

# Système d'exploitation

## Système d'exploitation - Utilisation

Mickael Rouvier

CERI – Avignon Université  
[mickael.rouvier@univ-avignon.fr](mailto:mickael.rouvier@univ-avignon.fr)





Section 1

---

## **Présentation sommaire de l'UCE**



a  
l

# Plan et objectifs du cours

- UE : Système d'exploitation
  - **Système d'exploitation - Utilisation (L1)**
  - Système d'exploitation – Administration (L2)
  - Système d'exploitation – Programmation (L3)
- Objectifs du cours
  - Découverte et prise en main de l'environnement GNU/Linux
  - Connaissance du fonctionnement GNU/Linux
- Plan du cours

## ● Cours 1

- Historique du SE
- Terminal
- Gestion des fichiers

## ● Cours 2

- Historique des commandes
- Manipulation des fichiers

## ● Cours 6

- Gestion de la mémoire et des processus
- Noyau, distribution et processus de démarrage

## ● Cours 3

- Redirection des flux
- Archive compressé

## ● Cours 4 et 5

- Programmation en bash

# Organisation

- **Attention** : il y a des choses que je ne dis qu'en cours, donc venez et soyez à l'heure
- Vous devez déposer tous vos TP sur la plateforme e-uapv
  - Envoyez une archive contenant tous les fichiers du TP et un README expliquant comment les faire marcher
  - Quoi qu'il arrive vous devez déposer votre TP dans l'état où il est.  
Ce n'est pas grave si vous n'avez pas fini.
- Notes
  - TP Final (Coefficient 0.4)
  - TP Continu (Coefficient 0.2)
  - Examen : Coefficient (0.4)

## Section 2

# **Introduction Système d'Exploitation (SE)**

al

# Introduction au Système d'Exploitation

- **SE les plus connus**

- Windows (dernière version : Windows 10)
- GNU/Linux (distribution bureau : Debian, Ubuntu, Fedora...)
- Mac OS X (dernière version : macOS Big Sur)

- **Qu'est-ce qu'un SE ? A quoi sert un SE ?**

- Quand les SE sont-ils apparus ?
- Quel est le rôle d'un SE ?
- Quels sont les services rendus par un SE ?



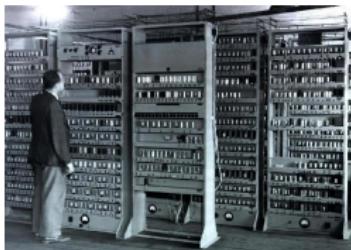
# Historique - Première génération : tubes à vide (1945)

- **Premier ordinateur ENIAC**

- Les premières machines à calculer sont construites au moyen de tubes à vides
- Ces machines énormes remplissaient des salles et étaient moins rapides qu'une calculatrice moderne
- Permettait de : table de tirs, décrypter

- **Pas besoin de système d'exploitation**

- Toute la préparation se faisait manuellement (câblé des branchements)
- Le programmeur opère avec la machine par interaction directe (beaucoup de manipulation et beaucoup de lourdeur)
- Concepteur = constructeur = programmeur = opérateur



# Historique - Deuxième génération : Traitement par lots (1956)

- **Problème :** Ordinateur non-programmable
  - Automatisation d'une suite de commandes (apparition des cartes perforées)
  - Enchaînement automatique des commandes : le **traitement par lots**
  - Aucun besoin d'assistance humaine pour réaliser l'opération (le programmeur n'est pas l'opérateur)
  - **Gestion d'E/S** (carte perforée)
- **Problème :** Certains programmes mal écrit pouvait boucler indéfiniment
  - **Gestion des processus** par limitation du temps
- **Problème :** un programme peut écrire n'importe où
  - **Gestion de la mémoire et des fichiers**



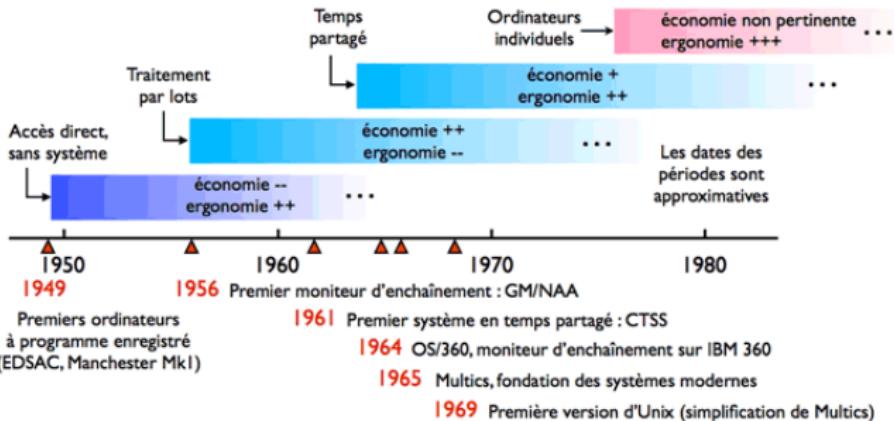
# Historique - Troisième génération : Temps partagé (1961)

- **Problème :** écrire les données sur un disque-dur prenaient beaucoup trop de temps
- **Problème :** besoin d'avoir un accès par utilisateur
  - Un ensemble de processus est lancé sur la machine et chacun est actif pendant un court très temps

# Historique - Quatrième génération : Apparition de la micro-informatique (1980)

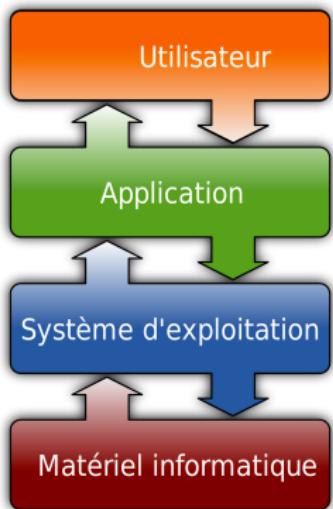
- **Problème :** Diversité des modèles (carte de son, CD-Rom, réseau...)
- **Problème :** Besoin d'une interface graphique (X11, Xerox...)
- **Problème :** Diversification des supports

# Historique - Résumé



- **Accès direct, sans système d'exploitation**
- **Traitement par lots**
- **Temps partagé**
- **Ordinateurs individuels**

# Quel est le rôle d'un SE



- **Quel est le rôle d'un SE**

- Permet de faire le lien entre les ressources matérielles d'un ordinateur et les applications de l'utilisateur (ex. donner une interface commune pour écrire sur un fichier qui peut se trouver sur un disque, clé USB, réseau... )
- Assure le démarrage et gérer l'accès à des ressources communes entre plusieurs utilisateurs ou programmes (ex. mémoire, disque, accès réseau...)

# Les services rendus par un SE

- **Gestion des fichiers**

- Gérer l'arborescence logique des fichiers
- Distribution physique sur le matériel de stockage (disque dur, clef usb...)

- **Gestion de la mémoire**

- Attribution de la mémoire à chaque application
- Partagée l'accès de la mémoire entre plusieurs applications

- **Gestion des applications**

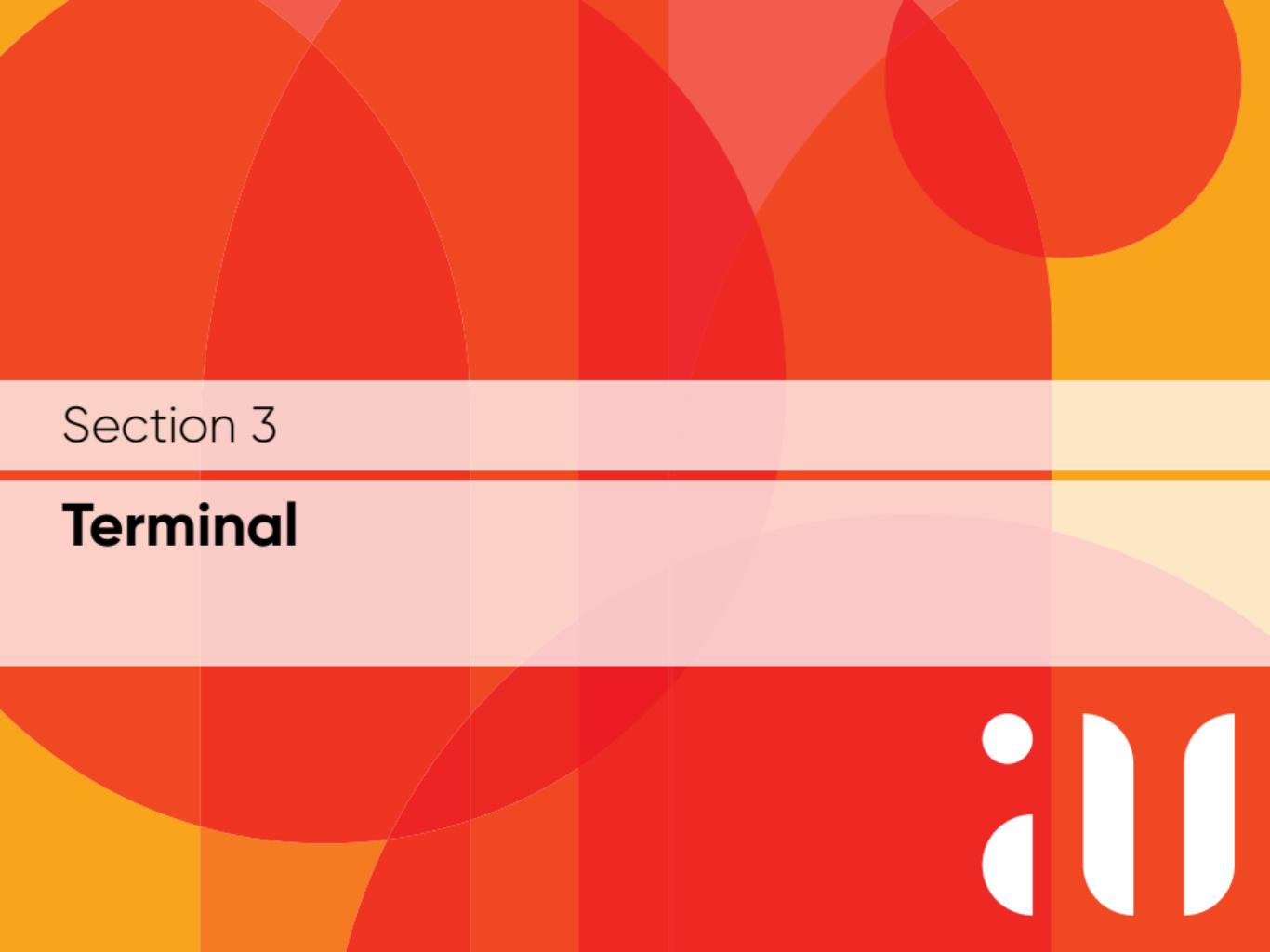
- Ordonnancement des applications

- **Gestion des entrées/sorties**

- Carte réseau, carte son, imprimante...

- **Gestion d'une interface entre le matériel informatique et l'application**

- Interface en ligne de commande
- Interface graphique (window manager : composite window manager, tiling manager...)



Section 3

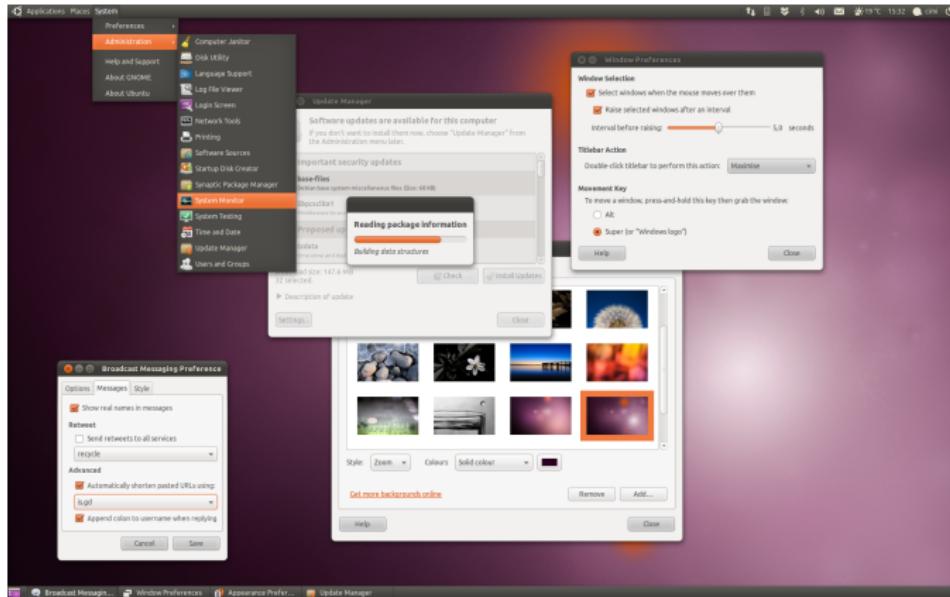
---

## Terminal

aid

# Interaction avec le SE

- En général en cliquant dans des fenêtres



# Problème fenêtrage

- **Inconvénient** de demander un service (ou exécuter une commande) par la méthode souris/fenêtre
  - Il doit y avoir un clic par commande possible (*si nombre de commande important alors nombre de fenêtre/bouton important*)
  - Ce n'est pas personnalisable (*le bouton qui fait la commande que vous voulez existe-t-il ?*)
  - Pas facilement automatisable (*on ne peut pas programmer facilement le clic de la souris dans une fenêtre*)

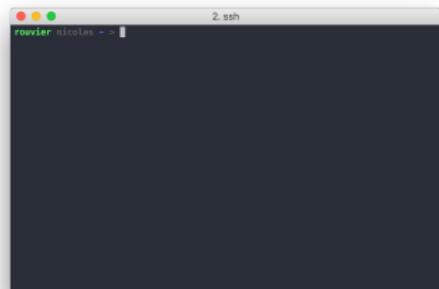
# Qu'est ce qu'un Terminal

- **Terminal** : C'est un écran noir qui est en attente d'une instruction d'une commande



# Introduction au Terminal

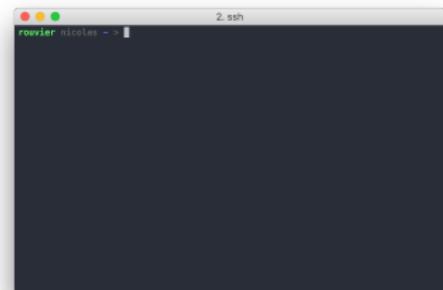
- Terminal
  - Lit et interprète les commandes (ce que vous tapez au terminal)
  - Ces commandes permettent entre autre d'interagir avec le noyau d'un SE
  - Quand une commande se terminent, le terminal attend une autre commande
- Intérêt du mode terminal
  - **Rapidité** : certains actions sont beaucoup plus rapide
  - **Exhaustivité** : le mode graphique par souci de simplicité, n'affiche aucune information pour trouver le problème
  - **Convivialité** : plus simple de donner une commande que d'expliquer là où il faut cliquer



# Un peu de vocabulaire

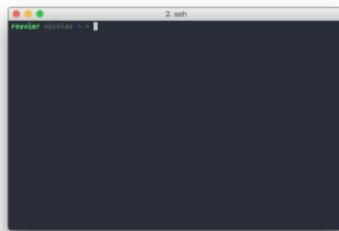
- **Terminal** : un écran noir qui est en attente d'une instruction, d'une commande
- **Interpréteur de commande** (ou **shell**) :
  - Le rôle de l'interpréteur de ligne de commande est de traiter une ligne de commande, comprendre son contenu et exécuter la tâche associée.
  - Une fois une ligne de commande écrite dans un terminal, celui-ci l'envoie à l'interpréteur de ligne de commande, qui la décortique et l'exécute.
- **Prompt** :

invite/> ■



# Différence entre Terminal et Shells

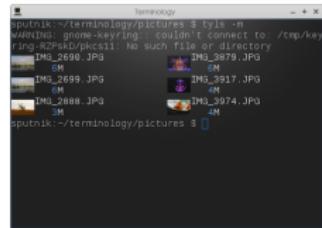
- Terminal



iTerm2



Cool Retro Term



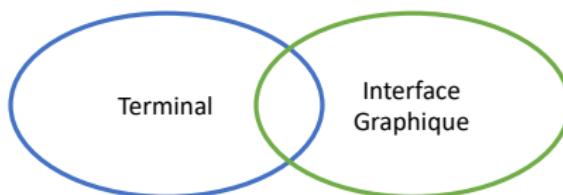
Terminology

- Shell

- Bash (Bourne Again Shell) : shell par défaut dans Ubuntu
  - Historique avancée
- zsh (Z Shell)
  - Complétion des commandes
  - Correction orthographique des commandes
- csh (C-Shell)

# Le Terminal est-il dépassé ?

- **Le Terminal est-il dépassé ?** La question ne doit même pas se poser



- **Inconvénient :**
  - Apprentissage plus long
  - Interface plus austère
- **Avantage :**
  - Nombre de fonctionnalité plus importante
  - Automatisation des fonctionnalités

**Le Terminal n'est pas prêt de disparaître (ex. nouveau terminal de Windows 10).**

# Command Line Argument - Partie 1

Une commande peut être suivi d'options et d'arguments :

```
invite/> command [-option(s)] [argument(s)]
```

## Convention POSIX :

- Une option est un hyphen suivit d'un caractère alphanumérique (par exemple : **-o**)
- Une option peut demander un argument (lequel doit apparaître immédiatement après l'option); (par exemple : **-o argument** ou **-oargument**)
- Les options qui ne demandent pas d'argument peuvent être tous groupé après un hyphen (par exemple : **-lst** est équivalent à **-l -s -t**)
- Les options peuvent apparaître dans n'importe quel ordre (par exemple : **-lst** est équivalent à **-lts**)
- Les options peuvent apparaître plusieurs fois (par exemple : **-l -s -t -s**)

# Command Line Argument - Partie 2

## GNU et les options longues :

- Les options longues sont deux hyphens suivis de caractères d'un ou plusieurs caractères alphanumérique et un trait d'union (par exemple : **-long-option**)
- On peut spécifier un argument à une option longue (par exemple : **-long-option=argument**)

## Unix vs BSD :

- Les options des programmes BSD ne commencent pas par un hyphen (par exemple : **tar zxvf file.tar.gz**)

## Terminologie : Options et arguments

# Command Line Argument - Partie 3

Comment prendre en compte les arguments dans un programme :

```
1 #!/bin/python
2 import sys
3
4 print sys.argv[0]
5 print sys.argv[1]
6 print sys.argv[2]
```

```
1 #include <iostream>
2
3 int main(int argc, char** argv) {
4     cout<< argv[0] << endl;
5     cout<< argv[1] << endl;
6     cout<< argv[2] << endl;
7
8     return 0;
9 }
```

```
invite/> ./main.py aaa bbb
main.py
aa
bb
```

```
invite/> ./main aaa bbb
main
aa
bb
```

# Obtenir de l'aide sur une commande - Partie 1

Le **man** est une commande qui permet d'obtenir de l'aide aux commandes du shell.

```
man [-s section] nom_de_commande
```

- Section : les pages du man sont divisés en plusieurs sections
  - Programmes exécutables ou commandes de l'interpréteur de commandes (shell)
  - Appels système (Fonctions fournies par le noyau)
  - Appels de bibliothèque (fonctions fournies par des bibliothèques)
  - Fichiers spéciaux (situés généralement dans /dev)
  - Formats des fichiers et conventions (par exemple /etc/passwd)
  - Jeux
  - Divers
  - Commandes de gestion du système (généralement réservées au superutilisateur)
  - Interface du noyau Linux

# Obtenir de l'aide sur une commande - Partie 2

```
invite> man ls
```



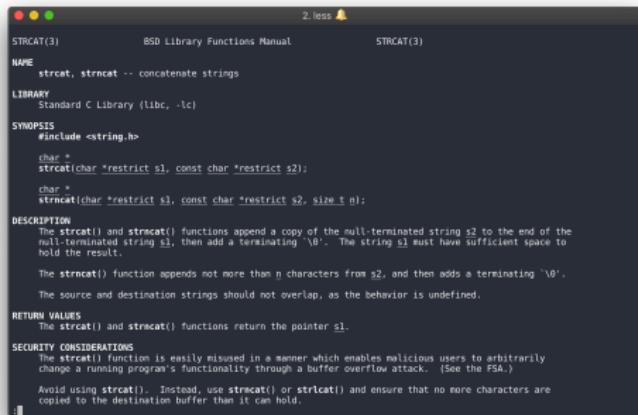
- Interactivité

- **flèches directionnels** : Naviguer dans la page de manuel
- **q** : Quitter
- **h** : Afficher l'aide
- **/pattern** : Rechercher le *pattern* en avant
- **?pattern** : Rechercher le *pattern* en arrière
- **n** : Aller à l'occurrence suivant
- **N** : Aller à l'occurrence précédent

# Obtenir de l'aide sur une commande - Partie 3

Le **man** permet aussi d'avoir des informations sur les fonctions du langage C.

```
invite/> man strcat
```



The screenshot shows a terminal window with a dark background and white text. The title bar reads "2. less". The content is the man page for the strcat function. It includes sections for NAME, SYNOPSIS, DESCRIPTION, RETURN VALUES, and SECURITY CONSIDERATIONS. The SYNOPSIS section shows two prototypes for the strcat function. The DESCRIPTION section explains that the function appends a copy of the null-terminated string s2 to the end of the null-terminated string s1, then adds a terminating '\0'. The RETURN VALUES section states that both functions return a pointer to s1. The SECURITY CONSIDERATIONS section warns that strcat is easily misused to enable buffer overflow attacks.

```
2. less
=====
BSD Library Functions Manual      STRCAT(3)
=====
NAME
    strcat, strncat -- concatenate strings
LIBRARY
    Standard C Library (libc, -lc)
SYNOPSIS
    #include <string.h>
    char *
    strcat(char *restrict s1, const char *restrict s2);
    char *
    strncat(char *restrict s1, const char *restrict s2, size_t n);
DESCRIPTION
    The strcat() and strncat() functions append a copy of the null-terminated string s2 to the end of the null-terminated string s1, then add a terminating '\0'. The string s1 must have sufficient space to hold the result.
    The strncat() function appends not more than n characters from s2, and then adds a terminating '\0'.
    The source and destination strings should not overlap, as the behavior is undefined.
RETURN VALUES
    The strcat() and strncat() functions return the pointer s1.
SECURITY CONSIDERATIONS
    The strcat() function is easily misused in a manner which enables malicious users to arbitrarily change a running program's functionality through a buffer overflow attack. (See the FSA.)
    Avoid using strcat(). Instead, use strncpy() or strlcat() and ensure that no more characters are copied to the destination buffer than it can hold.
:|
```

## Section 4

# Système de Gestion de Fichiers

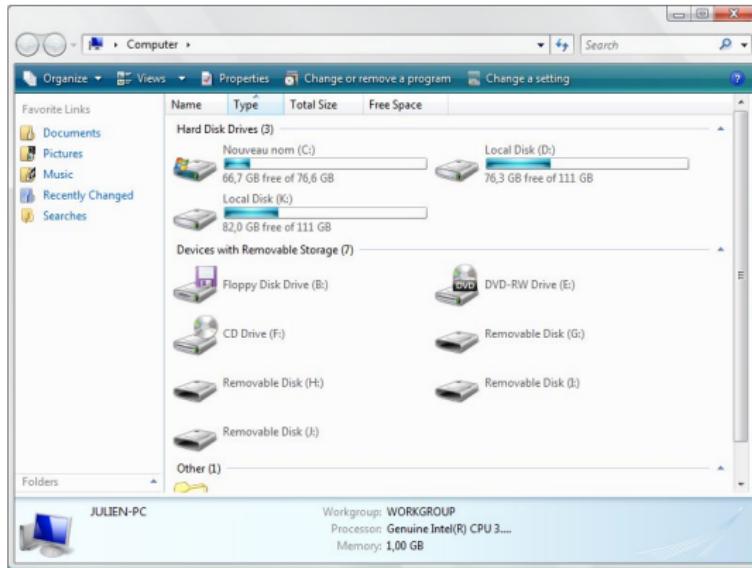
aid

# Définition

- **Définition** : le système de gestion de fichiers est la partie la plus visible d'un système d'exploitation qui se charge de gérer le stockage et la manipulation de fichiers sur une unité de stockage : partition, disque, CD, disquette...
- Il existe plusieurs système de gestion de fichiers : FAT, FAT32, NTFS, Ext3, Ext4, Btrfs, HFS, HFS+...
  - La taille maximale d'un fichier
  - La taille maximale d'une partition
  - La gestion des droits d'accès aux fichiers et répertoires
  - La journalisation
  - Compression de la partition
  - Chiffrage des données
  - Permet de prendre des instantanés (snapshots)
  - ...
  - Organisation des fichiers/dossier

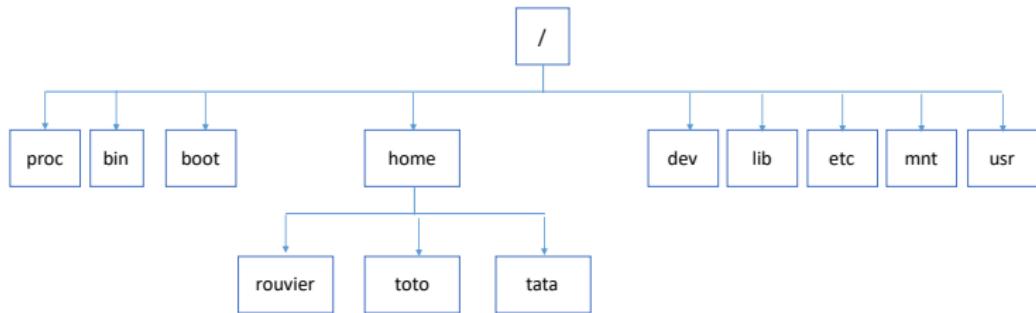
# Organisation des fichiers/dossiers - Windows

- Windows plusieurs racines (C ; D ; E ; ...)



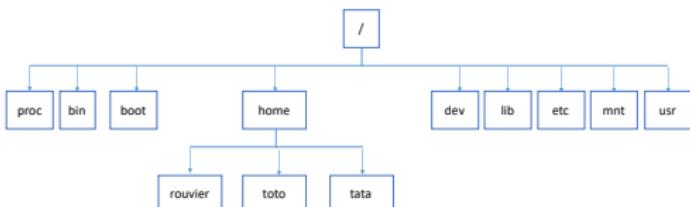
# Organisation des fichiers/dossiers - Linux

- Modèle hiérarchique en arbre inversé - 1 seul racine noté "/"



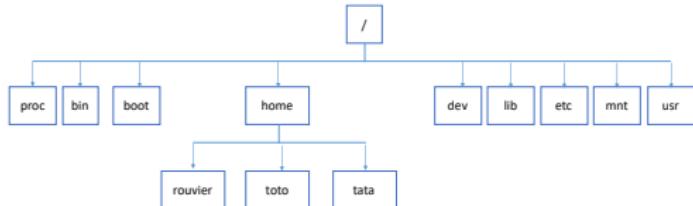
# Arborescence - Linux

- Modèle hiérarchique en arbre inversé - 1 seul racine noté "/"
  - **bin** : contient des applications (exécutables) susceptibles d'être utilisés par tous les utilisateurs de la machine.
  - **boot** : fichiers permettant le démarrage de Linux
  - **dev** : fichiers contenant les périphériques
  - **etc** : fichiers de configuration
  - **home** : répertoires personnels des utilisateurs
  - **mnt** : périphérique amovible
  - **opt** : contient des informations système
  - **root** : c'est le dossier personnel de l'utilisateur "root"
  - **usr** : dossier dans lequel vont s'installer la plupart des programmes demandés par l'utilisateur

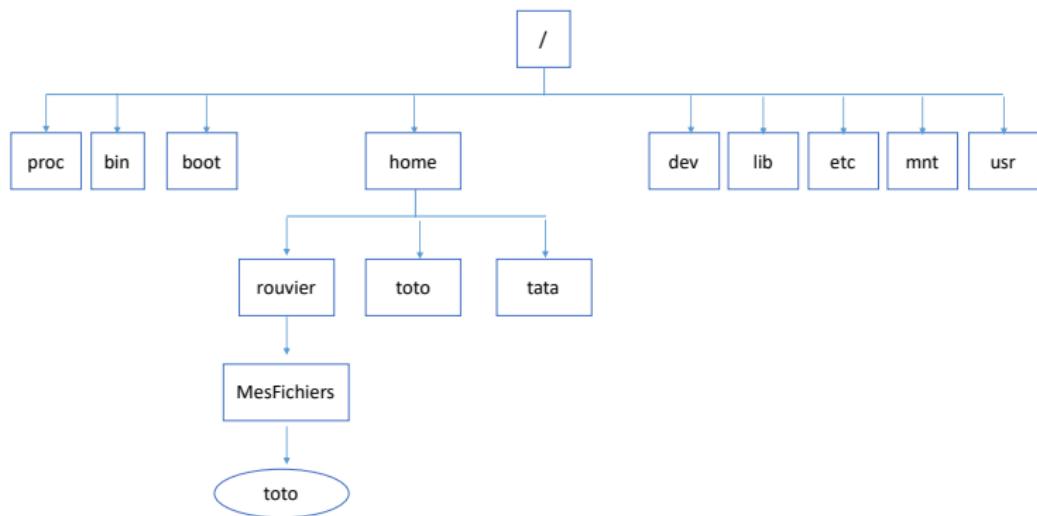


# Vocabulaire : Noms des répertoires spéciaux

- Vocabulaire relatif aux répertoires
  - répertoire **d'accueil** ou **home**
    - répertoire sur lequel on est positionné à la connexion
    - symbole ~
  - répertoire **courant** ou **working directory**
    - répertoire sur lequel on se trouve à tout moment : symbole .
  - répertoire **père**
    - répertoire au dessus du répertoire courant dans l'arborescence Unix
    - symbole ..



# Vocabulaire : Chemins d'accès à un fichier



- chemin absolu : **/home/rouvier/MesFichiers/toto**
- chemin relatif au répertoire d'accueil : **~/MesFichiers/toto**
- chemin relatif au répertoire courant : **./toto** ou **toto**

# Commandes sur les répertoires

Les commandes sur le répertoires :

- **cd** : **c**hange **d**irectory
- **mkdir** : **m**ake **d**irectory
- **rmdir** : **r**emove **d**irectory
- **pwd** : **p**rint **w**orking **d**irectory

# Change Directory

La commande **cd** permet de changer de répertoire au sein de l'arborescence du système :

**Pour aller dans le répertoire /etc :**

```
invite> cd /etc
```

```
invite> cd /etc/var/conf
```

**Pour se diriger vers le répertoire supérieur (parent) :**

```
invite> cd ..
```

```
invite> cd ../dir1/dir2
```

**Pour se diriger vers le répertoire d'accueil :**

```
invite> cd ~
```

```
invite> cd ~/dir1
```

# Make Directory

La commande **mkdir** permet de créer des répertoires :

**Exemple de création d'un répertoire avec mkdir :**

```
invite> mkdir nom_repertoire
```

```
invite> mkdir /etc/nom_repertoire
```

```
invite> mkdir ../nom_repertoire
```

**Exemple de création de plusieurs répertoires successifs :**

- -p : permet de créer des répertoires successifs

```
invite> mkdir -p repertoire1/repertoire2
```

**Exemple pour afficher un message à chaque création de répertoire :**

- -v : affiche un message à chaque création de répertoire

```
invite> mkdir -v repertoire1
mkdir: création du répertoire 'repertoire1'
```

# Remove Directory

La commande **rmdir** permet de supprimer des répertoires :

## Exemple de suppression d'un répertoire avec rmdir :

```
invite> mkdir nom_repertoire  
invite> rmdir nom_repertoire
```

**Attention** : le répertoire que vous souhaitez supprimer doit impérativement être vide.

```
invite> rmdir /etc  
rmdir: échec de suppression de « /etc »: Le dossier n'est pas  
vide
```

**Attention** : pour supprimer un répertoire non vide

```
invite> rm -rf nom_repertoire
```

# Print Working Directory

La commande **pwd** permet d'afficher le répertoire où vous vous trouvez :

**Exemple :**

```
invite/> pwd  
/home/root  
invite/> cd ..  
invite/> pwd  
/home
```

# Identification des fichiers

- Nom d'un fichier : préfixe.suffixe ou base.extension
  - Base/Préfixe : nom
  - Extension/Suffixe : type
- Exemples types fichiers :
  - **.txt** : fichier texte
  - **.sh** : scripts shell
  - **.c, .sh** : programmes C
  - pas d'extension : programme exécutable

# Identification des fichiers - fenêtrage

- Avec un système de fenêtrage / icônes



# Identification des fichiers : ligne de commande – Partie 1

```
invite/> ls
Archive.zip
Archive_cnn.zip
Chess-Engine
test.dic
toto.py
```

# Identification des fichiers : ligne de commande – Partie 2

- ls [-algiART...] [name] ...
- Options :
  - -a : all (même commençant par un .)
  - -l : format long
  - -c ou -t : tri par dernière date de modification
  - -R : récursif (permet d'afficher tout une arborescence)
- Exemple :

```
invite> ls -l
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Archive.zip
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Archive_cnn.zip
drw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 Chess-Engine
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 test.dic
-rw-r--r--@ 1 rouvier staff 3613 12 nov 08:02 toto.py
```

# Copie fichiers/répertoires

La commande **cp** permet de copier un fichier/un répertoire :

- Recopie de fichiers - recopie du fichier filename1 dans le fichier filename2

```
invite/> cp filename1.txt filename2.txt
```

- Copie de répertoire - copie récursive de dirname1 dans dirname2

```
invite/> cp -r dirname1/ dirname2/
```

- Copie de fichiers dans un répertoire - copie des fichiers filenames dans dirname

```
invite/> cp -r dirname1/filename.txt dirname/
```

# Déplacement de fichiers/répertoires

La commande **mv** permet de déplacer ou renommer un fichier/un répertoire :

- Renomme le fichier filename1 en filename2

```
invite/> mv filename1 filename2
```

- Renomme le fichier filename1 en filename2

```
invite/> mv dirname1 dirname2
```

- Place les fichiers filename dans le répertoire dirname

```
invite/> mv filename1 filename2 filename3 dirname
```

# Destruction de fichiers/répertoires

- rm [-fir] filename ....
  - -r : destruction récursive
  - -i : mode interactif
  - -f : force

```
invite/> rm filename1
```

```
invite/> rm -rf dirname1
```

# Globbing - Partie 1

- **Problématique** : comment exécuter une commande sur un sous-ensemble de fichier ?
  - Comment afficher uniquement les informations sur toutes les images d'un répertoire ?
  - Comment effacer uniquement les fichiers textes d'un répertoire ?
  - ....
- Le **globbing** permet de sélectionner un ensemble de noms de fichiers/dossiers en utilisant des caractères génériques (*wildcards*).
- Le globbing peut être utilisé sur toutes les commandes Linux : *ls, rm, cat, find, head, tail...*
- Syntaxe :

Wildcard	Description	Exemple	Matches	Not Match
*	Match pour tous les nombres, caractères	Law*	Law, Laws où Lawyer	GrokLaw, La où aw
*	Match pour tous les nombres, caractères	*Law*	Law, GrokLaw où Lawyer	La où aw
?	Match un seul caractères	?at	Cat, cat, Bat où bat	at
[abc]	Match un seul caractère donné dans les brackets	[CB]at	Cat où Bat	cat où bat
[a-z]	Match un caractère de la plage indiquée dans les brackets	Letter[0-9]	Letter0, Letter1 jusqu'à Letter9	Letters, Letter où Letter10

# Globbing - Partie 2

Quelques exemples en utilisant la commande ls :

```
invite/> ls f*
invite/> ls fil*
invite/> ls *.jpg
invite/> ls *.txt
invite/> ls [cb]at*
```

Quelques exemples en utilisant la commande rm :

```
invite/> rm -rf f*
invite/> rm -rf fil*
invite/> rm -rf *.jpg
invite/> rm -rf *.txt
invite/> rm -rf [cb]at*
```