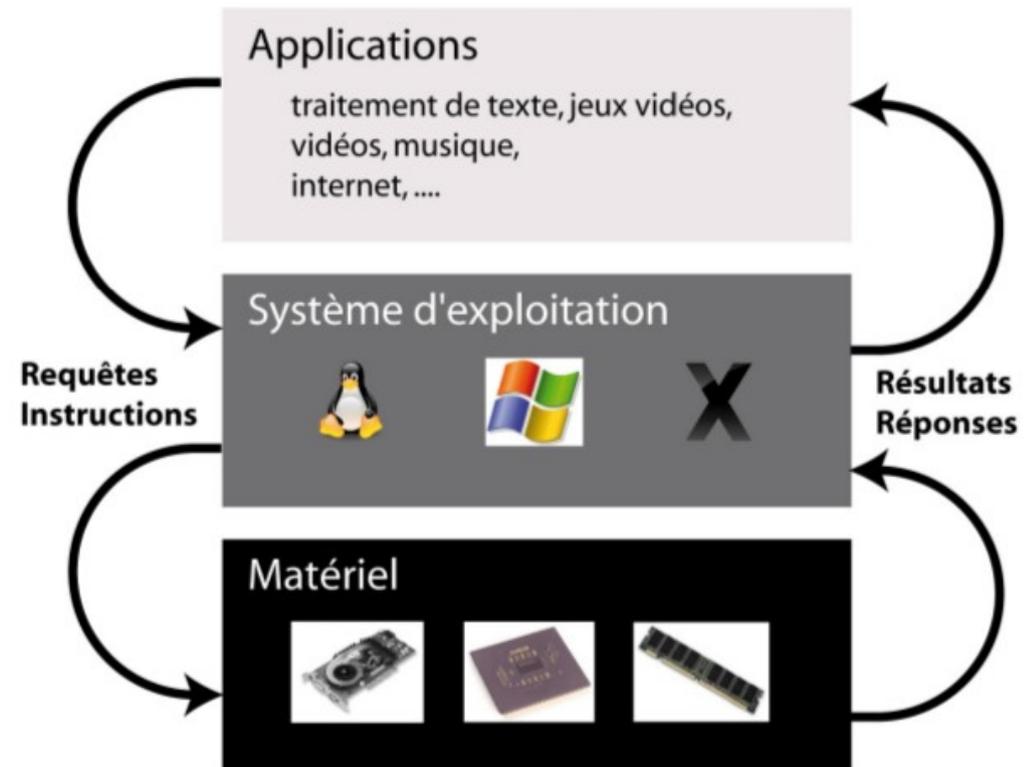
Le système d'exploitation permet de développer des programmes sans tenir compte de la complexité physique de la machine. Le programme utilise des fonctionnalités standardisées d'accès aux ressources matériel.

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, ...)
- ...

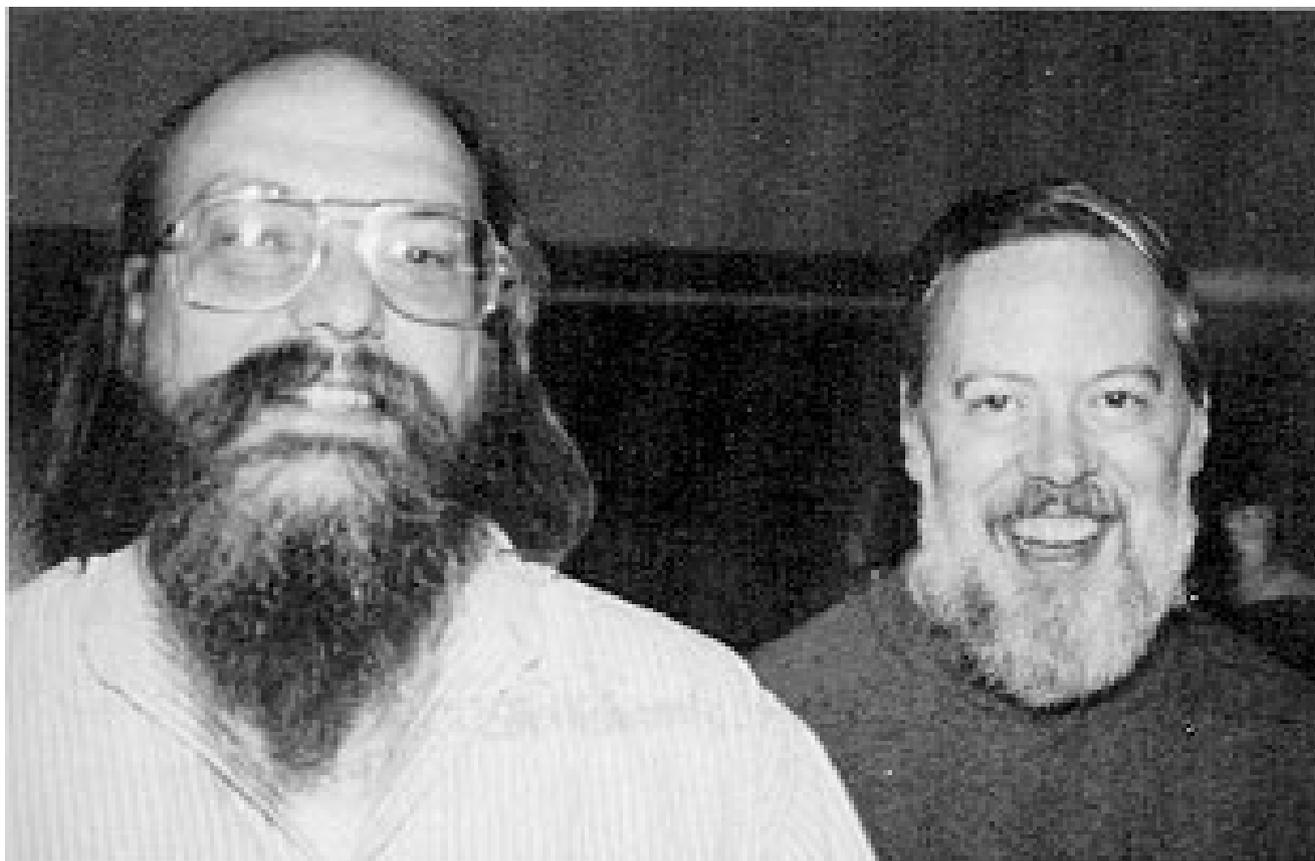
CÔTÉ UTILISATEUR

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



L'Histoire d'UNIX

- Première version créée dans les laboratoires Bell – 1969
- AT&T met sous licence le code source à un prix bas
 - Contrôle étroit de la marque déposée UNIX
 - Les détenteurs de licence doivent créer un nouveau nom pour leur système d'exploitation
 - Apparition de nombreuses “variétés” d'UNIX



Ken Thompson
1943-

Dennis Ritchie
1941-2011

Au commencement...

- UNICS: 1969 – PDP-7 minicomputer
- Réécriture pour PDP-11.
- V1: 1971
- V3: 1973 (pipes, langage C)
- **V6**: 1976 (réécrit en C, base pour BSD)
- V7: 1979 (licence, portabilité)

Juillet 1969 : premier humain sur la lune



PDP-11



Philosophie UNIX

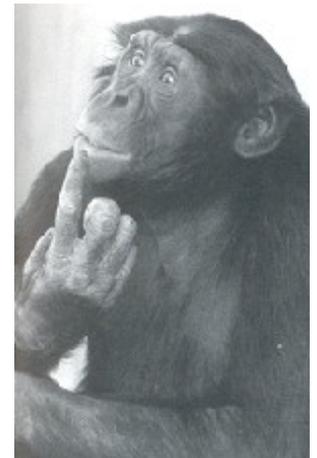
- « Ce qui est petit est beau »
 - Facile à comprendre
 - Facile à maintenir
 - Plus efficace
 - Meilleur pour être réutilisé
- « Chaque programme fait bien une seule chose »
 - On obtient des fonctionnalités plus complexes en combinant des programmes
 - Chaque programme est un **filtre**



Philosophie UNIX

...suite

- « Portabilité plutôt qu'efficacité »
 - Les implémentations les plus efficaces sont rarement portables
 - La portabilité est plus intéressante pour du matériel qui évolue rapidement
- « Utiliser des fichiers plats ASCII »
 - Un format simple et commun (l'ancêtre de XML)
 - Un exemple de « portabilité plutôt qu'efficacité »
- « Code réutilisable »
 - Un bon programmeur écrit du bon code ;
un excellent programmeur **emprunte ou adapte** du bon code



Philosophie UNIX

...suite

- « Scripter augmente l'efficacité et la portabilité »

```
print $(who | awk '{print $1}' | sort | uniq) | sed 's/ /,/g'
```

Liste les logins des utilisateurs avec une simple ligne de commande

- Construit rapidement des prototypes (langages interprétés de haut niveau)

who	755
awk	3,412
sort	2,614
uniq	302
sed	2,093

9176 lignes

Philosophie UNIX

...suite

- « éviter les interfaces captives »
 - L'utilisateur n'est pas toujours un humain
 - C'est joli, mais le code est grand et moche
 - Ça ne passe pas à l'échelle
- « le silence est d'or »
 - rapporter uniquement si quelque chose se passe mal



- « penser hiérarchiquement »

UN PEU D'HISTOIRE

GNU-LINUX « GNU's Not UNIX »

- 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.

AUJOURD'HUI

- C'est 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-compilés, de nombreux outils d'administration et de services (protocoles, ...)



Richard Stallman



Linus Torvalds

On parle de «distribution Linux»

- Autopsie d'une distribution Linux:
 - Version personnalisée d'un noyau Linux
 - Utilitaires et applications
 - Logiciel d'installation et de configuration
 - Assistance disponible via des entreprises commerciales comme Red Hat, SUSE, Canonical ...
 - Une communauté

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 COPYLEFT☺

Elles permettent de : Distribué en Copyleft☺, 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 ont accès.
- Libre est gratuit : Le coût est supporté par une communauté (utilisateurs, subventions publiques, subventions ou sociétés privées, ...).

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 a 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 bibliothèques dans les répertoires système, ...)

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 réflexe.
- 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 ...

Démo : man

Connexions Locales

- Connexion en mode texte sur console virtuelle
- Connexion graphique
- De multiples connexions non-GUI sont possibles par l'utilisation de consoles virtuelles
- Il existe 6 consoles virtuelles disponibles par défaut
- Disponibles par **CTRL - ALT - F [1 - 6]**
- Si X est exécuté, disponible par **CTRL - ALT - F7**

Environnements Graphiques Linux

- GNOME – environnement de bureau basé sur le toolkit GTK+
- KDE - environnement basé sur le toolkit Qt
- Awesome – très léger et configurable
- Et de nombreux autres ...

http://fr.wikipedia.org/wiki/Gestionnaire_de_fen%C3%AAtres

Changer votre Mot de Passe

- Les mots de passe devraient être changés après la première connexion
- Depuis un terminal, utilisez `passwd`

SECTION 2

Navigation dans le Système de
Fichiers en Ligne de Commande

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

<code>nom.extension</code>
<code>prefix.suffix</code>
<code>description.format</code>

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

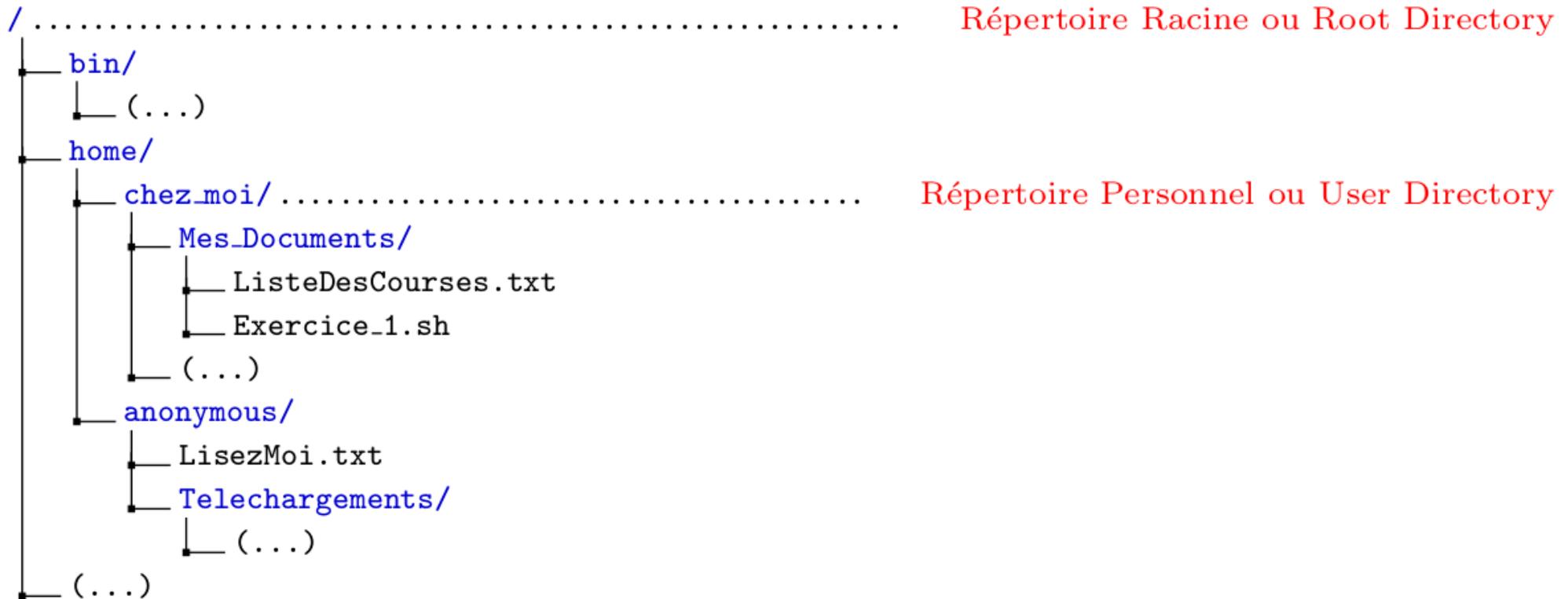
Enigmatique	Informatif
e3.c	teste_boucle_for.c
New.pdf	2011_IntroSys_cours_1.pdf
toto.sh	test_boucle_for.sh

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.

L'ORGANISATION ARBORESCENTE

EXEMPLE D'ARBORESCENCE LINUX



LES RÉPERTOIRES IMPORTANTS

- Le Répertoire 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 ...).

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 distributions 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 système sont protégés par des restrictions de droits,
- De nombreux fichiers ne peuvent être modifiés par un utilisateur "normal",
- Seul l'utilisateur `root`, ou les utilisateurs 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 noyau
/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

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 **E**ssai.txt et **e**ssai.txt peuvent donc être dans le même répertoire.

EXEMPLES DE CHEMINS ABSOLUS



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

```
home/..... ../..
├── chez_moi/..... ../
│   ├── Etoiles/..... Répertoire Courant ./
│   │   ├── SOLEIL.jpg..... SOLEIL.jpg ou ./SOLEIL.jpg
│   │   ├── Antares.jpg..... Antares.jpg ou ./Antares.jpg
│   │   └── Systeme_Solaire/..... ../Systeme_Solaire/
│   └── terre.gif..... ../Systeme_Solaire/terre.gif
```

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 exécution),
 - 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.

Les Sept Types de Fichiers de Base

- - fichier régulier
- d répertoire
- l lien symbolique
- b fichier spécial de type bloc
- c fichier spécial de type caractère
- p tube nommé
- s socket

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

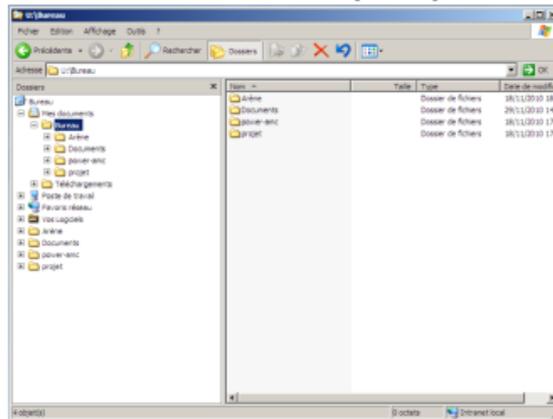
SOURCE c'est un fichier ou un répertoire utilisé en entrée d'une commande,

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

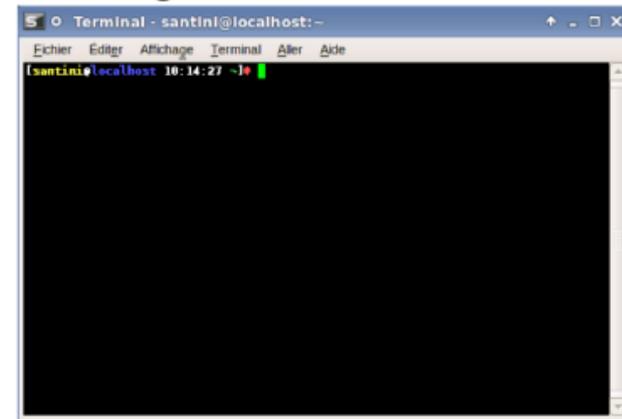
MANIPULATION DE L'ARBORESCENCE EN LIGNE DE COMMANDE

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 : print **w**orking **d**irectory
ls : **l**ist
cd : **c**hange **d**irectory
mkdir : **m**ake **d**irectory
rmdir : **r**emove **d**irectory
rm : **r**emove
cp : **c**opy
mv : **m**ove
df : **d**isk **f**ree
du : **d**isk **u**sed

MÉTACARACTÈRE ET CHEMINS CIBLÉS

LE CARACTÈRE *

- Le caractère * est utilisé comme un *joker* 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`).

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

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

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`).

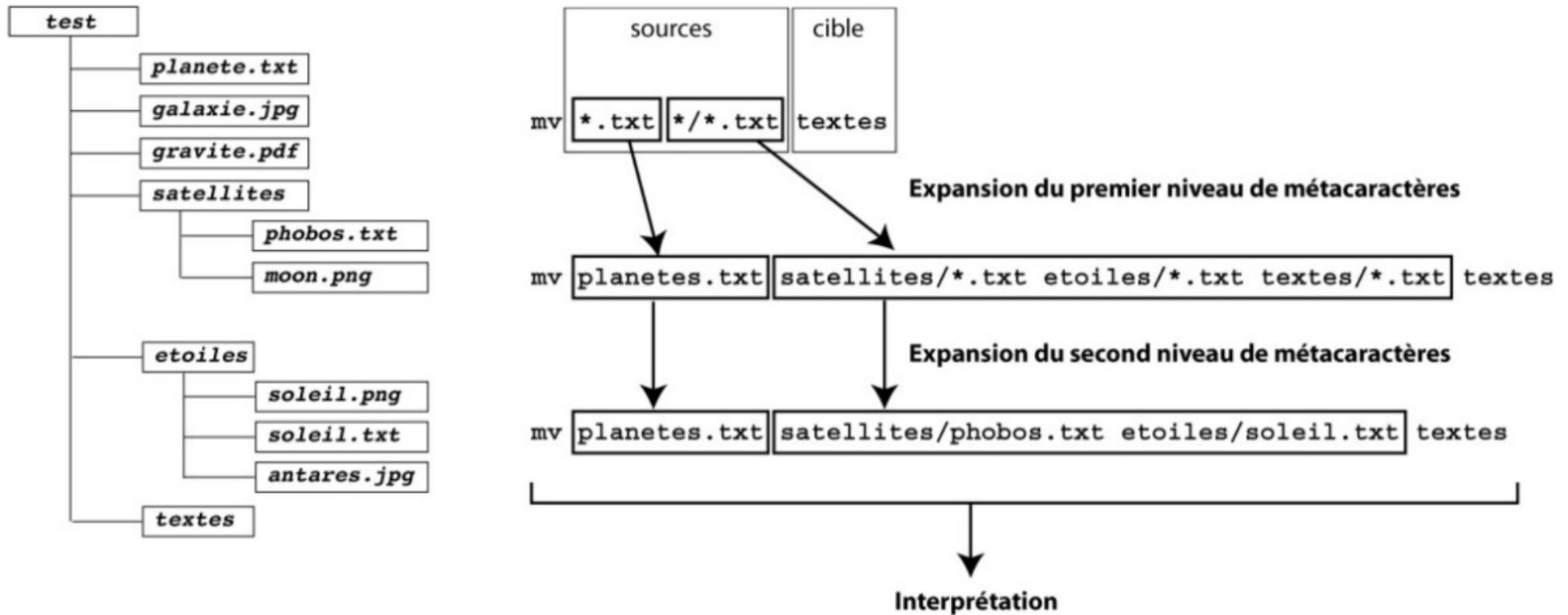
```
chez_moi/..... Répertoire Courant
├── aldebaran.jpg
├── alphacentauri.gif ..... Fichier ciblé
├── etacentauri.jpg ..... Fichier ciblé
└── Images/ ..... Répertoire Final
```

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

MÉTACARACTÈRE ET CHEMINS CIBLÉS

EXEMPLE PLUS COMPLEXE ET DÉTAILS DE L'INTERPRÉTATION

- Le caractère * est développé lors de l'interprétation.



Raccourcis de Ligne de Commande

Expansion de Fichiers

- Expansion de caractères génériques :
 - * - correspond à zéro ou plusieurs caractères
 - ? - correspond à tout caractère
 - [a - z] - correspond à un ensemble de caractères
 - [^a - z] - correspond à tout sauf l'ensemble donné

Raccourcis de Ligne de Commande

La Touche <Tab>

- Saisissez <Tab> pour compléter les lignes de commande :
 - Pour une commande, elle complétera le nom de la commande
 - Pour un argument, elle complétera un nom de fichier

- Exemples :

```
$ xboa<Tab>
```

```
$ xboard
```

```
$ ls monf<Tab>
```

```
$ ls monfichier.txt
```

Raccourcis de Ligne de Commande `history`

- `bash` stocke l'historique des commandes que vous avez entrées, ce qui peut être utilisé pour répéter des commandes
- Utiliser la commande `history` pour afficher la liste des commandes "retenues"

Expansion de Ligne de Commande Variable et Chaîne

- Paramètre/Variable:

- Remplace la valeur d'une variable dans la ligne de commande

```
$ cd $HOME/public_html
```

- Accolades :

- Une chaîne de texte est créée pour chaque modèle à l'intérieur des accolades, sans se soucier de l'existence de fichiers

```
$ rm hello.{c,o}
```

Protéger contre l'Expansion L'oblique Inverse

- L'oblique inverse est le caractère d'échappement et rend le caractère suivant littéral

```
$ echo Your cost: \$5.00
```

- Utilisée en tant que dernier caractère de ligne pour "continuer la commande sur la ligne suivante"

```
$ echo "This long sample string will be echoed \  
> back as one long line"
```

```
This very long sample string will be echoed back as  
one long line
```

Protéger contre l'Expansion

Les Guillemets

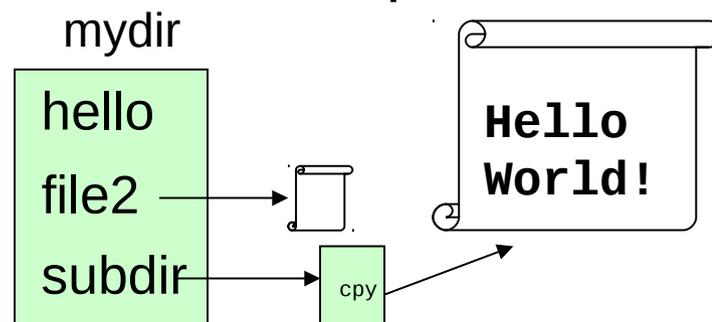
- Les guillemets empêchent l'expansion
 - Les guillemets simples (') empêchent toute expansion
 - Les guillemets doubles (") empêchent toute expansion, sauf :
 - \$ (dollar) incluant \$() et \$[], expansion de variables
 - ` (guillemet inverse), substitution de commandes
 - \ (oblique inverse), limite sur un seul caractère
 - ! (point d'exclamation), substitution d'historique

Astuces pour l'Historique

- Utiliser les flèches de direction pour faire défiler les commandes précédentes
- Taper `<Ctrl - r>` pour rechercher une commande dans l'historique des commandes.
 - `(reverse-i-search) ` `` :
- Pour rappeler le dernier argument de la commande précédente :
 - `<Échap>` . (la touche Échap suivie d'un point)
 - `<Alt>` . (appuyez sur la touche Alt et sur le point)

Liens

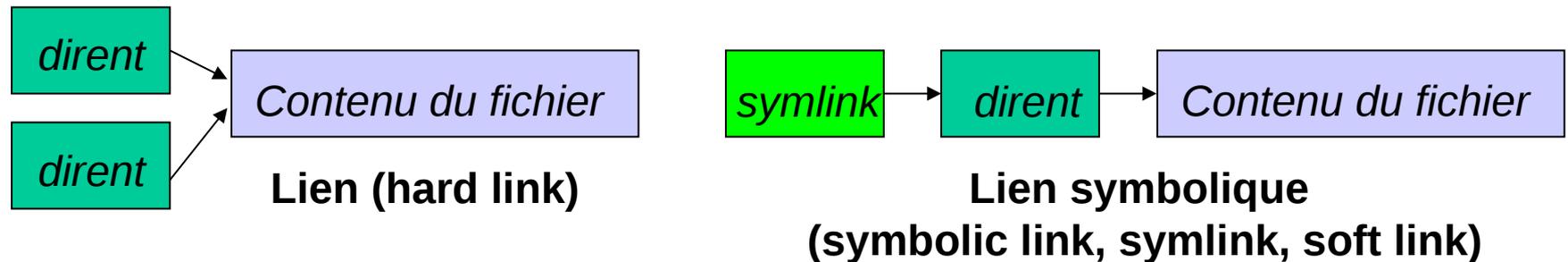
- Les répertoires sont des listes de fichiers et de répertoires
 - Chaque entrée d'un répertoire (directory entry ou dirent) est en lien avec un fichier sur le disque



- 2 entrées différentes peuvent être liées au même fichier (même répertoire ou à travers différents répertoires)
- Déplacer un fichier ne déplace pas (forcément) des données :
 - Création d'un lien pour la destination cible
 - Effacement de l'ancien lien
- commande `ln` : **link**

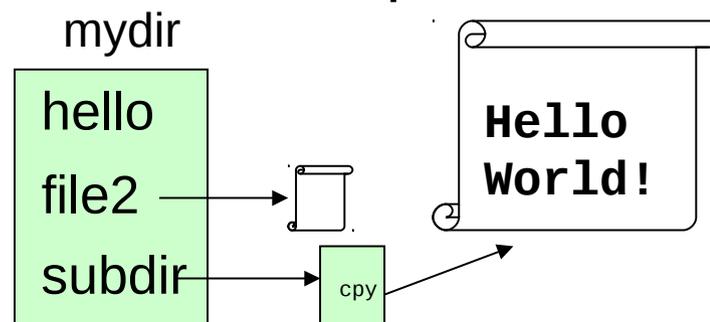
Liens symboliques

- **Les liens symboliques** sont différents de liens réguliers (**hard links**). Ils sont créés avec `ln -s`
- On peut les voir comme une entrée de répertoire qui **pointe** sur le nom d'un autre fichier
- Ça ne change pas le comptage du nombre de liens pour le fichier
 - Quand l'original est effacé, le lien symbolique reste
- Pourquoi des liens symboliques ?
 - Les *hard links* ne fonctionnent pas à travers les système de fichiers
 - Les *hard links* ne fonctionnent pas pour les répertoires



Liens

- Les répertoires sont des listes de fichiers et de répertoires
 - Chaque entrée d'un répertoire (directory entry ou dirent) est en lien avec un fichier sur le disque



- 2 entrées différentes peuvent être liées au même fichier (même répertoire ou à travers différents répertoires)
- Déplacer un fichier ne déplace pas (forcément) des données :
 - Création d'un lien pour la destination cible
 - Effacement de l'ancien lien
- commande `ln` : **link**

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'entraîne pas la destruction du fichier original.
- Plusieurs liens symboliques peuvent pointer le même fichier ou le même répertoire.

ln

SYNTAXE

```
ln -s source cible
```

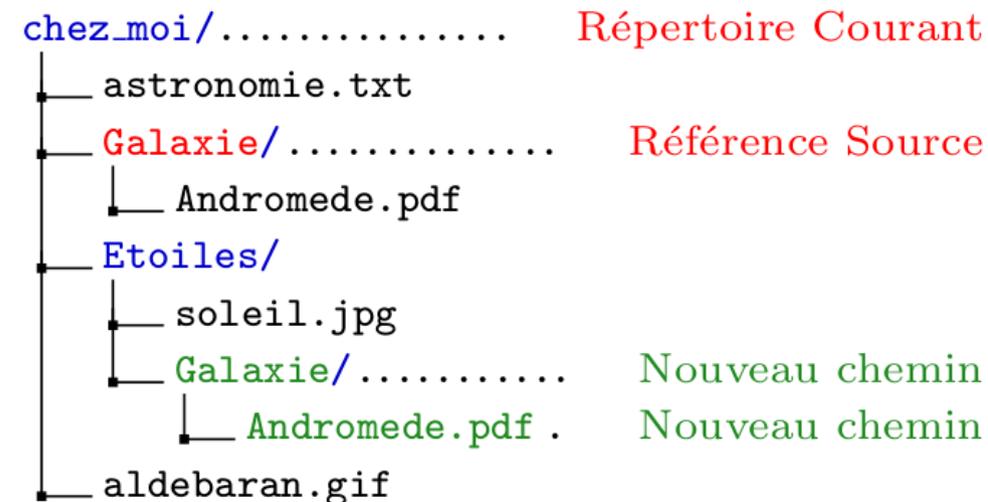
DESCRIPTION

- 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.



Démo 1n

Outils de recherche de fichiers

find

- **find** *pathlist expression*
- **find** est récursif à partir de *pathlist* et applique *expression* à chaque fichier
- *Une expression* peut être:
 - **-name** *pattern*
 - *vrai* si le nom de fichier correspond au *pattern*. Le *pattern* peut inclure *, est doit posséder des guillemets pour protéger contre l'interprétation
 - Ex: **find / -name '*.c'**

find

- **-type** *ch*
 - *c*=character, *b*=block, *f*=file (regular), etc...
Ex: **find /home -type f**
- **-user** *userid / username*
- **-group** *groupid / groupname*
- **-size** *size*
- **-perm** *[+ -]mode*

- *Et beaucoup d'autres ...*

find : opérateurs logiques

- *!* *expression* négation de expression
- *op1 -a op2* *op1* AND *op2*
- *op1 -o op2* *op1* OR *op2*
- () pour grouper des expressions

find: actions

- **-print**
 - Affichage
- **-ls**
 - Affichage type ls
- **-exec *cmd***
 - Exécute *cmd*, avec *cmd* qui doit se terminer par un point virgule protégé (`\;` or `' ; '`).
 - Si *cmd* contient `{}` alors il sera remplacé par le nom du fichier courant trouvé.
 - **exec** exécute *cmd* une fois par fichier trouvé
 - Exemple:
 - `find -name "*.o" -exec rm "{}" ";"`

Exemples de find

- Fichiers commençant par f dans *home directory*
- `find ~ -name 'f*' -print`
- Modifiés il y a moins de 1 jour
- `find ~ -mtime 1 -print`
- Plus grands que 10K
- `find ~ -size 10k -print`
- Compte les mots dans les fichiers de *home directory*
- `find ~ -exec wc -w {} \; -print`
- Efface les fichiers core depuis la racine
- `find / -name core -exec rm {} \;`

Démo find

SECTION 3

Propriétés et droits
sur les fichiers

Utilisateurs

- À tout utilisateur du système est attribué un identifiant unique appelé User ID (le *uid*)
- Les noms des utilisateurs et les uids sont sauvegardés dans `/etc/passwd`
- Les utilisateurs sont attribués à un répertoire personnel et à un programme qui est exécuté lorsqu'ils se connectent (normalement un shell)
- Les utilisateurs ne peuvent ni lire, ni écrire, ni exécuter les fichiers des uns des autres sans autorisation

Groupes

- Les utilisateurs sont affectés à des groupes avec un identifiant unique appelé gid
- Les gids sont sauvegardés dans `/etc/group`
- Chaque utilisateur reçoit son propre groupe privé
 - Ils peuvent également être ajoutés à d'autres groupes pour avoir des accès supplémentaires
- Tous les utilisateurs d'un groupe peuvent partager les fichiers qui appartiennent au groupe

L'Utilisateur `root`

- L'utilisateur `root` : un compte administratif spécial
 - `root` est parfois appelé le *super-utilisateur*
 - `root` a un contrôle total sur le système
 - Une capacité illimitée d'endommager le système !
- Vous ne devriez pas vous connecter en tant que `root` sans une très bonne raison
 - Le potentiel des utilisateurs normaux («sans privilèges») de faire des dégâts est limité

Le Modèle de Sécurité Linux

- Utilisateurs et groupes sont utilisés pour contrôler l'accès aux fichiers et aux ressources
- Les utilisateurs se connectent au système en donnant leur nom d'utilisateur et leur mot de passe
- Chaque fichier sur le système appartient à un utilisateur et est associé à un groupe
- Chaque processus a un propriétaire et une affiliation de groupe et ne peut accéder qu'aux ressources auxquelles son propriétaire ou groupe a également accès

Démo :

- UID / GID
- top et ps
- ls -l
 - type du fichier
 - propriétaire et groupe
 - droit (rwx)
 - 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		

Manipuler les attributs des fichiers

- **chmod** change les permissions
- **chown** change le propriétaire
- **chgrp** change le groupe
- **umask** user mask
 - Mode de création des fichiers

Seul le propriétaire et le super-utilisateur peuvent changer les attributs

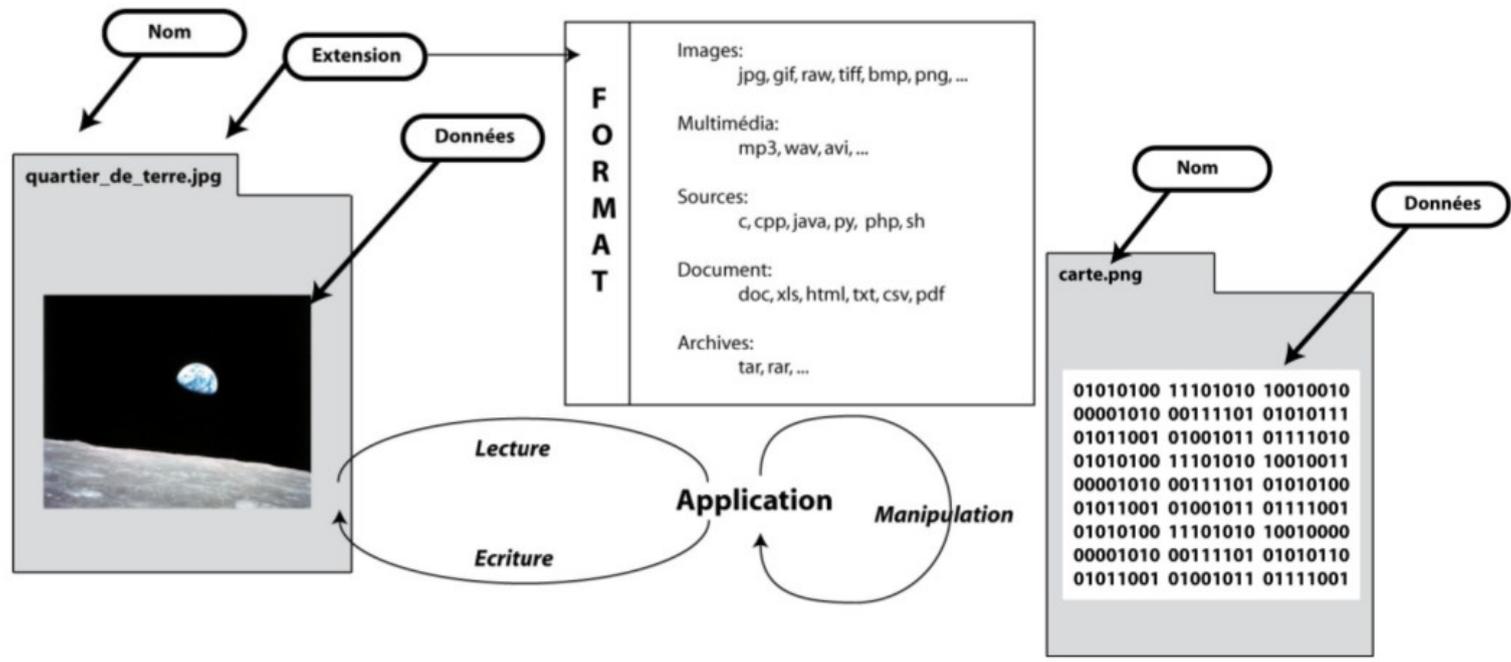
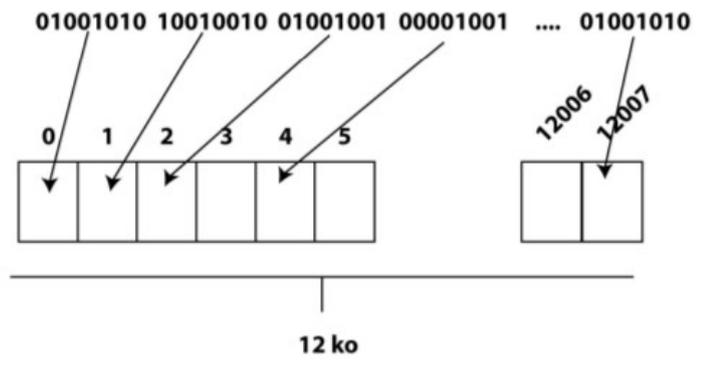
SECTION 4

Fichiers exécutables
et processus

FICHER BINAIRE ET FICHER 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.

FICHER BINAIRE ET FICHER TEXTE

DEUX GRANDS TYPES DE FICHIERS : BINAIRE VS NUMÉRIQUE

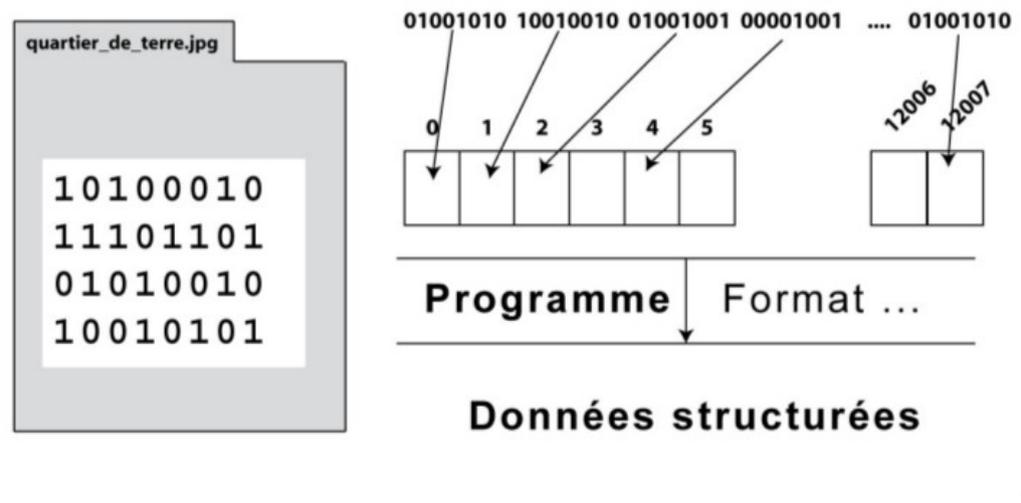
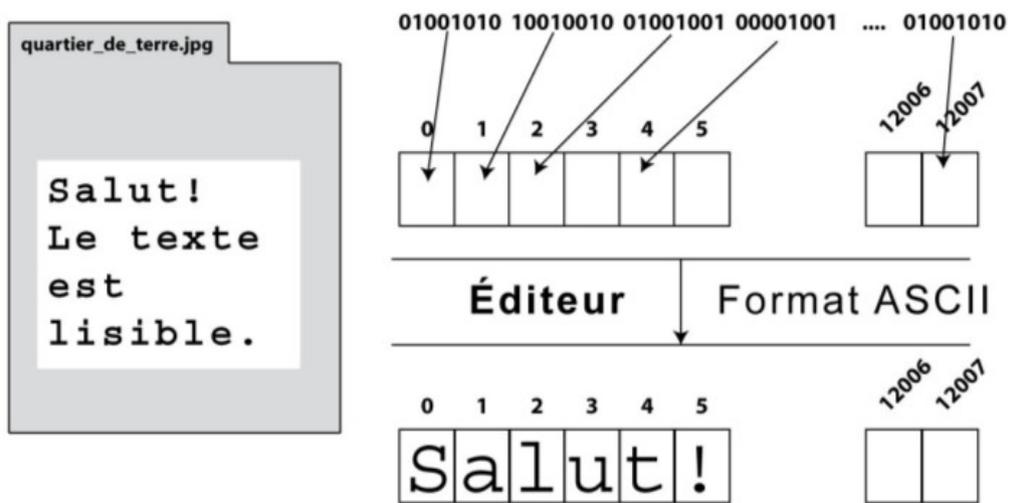
De façon générale un fichier binaire ne peut être "lu" que par un programme informatique, alors qu'un fichier texte peut être "lu" par être humain.

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.

dessine.c

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

L'EXÉCUTABLE

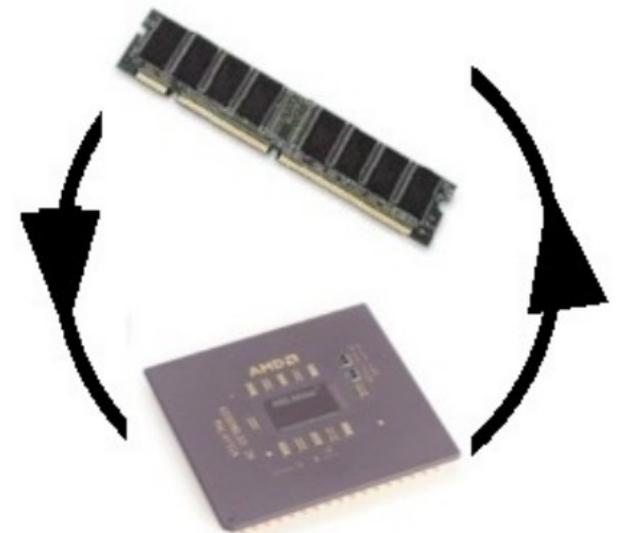
- Exprime les mêmes tâches dans un langage machine.
- Ce fichier ne fonctionne que sur des ordinateurs qui ont la même architecture.
- C'est un fichier binaire.

dessine

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

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.



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`.

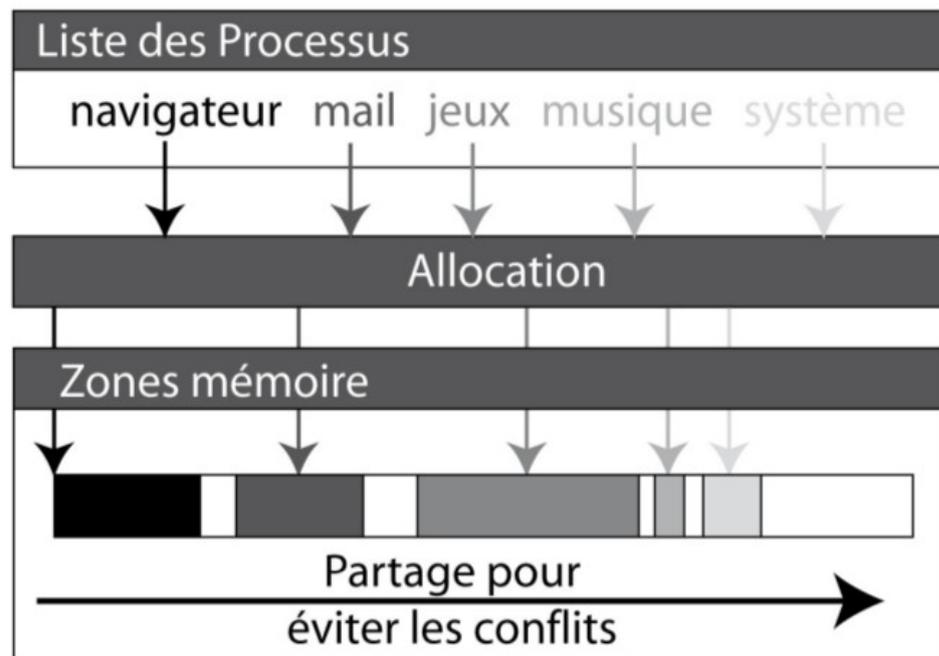
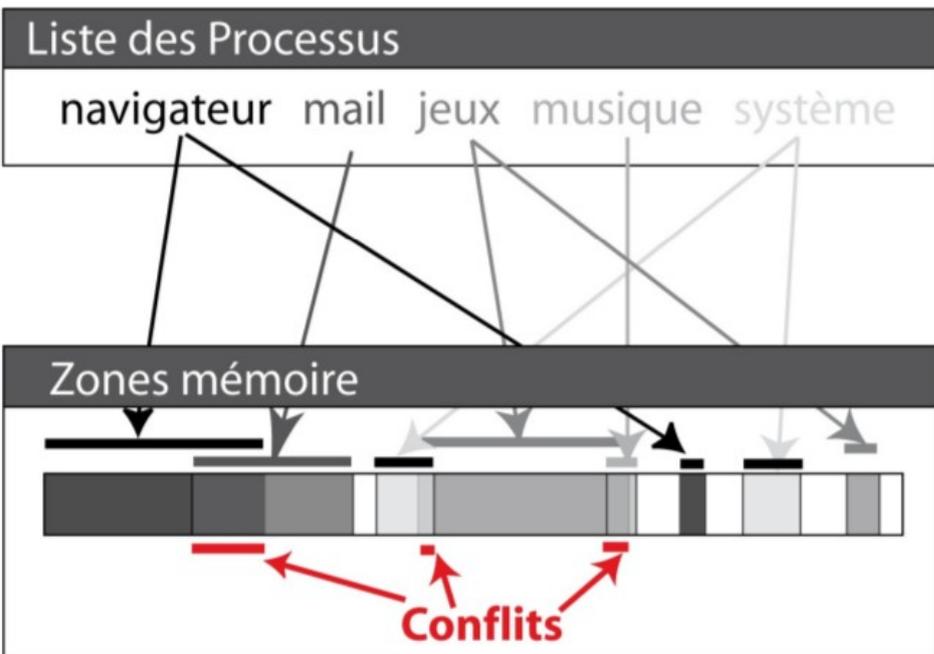
PPID - PARENT PROCESS IDENTIFIER

- 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.

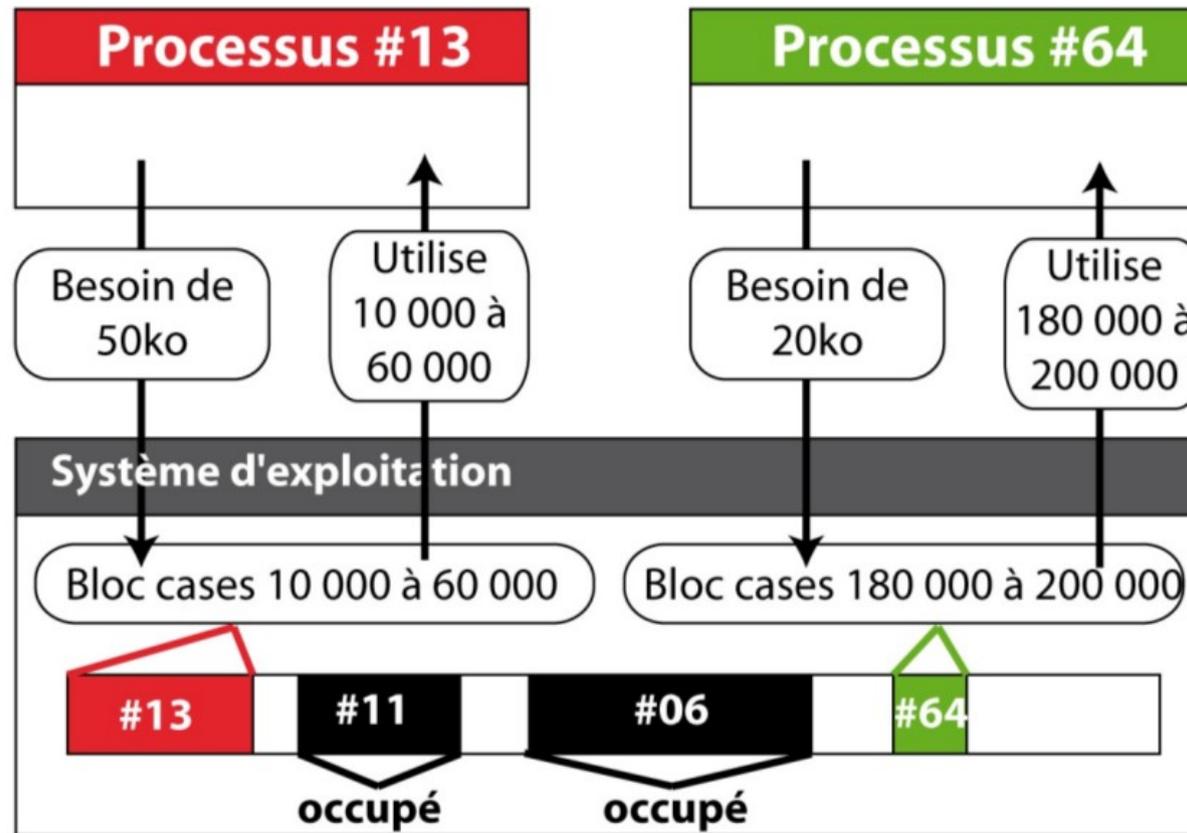
IL FAUT ASSURER L'INTÉGRITÉ DES DONNÉES !

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.

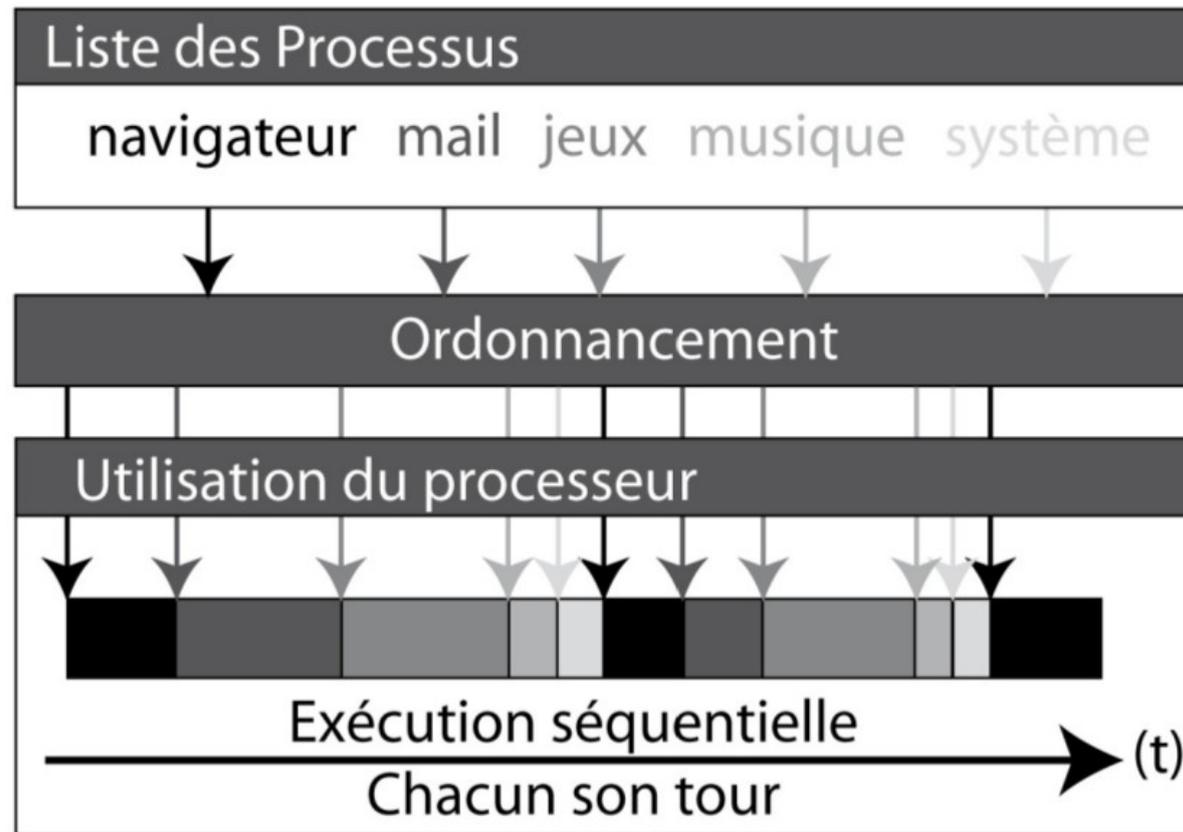
GESTION DE LA MÉMOIRE VIVE



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 
- 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 ~ ] █
```

Envoyer des Signaux aux Processus

- <ctrl> + c et <ctrl> + z
- Syntaxe
 - `kill [-signal] pid(s)`
 - `kill [-signal] %jobID`
- Envoie le signal spécifié au processus
 - TERM est le signal par défaut
 - `kill -1` affiche tous les signaux disponibles
- `killall`

Afficher les Travaux en Arrière Plan et Suspendus

- `jobs` affiche tous les processus en cours d'exécution en arrière plan ou ceux qui sont suspendus
- Le nombre entre crochets est le numéro de travail, utilisé pour tuer ou ramener des travaux au premier plan
- Les numéros de travail sont indiqués avec le symbole %

Reprendre des Travaux Suspendus

- Lorsqu'une commande est suspendue ou en arrière plan, elle peut être ramenée au premier plan avec **fg**
- Les travaux suspendus peuvent être relancés en arrière-plan avec **bg**
- Syntaxe
 - fg [numéro_travail]**
 - bg [numéro_travail]**

Modifier la Priorité d'Ordonnement de Processus

- À l'invocation d'un processus

Syntaxe :

```
nice [-n ajustement] commande
```

- Les processus sont ordonnancés avec la priorité 0 par défaut
- La valeur de priorité peut aller de -20 (priorité la plus haute) à 19 (la plus basse)

Modifier la Priorité d'Ordonnancement de Processus – suite

- `renice` change la priorité d'un processus en cours d'exécution

Syntaxe :

```
renice # [[-p|-g] PID] [[-u]utilisateur]
```

- `#` est la valeur de priorité
- Une fois que la valeur de priorité est augmentée, un utilisateur sans privilèges ne peut pas la diminuer

SECTION 5

Archivage et compression

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, ...

gzip

SYNTAXE

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

DESCRIPTION

- Comprime 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
```

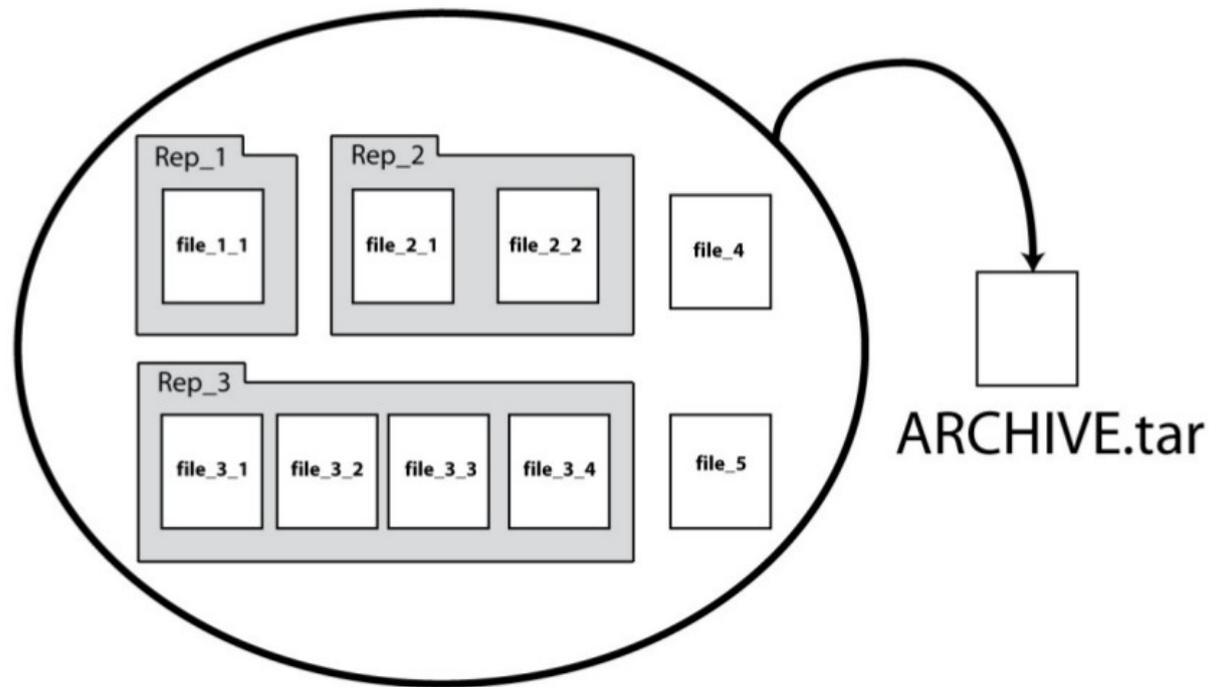
```
chez_moi/..... Répertoire Courant
├─ tellurique.tsv
└─ astronomie.txt..... Après gunzip
```

```
[ 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 cvf 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 à 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.
- L'option `f` (**F**ile), permet d'utiliser un fichier archive.

EXEMPLE D'UTILISATION:

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

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

tar(bis)

SYNTAXE

```
tar xvf nom_archive
```

DESCRIPTION

- Extrait les fichiers et répertoires d'une archive.
- Les fichiers sont placés dans le répertoire courant.
- L'option **x** (**eXtarct**) 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 xvfespace.tar
```

tar(ter)

SYNTAXE

```
tar cvzf nom_archive fichier_ou_repertoire <autres_sources>
```

SYNTAXE

```
tar xvzf 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 cvzf espace.tar astronomie.txt Images/
```

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

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

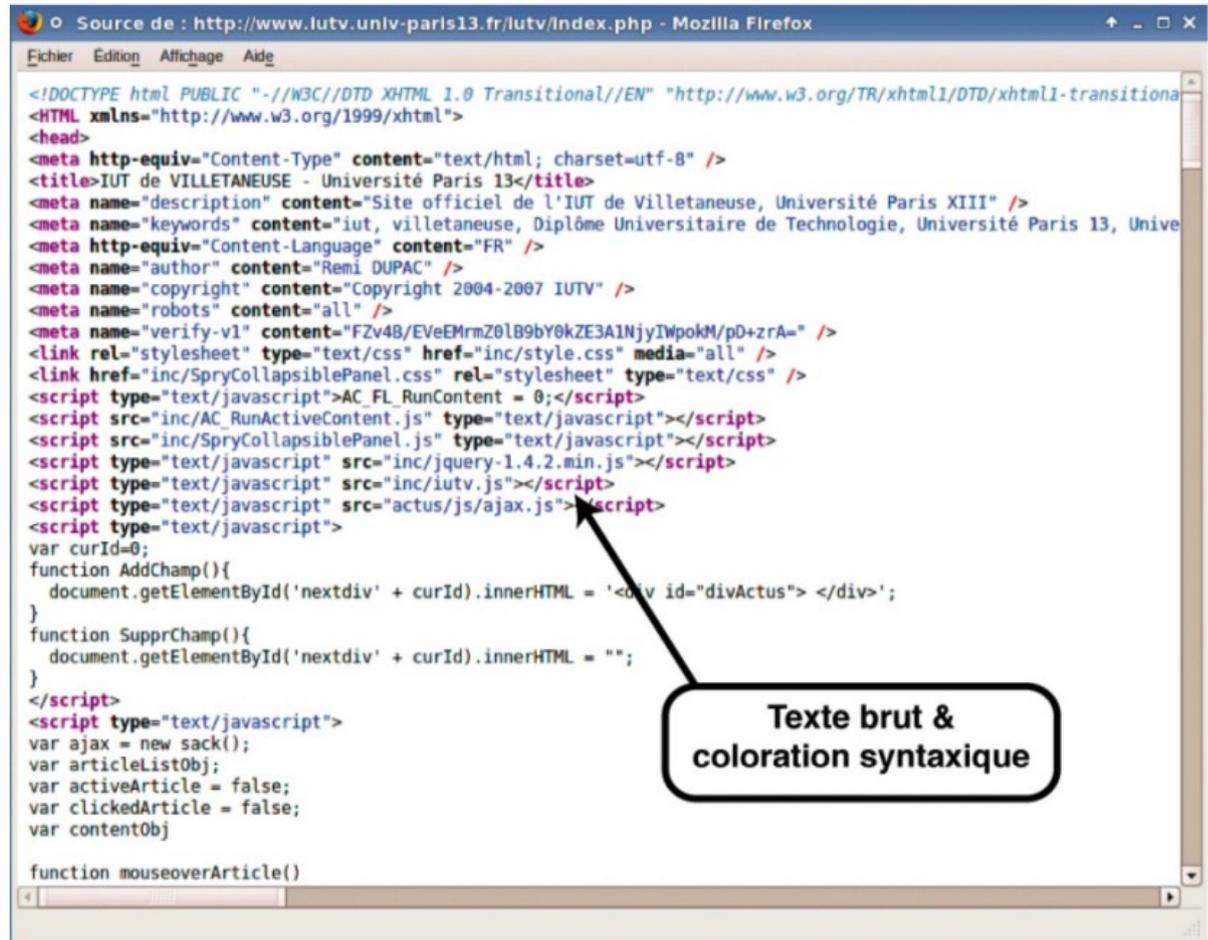
SECTION 6

Edition et manipulation
de fichiers

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.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>IUT de VILLETANEUSE - Université Paris 13</title>
<meta name="description" content="Site officiel de l'IUT de Villetaneuse, Université Paris XIII" />
<meta name="keywords" content="iut, villetaneuse, Diplôme Universitaire de Technologie, Université Paris 13, Unive" />
<meta http-equiv="Content-Language" content="FR" />
<meta name="author" content="Remi DUPAC" />
<meta name="copyright" content="Copyright 2004-2007 IUTV" />
<meta name="robots" content="all" />
<meta name="verify-v1" content="FZv4B/EVeEMrmZ0lB9bY0kZE3A1NjyIwpokM/pD+zrA=" />
<link rel="stylesheet" type="text/css" href="inc/style.css" media="all" />
<link href="inc/SpryCollapsiblePanel.css" rel="stylesheet" type="text/css" />
<script type="text/javascript">AC_FL_RunContent = 0;</script>
<script src="inc/AC_RunActiveContent.js" type="text/javascript"></script>
<script src="inc/SpryCollapsiblePanel.js" type="text/javascript"></script>
<script type="text/javascript" src="inc/jquery-1.4.2.min.js"></script>
<script type="text/javascript" src="inc/iutv.js"></script>
<script type="text/javascript" src="actus/js/ajax.js"></script>
<script type="text/javascript">
var curId=0;
function AddChamp(){
  document.getElementById('nextdiv' + curId).innerHTML = '<div id="divActus"> </div>';
}
function SupprChamp(){
  document.getElementById('nextdiv' + curId).innerHTML = "";
}
</script>
<script type="text/javascript">
var ajax = new sack();
var articleListObj;
var activeArticle = false;
var clickedArticle = false;
var contentObj;

function mouseoverArticle()
```

Texte brut & coloration syntaxique

LA COLORATION SYNTAXIQUE

- L'éditeur reconnaît le format du fichier et applique des règles de coloration sur certains mots clefs, délimiteurs, ... ,
- Contrairement aux documents mis en forme (pdf, doc), la coloration des caractères n'est pas sauvee 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
 + 	Tout sélectionner
 + 	Copier la sélection
 + 	Couper la sélection
 + 	Coller la sélection là où se trouve le curseur
 + 	Annuler la dernière opération
 + 	Enregistrer le document
 +  + 	Enregistrer le document sous ...
 + 	Ouvrir un document
 + 	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  +  ,
- 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  .
- Pour passer à la page suivante, l'utilisateur presse la touche  .
- 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

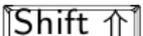
- Affiche le contenu d'un fichier,
- Permet de naviguer en avant et en arrière dans le fichier.
- Permet d'effectuer des recherches de mot(if)s.

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
	Affiche l'aide (abrégé des commandes)
	Avancer d'une page (forward)
	Reculer d'une page (backward)
	Avancer d'une ligne
	Reculer d'une ligne

Combinaison de touches	Action
	Quiter
	Aller à la première ligne
 + 	Aller à la dernière ligne
 + 	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 guillemets 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 ~ ] █
```

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	Mercury Venus
2	Earth 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

Commande #1 :

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

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

- 2 lignes.

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 précédente 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 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:

planete.v1		planete.v2	
1 Mercure		1 Planetes	← Insertion
2 Venus		2 Mercure	
3 Terre		3 Venus	
4 Mars		4 Terre	
5 Jupiter		5 Mars	
6 Saturn		6 Jupiter	
7 Uranus		7 Saturne	← Modification
8 Neptune		8 Uranus	
9 Pluton		9 Neptune	← Délétion

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

Outils de Formatage

- `expand` – expansion de tabulations en espaces
- `fmt` – reformate le texte en paragraphes

Autres Outils de Traitement de Chaînes

- `paste` – fusionne des fichiers
- `tr` – traducteur de caractères

Vérification d'Orthographe

- Correcteur d'orthographe interactif
- Moyen facile pour vérifier l'orthographe dans un fichier
 - `$ aspell check letter.txt`
- Peut créer des dictionnaires personnels

Filtrage avec des Expressions Régulières

- Les expressions régulières représentent un moteur de filtrage
- Utilisées par de nombreux outils, incluant `grep` `sed` `less` `awk` ...
- Valeurs
 - Puissance plutôt que facilité d'utilisation
 - Recherches gourmandes !
- Deux types de base et étendue

Les Caractères Génériques

- Les caractères génériques représentent un autre caractère simple :
 - tout caractère simple
 - [abc] tout caractère simple dans l'ensemble
 - [a - c] tout caractère simple dans la gamme
 - [^abc] tout caractère simple *en dehors* de l'ensemble
 - [^a - c] tout caractère simple *en dehors* de la gamme

Les Modificateurs

- Les modificateurs déterminent le nombre de caractères précédents
 - * zéro ou plus caractère précédent
 - \+ un ou plus caractère précédent
 - \? zéro ou un caractère précédent
 - \{i\} exactement i caractère précédent
 - \{i,\} i ou plus caractère précédent
 - \{i,j\} de i à j caractère précédent

Les Ancres

- Les ancres comparent le début ou la fin d'une ligne ou d'un mot
 - ^ la ligne commence avec
 - \$ la ligne se termine avec
 - \< le mot commence avec
 - \> le mot se termine avec

Combinaisons regex

- Les expressions régulières sont les plus utiles combinées entre elles

<code>. *</code>	zéro ou plus tout caractère
<code>[a - z] *</code>	zéro ou plus lettres
<code>\<cat \></code>	le mot 'cat'
<code>ab..ef</code>	ab et ef séparés de deux caractères
<code>.\{32\}</code>	32 fois tout caractère
<code>*</code>	un astérisque littéral

Les Expressions Régulières Étendues

- Une extension de l'ensemble des expressions régulières
- Les outils qui utilisent les regex étendues :
 - `egrep`
 - `grep -E` (équivalent à `egrep`)
 - `awk`

Syntaxe des regex Étendues

- La plupart des expressions régulières de base sont supportées
- Les expressions régulières de base précédées d'une oblique inverse n'en ont plus besoin
 `a{10,12}` compteur : 10, 11 ou 12 lettres «a»
- Exception : les ancres de mots («`\<` » et «`\>` ») ont toujours besoin d'une oblique inverse

awk

- Langage de programmation pour l'édition de texte
- Recherche des lignes dans un fichier correspondant à un ou plusieurs modèles
- Effectue les actions spécifiées sur les lignes trouvées
- Les modèles de recherche sont des expressions régulières étendues

Utiliser awk

- Les programmes `awk` sont dirigés par les données
- Les règles `awk` contiennent un modèle et une action dans des accolades

modèle { action }

- L'action est appliquée sur toute ligne correspondant au modèle

```
awk '/bash/ { print }' /etc/passwd
```

```
awk '/[2-5]+/ { print }' /etc/inittab
```

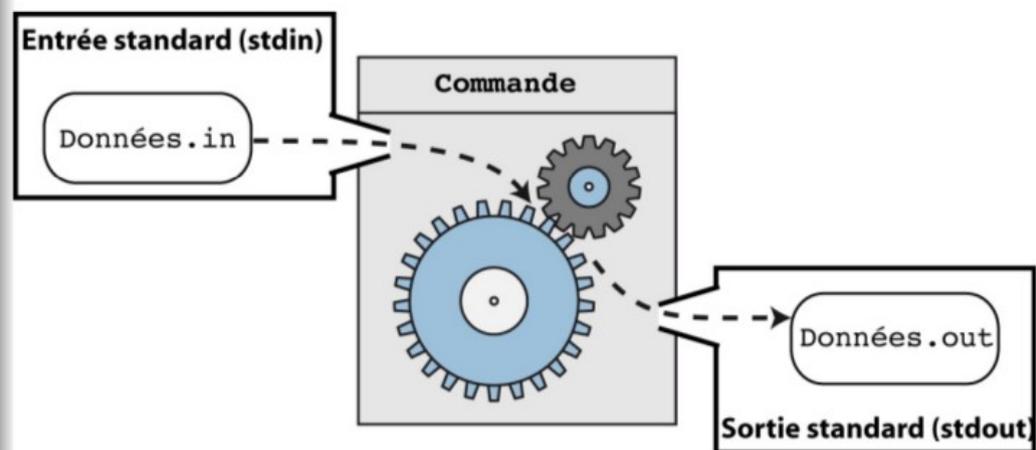
SECTION 7

E/S et tubes

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

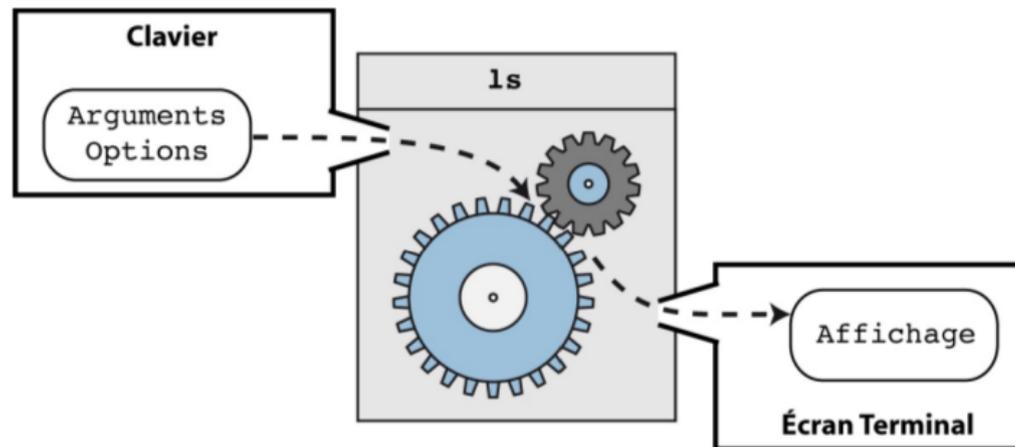
Pour fonctionner, un programme a donc besoin de lire des données (flux d'entrée : input) et d'écrire les résultats de ses évaluations (flux de sortie : output). On distingue 3 types de flux de données :

- **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 vues : 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`

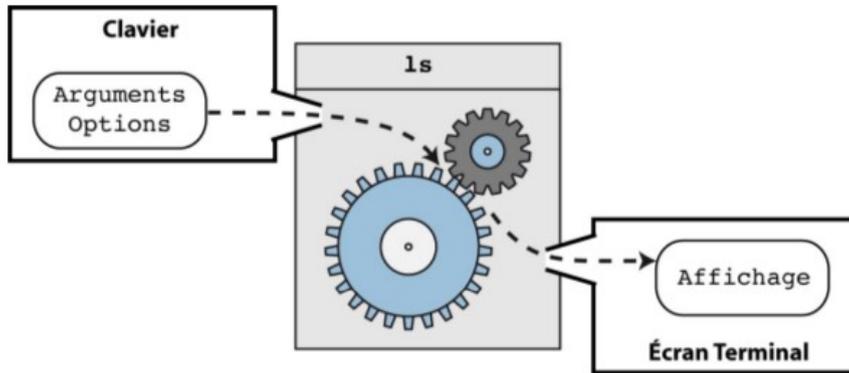
- **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec écrasement du contenu** si le fichier de sortie existe déjà.

- `command 2>> fichier.err`

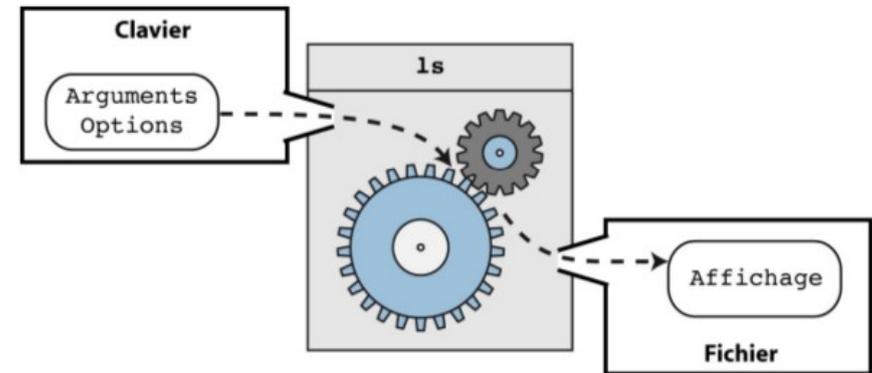
- **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec préservation du contenu** si le fichier de sortie existe déjà.

EXEMPLE DE REDIRECTION

COMPORTEMENT PAR DÉFAUT DE LA COMMANDE `ls`



REDIRECTION DE LA SORTIE 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.

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

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

[ login@localhost ~ ] █
```

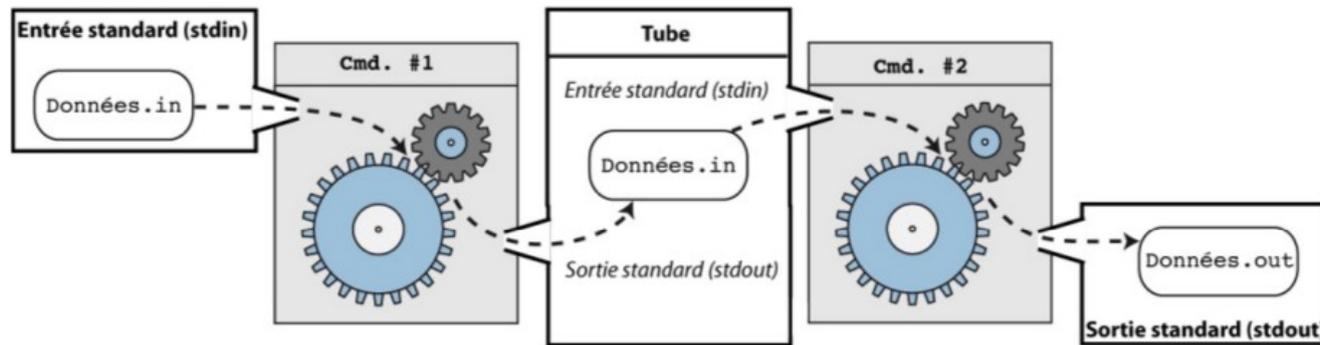
La sortie standard de la première commande `ls` est redirigée vers le fichier `1.out`. La liste du contenu du répertoire courant est écrite dans le fichier `1.out`.

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

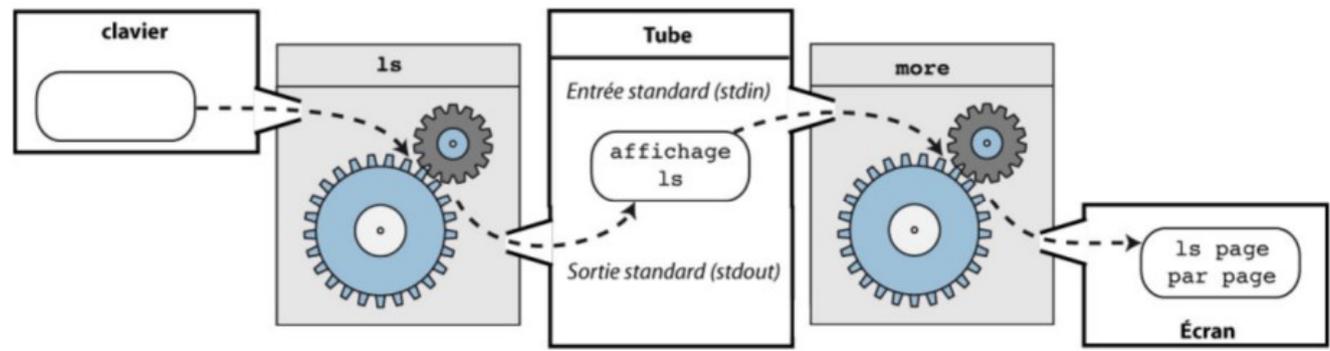
```
betelgeuse.jpg      etacentauri.jpg
soleil.jpg          syrius.gif
vega.png
[ login@localhost ~ ] █
```

```
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

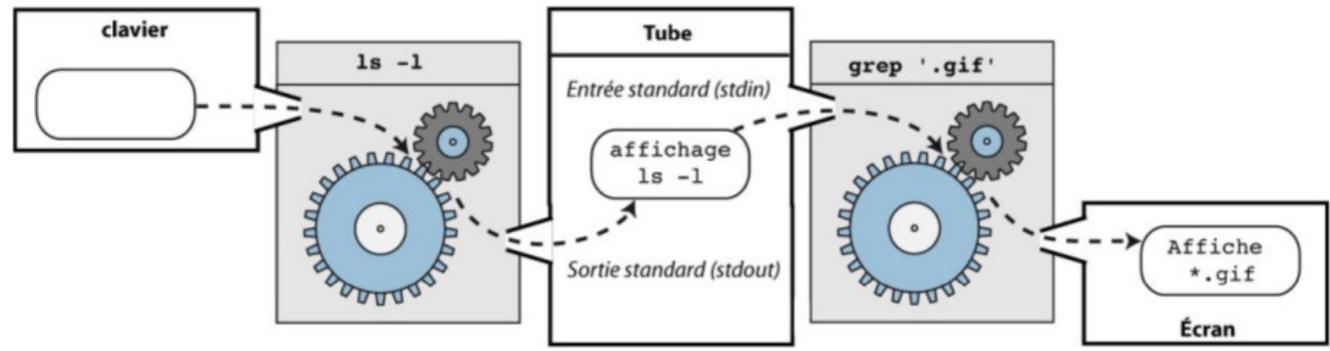
[ login@localhost ~ ] █
```

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

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 souhaite 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 souhaite 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

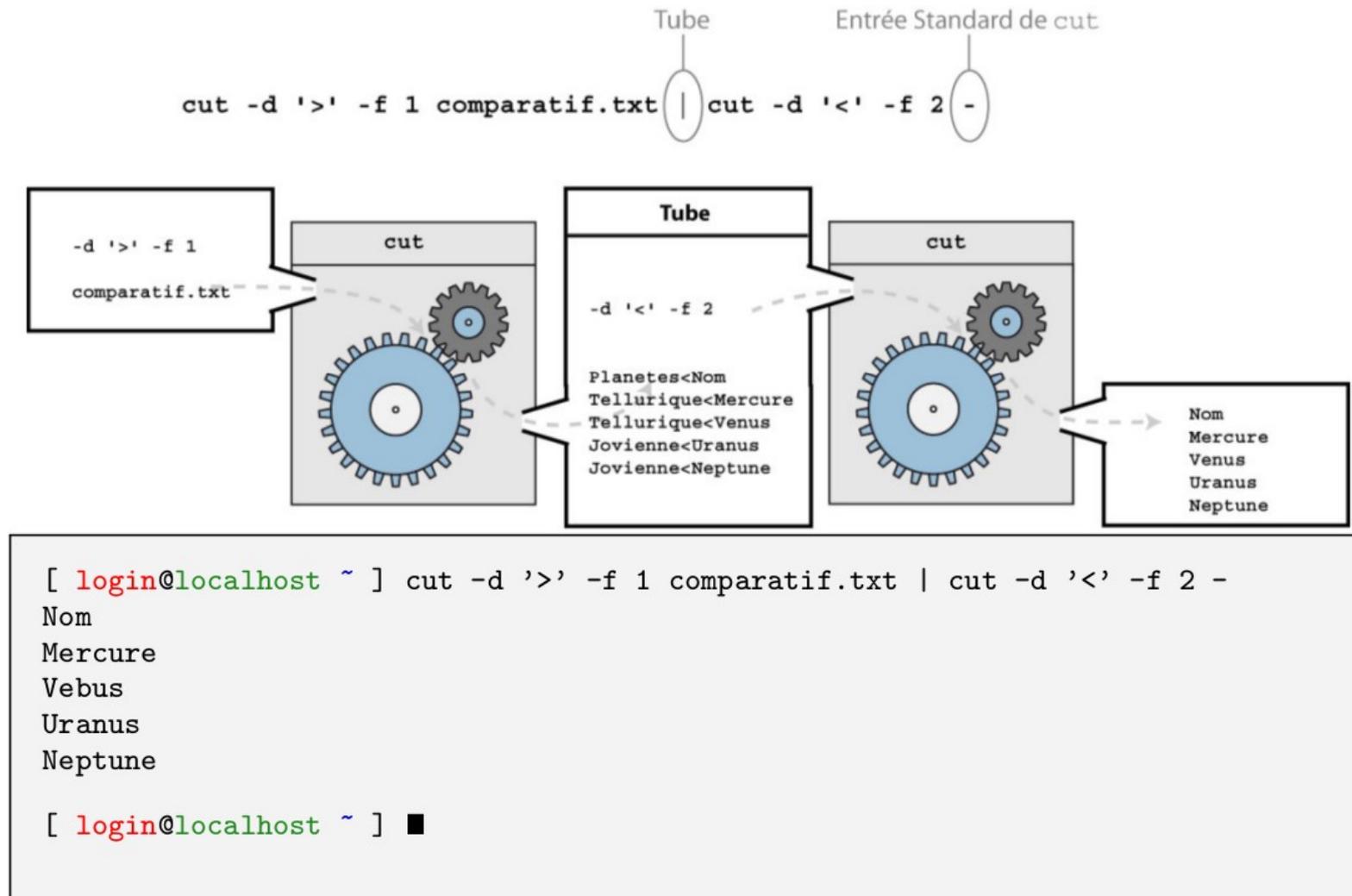
[ login@localhost ~ ] █
```

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 <.



tee

- Vous permet d'appliquer *tee* à un tube : redirection d'une sortie vers un fichier tout en continuant l'action du tube vers un autre programme

```
$ set | tee set.out | less
```
- Dans cet exemple, la sortie de **set** est écrite dans le fichier **set.out** tout en étant conduite vers **less**

SECTION 8

Execution et `.bashrc`

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`, ...),
- Pourtant ...

DES CHEMINS QUI MÈNENT NULLE PART

- les noms des commandes (`cd`, `ls`, `python`, `firefox` ...) sont toujours saisis 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

- ⇒ Il s'agit d'une liste de répertoires que l'interpréteur parcourt 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:

```
/..... Répertoire Racine
├── bin/
│   ├── ls ..... Exécutable #1
│   └── ...
└── home/
    ├── chez_moi/ ..... Répertoire Courant
    │   └── bin/
    │       └── ls ..... Exécutable #2
```

```
[ 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 définies 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
<code>/etc/profile</code>	root	Tous	Au début de chaque shell de login
<code>/home/chez_moi/.profile</code>	utilisateur	utilisateur	Au début de chaque shell de login
<code>/etc/bashrc</code>	root	Tous	Au début de chaque shell
<code>/home/chez_moi/.bashrc</code>	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 commande `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

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

EXEMPLE D'UTILISATION:

chez_moi/.. Répertoire Courant

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

```
[ 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 ~ ] █
```

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:

```
/..... Répertoire Racine
├── bin/
│   ├── ls ..... Exécutable #1
│   └── ...
└── home/
    ├── chez_moi/ ..... Répertoire Courant
    │   └── bin/
    │       └── ls ..... Exécutable #2
```

```
[ 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
```

SECTION 9

Scripts Bash

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 appelle *affectation* le fait d'associer une valeur à un nom de variable.

UTILISATION

- Les variables sont utilisées pour stocker des valeurs qui doivent être utilisées 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 '=') :
- Pour accéder au contenu d'une variable, on fait précéder son nom du caractère \$.

```
nom_de_la_variable=valeur
```

```
$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 à 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 son 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

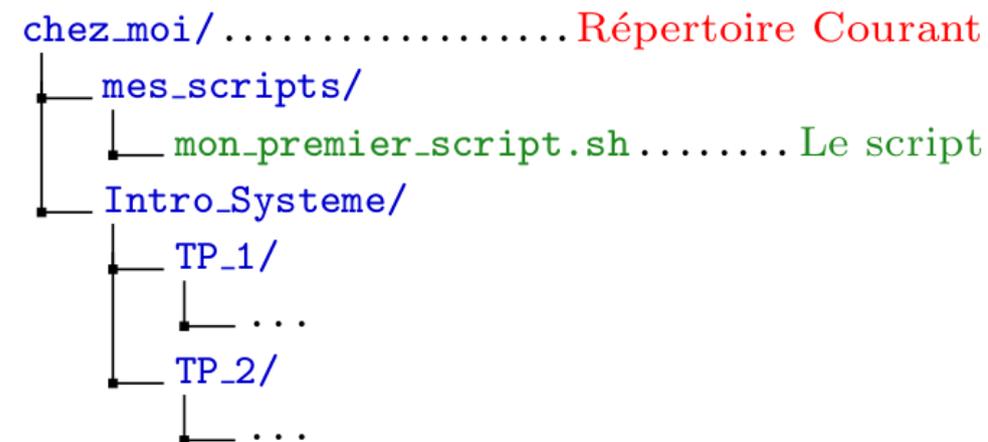
CAS D'ÉTUDE

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
```



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 ] █
```

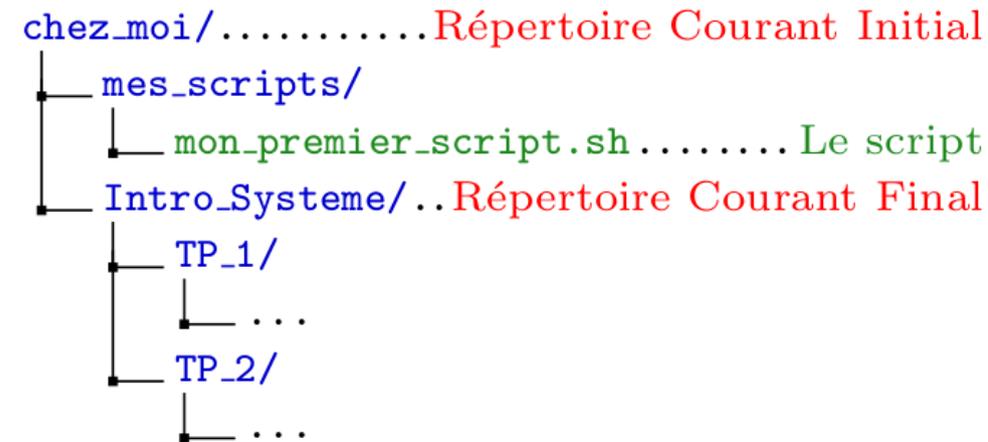
LES SCRIPTS

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
```



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

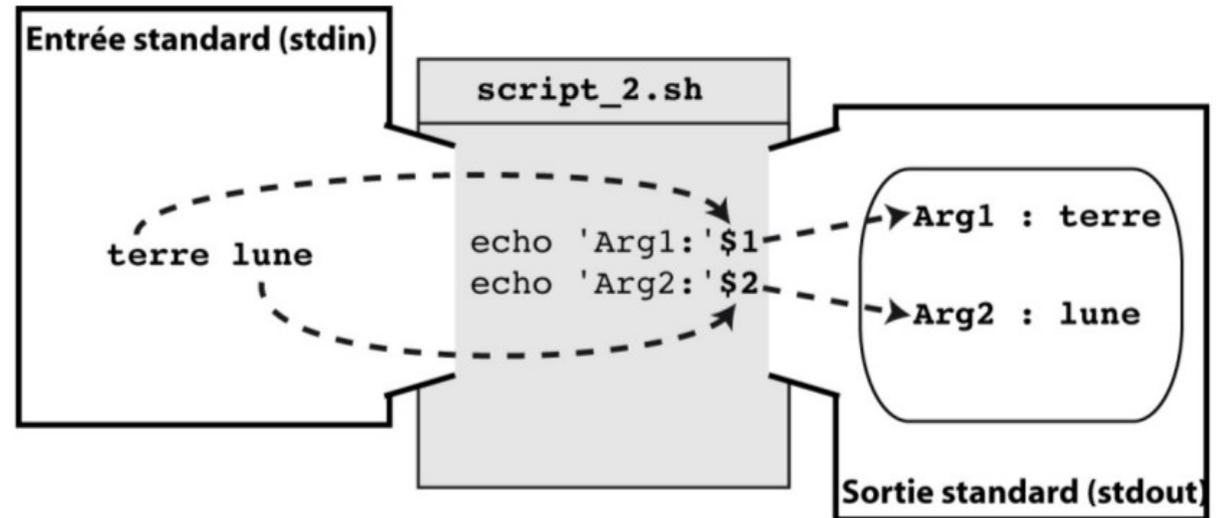
[ login@localhost ~/Intro_Systeme ] pwd
~/Intro_Systeme

[ 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 ~ ] █
```

Pour décaler les paramètres : `shift` (\$1 disparaît, \$2 devient \$1, etc.)

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 ~ ] █
```

Produire des Sorties

- Utiliser `echo` pour produire de simples sorties

```
echo 'Bienvenue au paradis de Linux !'
```

```
echo -n "Veuillez entrer le nom du fichier : "
```

- Utiliser `printf` pour produire des sorties formatées

```
printf "Le résultat est %0.2f\n" $RESULTAT
```

- Syntaxe semblable à la fonction `printf()` de C
- N'ajoute pas automatiquement de nouvelle ligne à la fin de la sortie

Manipuler les Entrées

- Utiliser `read` pour donner une valeur d'entrée à une variable shell :

```
echo -n "Entrez le nom du fichier : "
```

```
read NOMFICHIER
```

- **read** lit l'entrée standard et donne un mot à chaque variable
- Les mots restants sont donnés à la dernière variable

Utiliser des Fonctions dans les Scripts Shell

- Les scripts shell peuvent inclure des fonctions shell
- Les fonctions shell peuvent améliorer la lisibilité de programmes. Elles aident également à supprimer la répétition de code dans les scripts
- Les fonctions shell doivent être déclarées avant d'être utilisées

Utiliser des Fonctions – suite

- Des arguments peuvent être transmis à une fonction shell à l'aide de son propre ensemble de paramètres de position (**\$1**, **\$2**, etc.)

`maFonction $nomfichier`

La valeur de **\$nomfichier** sera disponible sous **\$1** au sein du corps de **maFonction**

- Les fonctions peuvent renvoyer des valeurs en utilisant le mot-clé **return** qui définit la valeur de la variable spéciale **\$?**

État de Sortie

- Les commandes se terminent avec un état de sortie
 - 0 pour un succès, de 1 à 255 pour un échec
 - L'état de sortie de la dernière commande exécutée est gardé dans la variable `$?` tout comme les valeurs retournées par les fonctions shell
- Les scripts shell peuvent définir un état de sortie avec la commande `exit`

```
exit 1 # Indique une erreur
```

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

TESTS SUR LES VALEURS NUMÉRIQUES

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

Structures de Sélection

Utiliser l'Instruction `case`

- L'instruction `case` offre une autre méthode pour effectuer des sélections qui peuvent être plus simples que de nombreux tests `if - then - else`

```
case variable in
  modèle1)
    faire quelque chose ;;
  modèle2)
    faire autre chose ;;
esac
```

Structures de Répétition

La Boucle `while`

- La structure de la boucle **while** offre une méthode utile pour effectuer un ensemble de commandes pendant qu'une condition reste vraie.

```
while condition
do
    commandes...
done
```

continue et break

- Les boucles peuvent être interrompues durant leur exécution
 - `continue` interrompt l'exécution de la boucle et re-examine la condition initiale, redémarrant parfois la boucle
 - `break` interrompt complètement le traitement de la boucle, passant au-delà de l'instruction `done`
 - `exit` quitte le script shell
 - Vous pouvez donner un état de sortie

Débogage de Scripts Shell

- Afin de déboguer un script shell, appelez l'interpréteur shell avec les options de débogage ou modifier le shebang pour inclure les options de débogage

```
bash -x nomscript
```

```
bash -v nomscript
```

```
#!/bin/bash -x
```

```
#!/bin/bash -v
```