

Linux - Fondamentaux

m2iformation.fr



Programme

- Introduction
 - Origines de GNU/Linux.
 - Définitions du logiciel libre et open source.
 - Description des organisations concernées.
 - Aperçu des différents systèmes d'exploitation et solutions open source.
- Architecture du système
 - Notions de base de l'architecture.
 - Caractéristiques générales des couches du système.
 - Couches graphiques.
 - Aperçu des différents Shells.
 - Présentation des principales distributions.
 - Choix et installation d'une distribution
- Système de fichiers
 - Hiérarchie du système de fichiers.
 - Différents types de systèmes de fichiers.
 - Commandes utiles et interaction.
- Premiers pas
 - Introduction au Shell et à l'environnement.
 - Introduction à l'interface graphique.
 - Utilisation des terminaux et des applications.
 - Navigation et interaction avec les fichiers/dossiers.
 - Gestion des comptes utilisateurs et administrateurs.

Programme

- Gestion des fichiers
 - Commandes pour gérer les dossiers.
 - Lecture et interaction avec le contenu des fichiers.
 - Commandes pour gérer les fichiers.
 - Gestion des alias et des liens symboliques ou physiques.
- Scripting Shell/Bash
 - Bases du Shell et instructions.
 - Commandes principales :
 - Recherche, capture, création.
 - Utilisation de l'aide et de l'historique.
 - Gestion des variables Shell et exportation.
 - Capture des résultats de commandes.
 - Echappement des caractères et protection.
 - Utilisation des pipes
- Réseaux
 - Notions de base du réseau.
 - Configuration réseau.
 - Transfert de fichiers et connexions distantes.
- Scripting et redirections
 - Introduction aux instructions et boucles.
 - Introduction au scripting.
 - Aperçu des flux standard (stdin, stdout, stderr).



Introduction

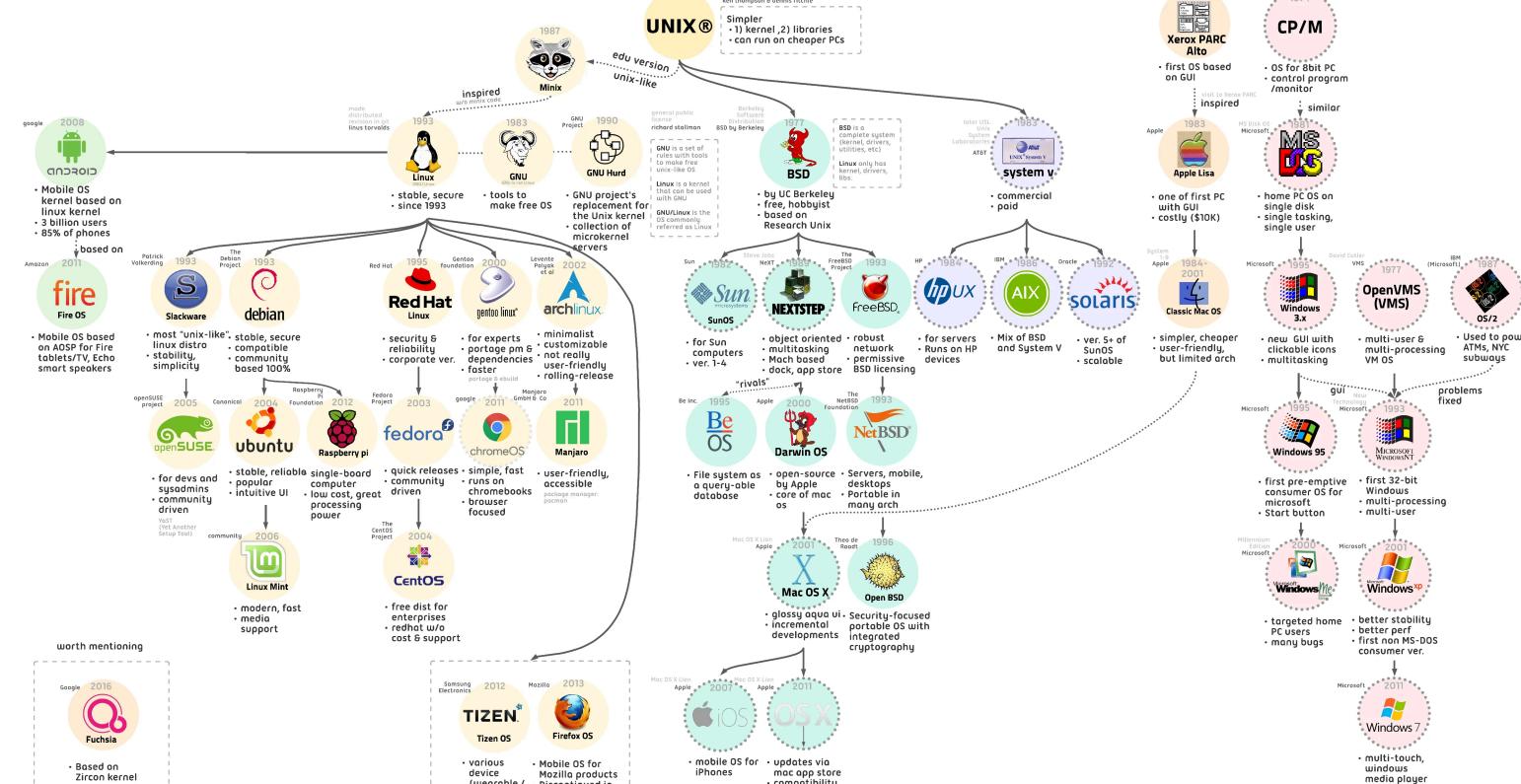
Origines de GNU/Linux

SIMPLE HISTORY OF OS

for any corrections, please contact
github: @chococigar
12 / 2020

Note that this diagram is **very simplified**.

• dotted outline: proprietary



L'histoire simple de Linux

◆ 1. Les racines – De MULTICS à UNIX

- Dans les **années 1960**, le projet MULTICS (Service informatique multiplexé) a inspiré le besoin de systèmes d'exploitation plus simples et efficaces.
- Cela a mené à la création de **UNIX** aux laboratoires Bell en **1969**, qui est devenu la base de nombreux futurs systèmes d'exploitation.

◆ 2. MINIX – Un UNIX éducatif

- En **1987**, **MINIX** est créé comme un petit système de type UNIX à but pédagogique par Andrew Tanenbaum.
- MINIX a inspiré **Linus Torvalds**, qui voulait concevoir son propre noyau.

3. Naissance du noyau Linux (1991)

- En **1991**, Linus Torvalds publie le **noyau Linux** en tant que projet personnel.
- Il l'a combiné aux outils du **Projet GNU** (lancé en 1983 par Richard Stallman) pour créer un système complet **libre et open source**.
- Cette combinaison est souvent appelée **GNU/Linux**.

γ 4. L'essor des distributions Linux

À partir du noyau, différentes communautés et entreprises ont créé des **distributions Linux** pour divers usages :

* **Slackware (1993)**

- Une des plus anciennes distributions.
- Ciblait la simplicité et le minimalisme.
- A inspiré de nombreuses autres.

📦 **Debian (1993)**

- Portée par la communauté, très stable.
- 100 % logiciel libre par philosophie.
- Est devenue la **base de nombreuses autres distributions** (Ubuntu, Kali, etc.).

📦 **Red Hat (1995)**

- Axée sur l'entreprise, avec support commercial.

5. Ramifications – Distributions populaires et leurs buts

Ubuntu (2004)

- Basée sur Debian.
- Crée pour être **conviviale**, avec des sorties régulières et un support à long terme (LTS).
- Conçue pour les ordinateurs personnels et les serveurs.

Fedora

- Parrainée par Red Hat.
- Met l'accent sur **l'innovation**, les nouvelles fonctionnalités.
- Sert de laboratoire pour **Red Hat Enterprise Linux (RHEL)**.

● CentOS

- Reprise gratuite de RHEL.
- Pensée pour les environnements de production **sans abonnement** Red Hat.

■ Linux Mint

- Basée sur Ubuntu.
- Conçue pour ressembler à Windows – **idéale pour les débutants**.

■ Arch Linux

- Pour utilisateurs avancés.
- **Publication continue**, minimale par défaut, très personnalisable.

Kali Linux

- Basée sur Debian.
- Conçue pour les **experts en sécurité et le pentesting**.

6. Alpine Linux

- Peu détaillée ici, mais Alpine fait partie des distributions axées sur le **minimalisme**.
- Très utilisée dans les conteneurs (Docker).
- **Extrêmement légère et sécurisée**.

7. Linux au-delà du bureau

Linux alimente bien plus que des ordinateurs :

Android

- Basé sur le **noyau Linux**, Android est le système mobile le plus utilisé au monde.
- Plus d'un milliard d'appareils tournent sous Android.

Fire OS

- Dérivé d'Android utilisé sur les appareils Amazon (Fire TV, Echo Show, etc.).

Tizen / Firefox OS

- Systèmes basés sur Linux pour l'IoT, les téléviseurs intelligents, les systèmes embarqués.

 **Résumé : Pourquoi Linux est important**

Aspect	Description
Open source	Tout le monde peut consulter, modifier et redistribuer le code
Flexible	Utilisé sur ordinateurs, serveurs, smartphones, routeurs, conteneurs, superordinateurs
Diversifié	De nombreuses distributions adaptées à chaque besoin
Communautaire	Fort soutien via des forums, communautés et développement collaboratif

🔍 Définitions du logiciel libre et open source

🧠 Pourquoi c'est important

Quand on parle de **Linux**, on parle en réalité de **logiciels libres et open source** (FOSS).

Mais les termes « *logiciel libre* » et « *open source* » ne sont **pas synonymes** :

Ils décrivent souvent le **même logiciel**, mais proviennent de **philosophies différentes**.

● Qu'est-ce qu'un logiciel libre ? (selon la Free Software Foundation)

Le **logiciel libre** signifie **liberté**, pas gratuité.

La **Free Software Foundation (FSF)** définit le logiciel libre selon **quatre libertés fondamentales** :

1. **Liberté d'exécuter** le programme pour n'importe quel usage.
2. **Liberté d'étudier** comment le programme fonctionne et de le modifier à volonté (*accès au code source requis*).
3. **Liberté de redistribuer** des copies du logiciel.
4. **Liberté de distribuer** des versions modifiées du programme.

➔ **Exemple** : Les outils GNU utilisés dans Linux (`bash`, `gcc`, `make`, etc.) sont tous des **logiciels libres**.

💡 **Important** : “Libre” ici ne veut pas dire “gratuit” – ça veut dire **libre comme dans liberté d'expression**, pas comme “bière gratuite”.

● Qu'est-ce que l'open source ? (selon l'Open Source Initiative)

Le terme **open source** met davantage l'accent sur les **avantages pratiques** du partage de code, comme :

- Développement plus rapide
- Meilleure collaboration
- Transparence du code

L'**Open Source Initiative (OSI)** a créé la **Définition de l'Open Source** avec 10 critères, comme :

- Redistribution gratuite
- Disponibilité du code source
- Pas de discrimination selon l'usage ou les utilisateurs

➡ **Exemples** : Des projets comme **Firefox**, **Kubernetes**, ou **VS Code** sont open source et suivent ces règles.

● Différence principale : la philosophie

Aspect	Logiciel libre	Open source
Priorité	Éthique et liberté des utilisateurs	Avantages pratiques et techniques
Fondateur	Richard Stallman (FSF)	Eric Raymond, Bruce Perens (OSI)
Expression	“Libre comme la liberté”	“Code ouvert à l’inspection et la contribution”
Vision	“Vous devez contrôler le logiciel”	“Le code ouvert permet un meilleur logiciel”

Bien que cela s'applique **au même type de logiciels**, l'**intention philosophique** est différente.

🔧 Comment ça se relie à Linux

- Le **noyau Linux** est sous licence **GNU GPL (General Public License)**, qui est une **licence de logiciel libre**.
- Linux est donc à la fois **libre ET open source**.
- Mais selon les personnes qui en font la promotion :
 - Certaines insistent sur la **liberté** (FSF).
 - D'autres sur la **collaboration et l'innovation** (OSI).

● 1. Free Software Foundation (FSF)

📌 Qui sont-ils :

- Fondée en **1985** par **Richard Stallman**
- Organisation à but non lucratif basée aux États-Unis
- Créeée pour **défendre la liberté des utilisateurs**

🎯 Mission :

- Protéger et promouvoir les **libertés des utilisateurs de logiciels**
- Développer et maintenir le **Projet GNU**
- Éduquer le public sur l'importance de la liberté logicielle

🛠️ Ce qu'ils font :

- Créateurs de la **GNU GPL (General Public License)**
- Soutiennent des outils comme **Bash, GCC, GIMP**, etc.
- Mènent des campagnes comme “Defective by Design” contre les DRM

🌐 Lien avec Linux :

- Les **outils GNU** de la FSF sont ce qui rend Linux réellement utilisable (GNU/Linux).
- La FSF veille à ce que Linux reste **libre comme la liberté**.

● 2. Open Source Initiative (OSI)

📌 Qui sont-ils :

- Fondée en **1998** par **Eric S. Raymond** et **Bruce Perens**
- Organisation à but non lucratif dédiée à la **définition et promotion du code open source**

🎯 Mission :

- Fournir une définition **claire et pratique** de l'open source
- Maintenir une liste de **licences validées par l'OSI**
- Encourager les entreprises et développeurs à adopter les pratiques open source

🛠️ Ce qu'ils font :

- Maintiennent la **Définition de l'Open Source**
- Certifient des licences comme **MIT, Apache, BSD, Mozilla Public License**

🌐 Lien avec Linux :

- Le noyau Linux utilise une **licence GPL (FSF)**, mais beaucoup d'outils autour suivent des licences **validées par l'OSI**.
- L'OSI facilite l'adoption du libre dans le monde **professionnel et académique**.

● 3. The Linux Foundation

📌 Qui sont-ils :

- Crée en **2000** (anciennement Open Source Development Labs)
- Devient **The Linux Foundation** en 2007
- Consortium de grandes entreprises : Intel, IBM, Google, Microsoft...

🎯 Mission :

- Promouvoir la croissance de **Linux et des logiciels collaboratifs**
- **Financer et soutenir Linus Torvalds** (créateur du noyau Linux)
- Héberger des projets open source majeurs (Kubernetes, Node.js, Hyperledger)

🛠️ Ce qu'ils font :

- Proposent des **certifications** (LFCS, LFCE)
- Organisent des conférences comme **Open Source Summit**
- Gèrent l'**infrastructure technique** de projets open source majeurs

🌐 Lien avec Linux :

- La Linux Foundation joue un rôle clé dans le **développement du noyau**.
- Elle fait le **pont entre la communauté et l'industrie**.

4. The Debian Project

📌 Qui sont-ils :

- Fondé en **1993** par **Ian Murdock**
- Projet communautaire qui développe **Debian GNU/Linux**

🎯 Mission :

- Créer un système d'exploitation **libre, stable et universel**
- Respecter le **Contrat social Debian** et les **lignes directrices du logiciel libre Debian (DFSG)**

🛠️ Ce qu'ils font :

- Développent Debian – la base d'Ubuntu, Kali, etc.
- Fonctionnement transparent et **gouvernance démocratique**

🌐 Lien avec Linux :

- Debian est une **distribution phare** dans l'écosystème Linux
- Ses principes communautaires influencent de nombreuses décisions autour de Linux

Tableau récapitulatif

Organisation	Année	Rôle principal	Contribution clé
FSF	1985	Promouvoir la liberté logicielle	Outils GNU, licence GPL
OSI	1998	Définir l'open source, certifier les licences	Définition OSI, validation de licences
Linux Foundation	2000	Soutenir Linux et les projets liés	Financement du noyau, certifications
Projet Debian	1993	Distribution Linux communautaire	Debian OS, contrat social



Architecture Système

◆ 1. Bases de l'architecture

L'architecture système désigne la **structure globale** d'un système informatique : comment ses composants – **matériels et logiciels** – sont **organisés et interagissent**. Dans un système GNU/Linux, l'architecture comprend généralement :

1. Matériel

- Composants physiques : CPU, RAM, disque, cartes réseau.

2. Noyau (kernel)

- Le **cœur** du système. Gère la mémoire, les processus, les périphériques.

3. Bibliothèques système

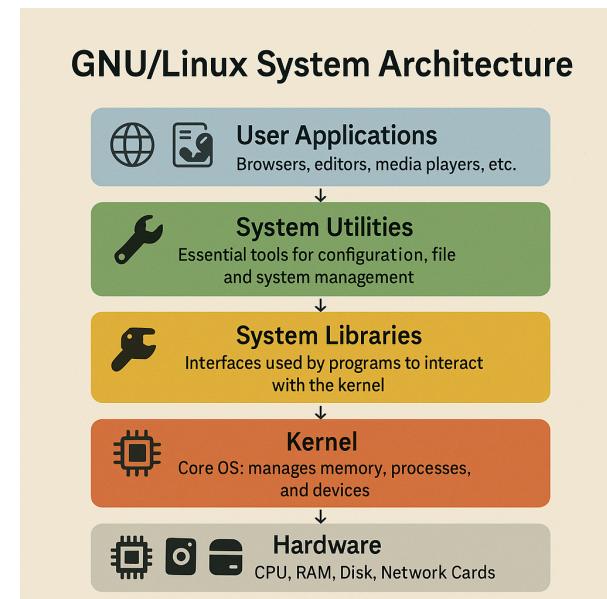
- Outils utilisés par les logiciels pour interagir avec le noyau.

4. Utilitaires système

- Programmes essentiels pour la configuration, la gestion de fichiers, etc.

5. Applications utilisateur

- Logiciels comme navigateurs, éditeurs de texte, lecteurs multimédias.



◆ 1. Bases de l'architecture

Quand vous ouvrez Firefox :

- L'application utilise des bibliothèques système (comme GTK).
- Ces bibliothèques font appel au **noyau** via des appels système.
- Le noyau accède au **matériel** (écran, réseau, mémoire).

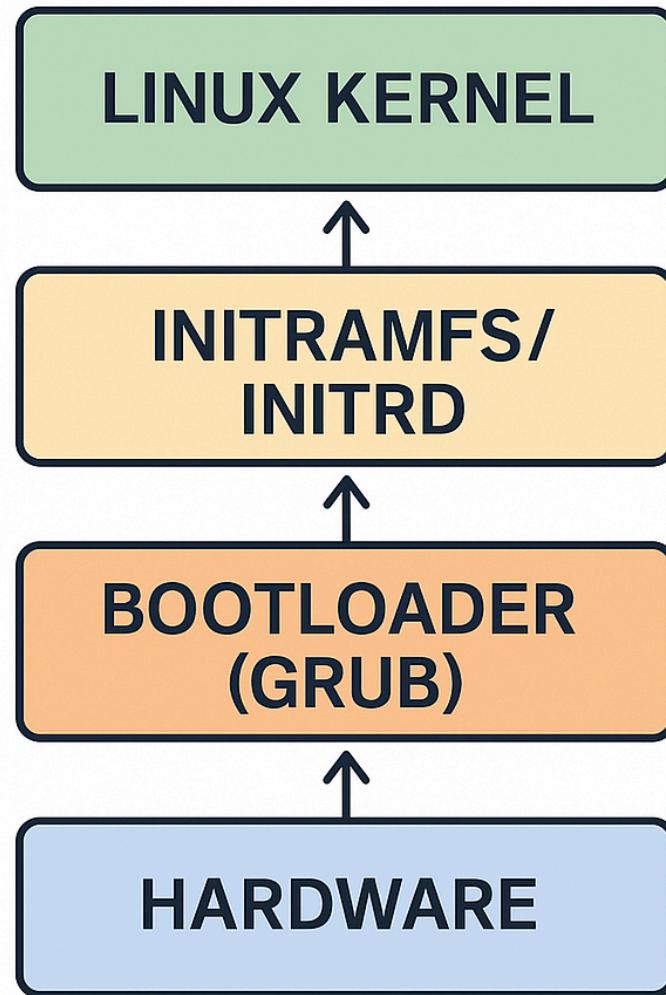
◆ 2. Caractéristiques générales des couches système

Pensez au système comme une **structure en couches** – chaque couche dépend de celle du dessous.

Couche	Rôle
Matériel	Machine physique
Noyau (Kernel)	Gère le matériel et les appels système
Shell / CLI	Interface utilisateur pour interagir avec le noyau
Utilitaires	Outils en ligne de commande ou graphiques
Applications	Logiciels pour l'utilisateur (LibreOffice, Firefox, etc.)

Chaque couche **abstrait** la précédente – ce qui simplifie l'utilisation sans accès direct au matériel.

📘 Chaîne complète entre matériel et noyau Linux



🧱 1. Matériel

Définition :

Les **composants physiques** de l'ordinateur – CPU, RAM, disque dur, SSD, GPU, carte réseau, etc.

C'est la **couche la plus basse**, et elle ne peut rien faire seule. Elle a besoin du **firmware** pour fonctionner.

🧠 2. Firmware : BIOS / UEFI

Définition :

Le firmware est le premier logiciel lancé au démarrage de l'ordinateur.

◆ **BIOS (Basic Input/Output System)**

- Ancien firmware, sur les systèmes plus vieux
- Stocké sur la carte mère
- Utilise le MBR (Master Boot Record)
- Lance le chargeur d'amorçage (bootloader)

◆ UEFI (Unified Extensible Firmware Interface)

- Remplace modernement le BIOS
- Gère le secure boot, GPT, interface graphique
- Lance le bootloader depuis la partition EFI (`/boot/efi`)
- Plus puissant et flexible que le BIOS

Rôle :

- Initialise le matériel (POST)
- Cherche un périphérique de démarrage valide
- Lance le **chargeur d'amorçage**

⚙ 3. Chargeur d'amorçage (GRUB)

Définition :

Petit programme qui **démarre le système d'exploitation**.

◆ GRUB (GRand Unified Bootloader)

- Bootloader par défaut sur la plupart des distributions Linux
- Permet :
 - Le multiboot
 - Des paramètres personnalisés du noyau
 - Des modes de secours

✓ Rôle :

- Charge l'image du noyau (`vmlinuz`, `bzImage`)
- Charge `initramfs` si présent
- Passe des **paramètres** au noyau (`root=`, `quiet`, etc.)

4. Initrd / Initramfs

Définition :

Un **système de fichiers temporaire** chargé en **RAM** avant de monter le vrai système (/).

- `initrd` = ancien format
- `initramfs` = moderne et plus flexible

✓ Rôle :

- Contient :
 - Pilotes pour disques et systèmes de fichiers
 - Outils pour LVM, chiffrement, RAID
 - Scripts pour monter la racine réelle

- Prépare l'environnement pour le noyau

💡 Sans `initramfs`, le noyau peut ne pas reconnaître vos disques.

🔧 Voir ou reconstruire :

```
ls /boot/init*
sudo update-initramfs -u
```

5. Le noyau Linux

Définition :

Le **cœur** du système. Il est maintenant entièrement chargé en mémoire et opérationnel.

Rôle :

- Gère :
 - La mémoire
 - Le processeur
 - Les systèmes de fichiers
 - Le réseau
 - Les pilotes
- Démarre le **processus PID 1** (souvent `systemd`)

C'est le noyau qui **lance le premier processus utilisateur**.

◆ 3. Couches graphiques (pile d'affichage)

Linux peut fonctionner uniquement en **ligne de commande**, mais beaucoup installent une **interface graphique**.

Composants de la pile graphique :

Composant	Description
X Window System (X11)	Ancien serveur graphique
Wayland	Remplaçant moderne de X11 (plus rapide/sécurisé)
Display Manager	Écran de connexion (GDM, LightDM, SDDM)
Environnement de bureau	KDE, GNOME, XFCE

Exemple :

- GNOME utilise Wayland ou X11.
- KDE Plasma aussi.
- Ces composants **dialoguent avec le noyau** via les pilotes pour afficher à l'écran.

◆ 4. Présentation des principales distributions

Les systèmes GNU/Linux existent sous **plusieurs versions**, appelées **distributions**.

Elles combinent :

- Le noyau Linux
- Des outils GNU
- Un gestionnaire de paquets
- Un environnement graphique ou minimal

Distribution	Public cible / Caractéristiques
Ubuntu	Conviviale, idéale pour les débutants
Debian	Stabilité, fiabilité pour serveurs
Fedora	Moderne, pour développeurs
Arch Linux	Publication continue, pour utilisateurs avancés
Linux Mint	Similaire à Windows, très accessible
Kali Linux	Sécurité, pentest, hacking éthique
Alpine Linux	Minimaliste, pour conteneurs et systèmes embarqués

◆ 5. Choisir et installer une distribution

✓ Comment choisir sa distribution :

Posez-vous les bonnes questions :

- Ai-je besoin de **stabilité** ou de **fonctionnalités récentes** ?
- Suis-je **débutant** ou **avancé** ?
- Est-ce pour un **bureau**, un **serveur**, un **cloud** ?
- Qu'est-ce qui compte le plus ? **Vie privée, performance, outils sécurité** ?

Cas d'usage	Distribution recommandée
PC de bureau (débutant)	Ubuntu, Linux Mint
Serveur	Debian, CentOS, Ubuntu Server
Tests sécurité	Kali Linux
Cloud / Docker	Alpine Linux
Configuration sur mesure	Arch Linux



Installation

⚙ Étapes d'installation (général)

1. **Téléchargez l'ISO** depuis le site de la distribution.
2. **Créez une clé USB bootable** avec des outils comme Rufus ou BalenaEtcher.
3. **Démarrez depuis la clé USB** et lancez l'installateur en live.
4. Choisissez :
 - Langue
 - Disposition du clavier
 - Fuseau horaire
 - Partitionnement du disque
 - Nom d'utilisateur et mot de passe
5. **Installez et redémarrez !**

Méthodes avancées d'installation de Linux

1. Installation automatisée (Non supervisée / Preseed / Kickstart)

Cette méthode vous permet **d'automatiser tout le processus d'installation** à l'aide d'un fichier de configuration – plus besoin de cliquer manuellement dans les menus.

Méthode	Description
Preseed (Debian/Ubuntu)	Utilise un fichier <code>.cfg</code> contenant les réponses aux questions de l'installateur. Souvent utilisé avec PXE ou une ISO personnalisée.
Kickstart (Red Hat/CentOS/Fedora)	Fichier <code>.ks</code> pour automatiser les installations basées sur RHEL.
Autoinstall (Ubuntu 20.04+)	Remplaçant basé sur YAML pour Preseed. Fonctionne avec cloud-init.
YaST AutoYaST (SUSE)	Utilise des profils XML pour automatiser les installations SUSE.

🌐 2. Démarrage réseau (PXE + TFTP + NFS ou HTTP)

Lancez l'installation de Linux **via le réseau** – pas besoin de clé USB, CD ou ISO.

✓ Utile pour :

- Installer Linux sur plusieurs machines en même temps
- Ordinateurs anciens sans support USB/DVD

1. La machine cible envoie une requête PXE via le réseau local
2. Un **serveur PXE (TFTP)** fournit les fichiers de démarrage (GRUB, noyau...)
3. Un serveur NFS ou HTTP fournit l'installateur et/ou l'ISO
4. La machine démarre l'installateur depuis le réseau

🔧 Composants requis :

- Serveur DHCP
- Serveur TFTP
- Serveur HTTP ou NFS pour les médias d'installation
- Optionnel : fichier Kickstart ou Preseed pour automatiser

3. ISO personnalisée / Linux remastérisé

Vous pouvez **créer une ISO Linux personnalisée** contenant déjà :

- Vos configurations
 - Des logiciels préinstallés
 - Des utilisateurs définis
 - Votre identité visuelle
-  Idéal pour :
- Les entreprises distribuant leur propre distro interne
 - Les environnements de formation ou de labo
 - Les ISO de secours et de récupération

✿ Outils pour créer des ISOs personnalisées :

Outil	Description
Cubic (Ubuntu)	Interface graphique pour personnaliser des ISOs Ubuntu
live-build (Debian)	Créez votre propre ISO Debian depuis zéro
Linux Live Kit	Transforme un système existant en ISO bootable

4. Modèles de machines virtuelles (OVA, VDI, QCOW2, etc.)

Au lieu d'installer Linux à chaque fois, vous pouvez **utiliser ou créer des images de machines virtuelles** prêtes à l'emploi.

Utile pour :

- Développeurs et formateurs
- Étudiants dans des environnements isolés
- Tests automatisés (CI/CD)

🔧 **Formats courants :**

Format	Utilisé par
.vdi	VirtualBox
.vmdk	VMware
.qcow2	KVM, QEMU
.ova	Open Virtual Appliance

💡 Astuce : vous pouvez exporter une VM Linux configurée et permettre à d'autres de l'importer directement – plus besoin d'installation.

Cloud 5. Images cloud / Installation basée sur cloud-init

Utilisées sur les plateformes cloud (AWS, Azure, GCP, OpenStack), Linux y est installé à partir **d'images préconstruites**.

◆ Exemples :

- `ubuntu-22.04-server-cloudimg-amd64.img`
- `centos-stream-9-cloud.qcow2`

Cloud-init gère :

- L'ajout des clés SSH
- La création d'utilisateurs
- La configuration réseau
- Les scripts initiaux

Parfait pour :

- Provisionnement de serveurs
- Déploiements Infrastructure-as-Code
- Pipelines DevOps

◆ 6. Aperçu des différents shells

Un **shell** est une interface qui permet d'interagir avec le système d'exploitation – généralement via une **interface en ligne de commande (CLI)**.

💡 Shells courants sous Linux :

Shell	Description
bash	Bourne Again Shell – défaut sur la plupart des systèmes 
sh	Shell Bourne original – très basique
zsh	Comme bash mais plus puissant/flexible
fish	Shell interactif convivial – moderne et facile
csh/tcsh	Variantes de C Shell – syntaxe façon C

◆ 1. sh - Bourne Shell

- Le **shell Unix d'origine**, créé dans les années 1970 par Stephen Bourne.
- Toujours présent sur tous les systèmes UNIX et Linux sous `/bin/sh`.
-  **Fonctionnalités :**
 - Syntaxe très basique.
 - Pas d'auto-complétion.
 - Pas d'historique de commandes.
-  **Quand l'utiliser :**
 - Scripts anciens qui nécessitent la conformité POSIX.
 - Pour écrire des scripts très portables.
-  **Pourquoi :**
 - Minimal et **pris en charge partout** – idéal pour du scripting bas niveau.

◆ 2. **bash** - Bourne Again Shell

- Développé dans le cadre du projet GNU.
- **Shell par défaut** sur la plupart des distributions Linux.
- **🔧 Fonctionnalités :**
 - Historique des commandes (**Flèches Haut/Bas**)
 - Auto-complétion par tabulation
 - Édition en ligne de commande
 - Fonctions de script : fonctions, boucles, tableaux
 - Calculs et variables intégrés
- **📌 Quand l'utiliser :**
 - **Scripting général**
 - Administration système quotidienne
 - La plupart des tutoriels supposent l'usage de **bash**
- **✓ Pourquoi :**
 - Puissant, très supporté, et **compatible avec sh**

◆ 3. zsh - Z Shell

- Un shell **riche en fonctionnalités**, conçu pour étendre `bash`.
- Très populaire chez les développeurs.
-  **Fonctionnalités :**
 - Tout ce que fait `bash` +
 - **Auto-complétion avancée**
 - **Correction orthographique**
 - **Plugins et thèmes** (Oh My Zsh)
 - Globbing étendu (`**`, `**/*.txt`)
 - Invite personnalisable
 - Partage de l'historique entre sessions
-  **Quand l'utiliser :**
 - Pour les **power users ou développeurs** qui veulent booster leur terminal
 - Pour personnaliser son expérience CLI
-  **Pourquoi :**
 - Fluide, moderne, très agréable à utiliser avec Oh My Zsh

◆ 4. fish - Friendly Interactive Shell

- Un shell **moderne** axé sur la convivialité et l'ergonomie.

Fonctionnalités :

- **Coloration syntaxique** en temps réel
- **Suggestions automatiques** pendant la frappe
- **Aide intégrée**
- Pas besoin de `.bashrc` ou `.zshrc` (utilise des fonctions/variables universelles)
- Syntaxe de script simplifiée (**non compatible POSIX**)
-  **Quand l'utiliser :**
 - Pour **les débutants** ou ceux qui veulent un terminal agréable
 - Pour un usage interactif (pas pour le scripting)
-  **Pourquoi :**
 - Idéal pour débuter et être productif rapidement. “Ça fonctionne direct.”

◆ 5. `csh` et `tcsh` - C Shell

- `csh` a été conçu pour **ressembler au langage C**
- `tcsh` est une version améliorée de `csh`
-  **Fonctionnalités :**
 - Syntaxe façon C (`if (x) then`)
 - Historique de commandes et alias (dans `tcsh`)
 - Gestion des jobs
-  **Quand l'utiliser :**
 - Si vous travaillez dans un environnement qui utilise déjà ces scripts
 - Certains systèmes BSD utilisent encore `tcsh` par défaut
-  **Pourquoi :**
 - Pour des raisons **historiques** – déconseillé pour les scripts modernes

Tableau récapitulatif

Shell	POSIX ?	Fonctionnalités interactives	Compatible scripting ?	Cas d'usage idéal
sh	✓	✗ Aucune	✓ Très portable	Scripts anciens, portables
bash	✓	✓ Historique, autocomplétion	✓ Complet	Scripting général, terminal courant
zsh	✓	✓ Plugins, thèmes, correction	✓ Puissant	Dév, personnalisation
fish	✗	✓ Coloration, suggestions	✗ Limité	Usage interactif, débutants
tcsh	✗	✓ Syntaxe C, historique	✗ Limité	Systèmes BSD, environnements anciens

🔧 Quel shell choisir ?

Vous êtes...	Shell recommandé
Débutant	fish ou bash
Administrateur système	bash
Développeur	zsh + Oh My Zsh
Script portable	sh
Utilisateur de BSD ou systèmes anciens	tcsh



Système de fichiers

◆ 1. Hiérarchie du système de fichiers

La **hiérarchie du système de fichiers Linux** décrit **comment les fichiers et dossiers sont organisés**.

Tout commence à partir du **répertoire racine** : `/`

Ensuite, le système se déploie comme un **arbre**.

⌚ Répertoires principaux sous `/` :

Répertoire	Rôle
<code>/bin</code>	Commandes système essentielles (<code>ls</code> , <code>cp</code> , <code>mv</code> , etc.)
<code>/sbin</code>	Binaire système réservé aux administrateurs
<code>/etc</code>	Fichiers de configuration système et applications
<code>/home</code>	Dossiers personnels des utilisateurs (<code>/home/alice</code>)
<code>/var</code>	Données variables (logs, caches...)
<code>/usr</code>	Programmes utilisateur, bibliothèques (<code>/usr/bin</code> , <code>/usr/lib</code>)
<code>/lib</code>	Bibliothèques partagées essentielles au démarrage

Répertoire	Rôle
/tmp	Fichiers temporaires (vidés au redémarrage)
/boot	Fichiers de démarrage (noyau, GRUB)
/dev	Fichiers de périphériques (/dev/sda, /dev/null, etc.)
/mnt, /media	Points de montage pour supports externes (USB, CD-ROM)
/root	Dossier personnel de l'utilisateur root (admin)
/proc, /sys	Systèmes de fichiers virtuels d'information système

📌 **Tout est fichier sous Linux** – même les périphériques ou l'état du système.

◆ 2. Types de systèmes de fichiers

Linux peut utiliser **plusieurs systèmes de fichiers**, chacun ayant ses usages.

Système de fichiers	Description
ext4	Le plus courant sous Linux, rapide et stable ✓
FAT32	Compatible Windows / clés USB, mais limité à 4 Go/fichier
NTFS	Format Windows, lisible/écrivable sous Linux (avec pilotes)
XFS	Performant, adapté aux grandes structures
Btrfs	Moderne, avec snapshots et contrôles d'intégrité intégrés
tmpfs	Système temporaire en RAM (utilisé par <code>/tmp</code>)

▶ Par défaut, la plupart des distributions Linux utilisent **ext4**, sauf personnalisation.

◆ 3. Commandes utiles pour le système de fichiers

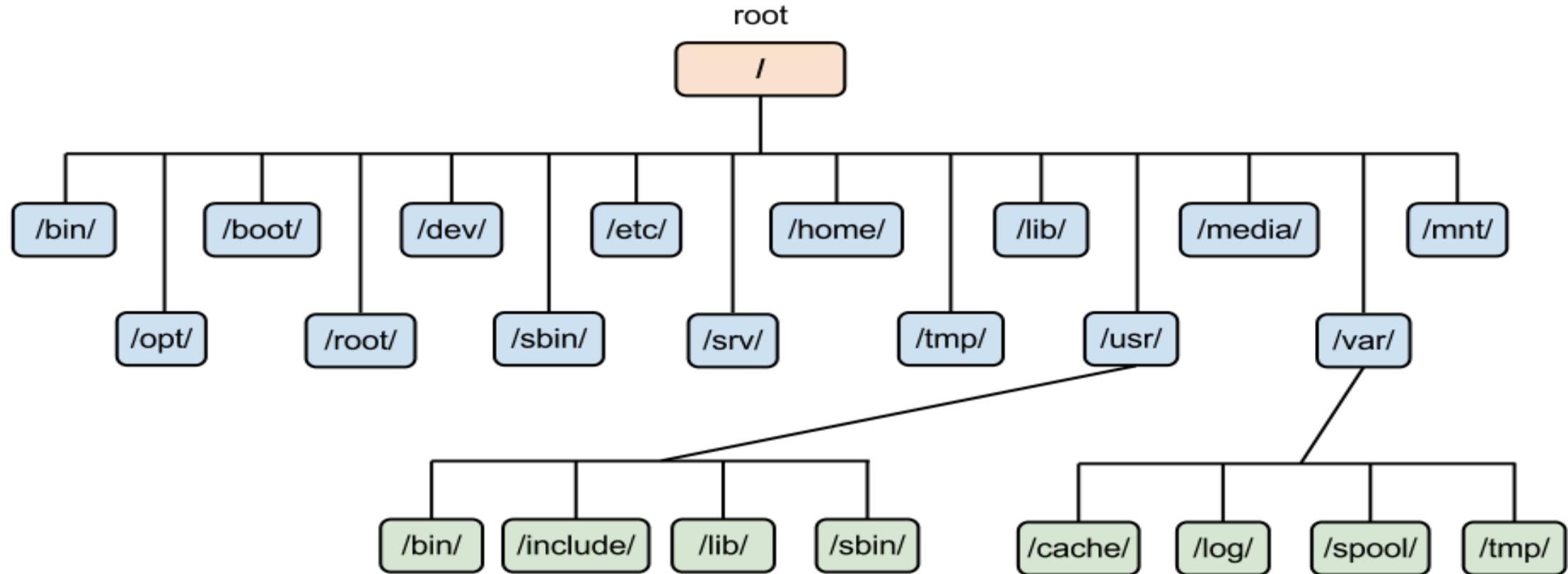
Voici les commandes clés pour interagir avec les fichiers et répertoires :

Commande	Rôle	Exemple
<code>ls</code>	Lister le contenu d'un dossier	<code>ls -l /etc</code>
<code>cd</code>	Changer de répertoire	<code>cd /home/utilisateur</code>
<code>pwd</code>	Afficher le chemin courant	<code>pwd</code>
<code>mkdir</code>	Créer un répertoire	<code>mkdir mondossier</code>
<code>rmdir / rm -r</code>	Supprimer un répertoire	<code>rm -r mondossier</code>
<code>touch</code>	Créer un fichier vide	<code>touch fichier.txt</code>
<code>cp</code>	Copier des fichiers	<code>cp fichier.txt copie.txt</code>
<code>mv</code>	Déplacer ou renommer un fichier	<code>mv fichier.txt /tmp/</code>
<code>rm</code>	Supprimer un fichier	<code>rm fichier.txt</code>

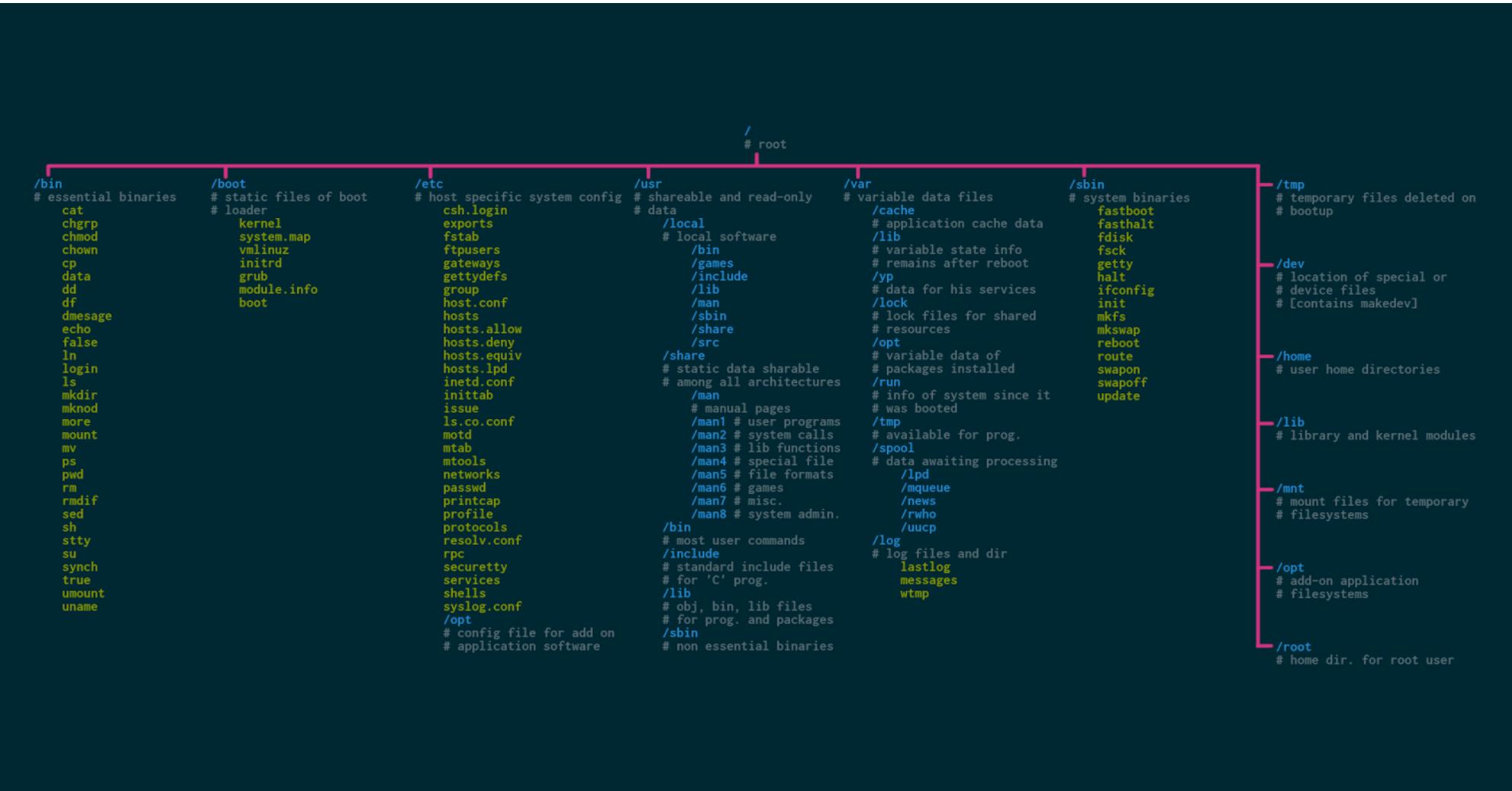
Commande	Rôle	Exemple
find, locate	Rechercher des fichiers	find / -name "* .log"
df -h	Afficher l'espace disque par partition	df -h
du -sh	Afficher la taille d'un dossier/fichier	du -sh Documents/

 On peut interagir avec le système de fichiers via le **terminal ou l'explorateur graphique**.

Le système de fichier avec Linux



Un peu plus détaillé





Prise en main

Utopios® Tous droits réservés

◆ 1. Introduction au Shell et à l'environnement

Un **shell** est un programme qui vous permet d'interagir avec le système en **tapant des commandes**. L'**environnement** est un ensemble de :

- Variables (`$PATH`, `$HOME`, etc.)
- Fichiers de configuration (`.bashrc`, `.zshrc`)
- Alias et fonctions personnalisées

 **Shells disponibles** : `bash`, `zsh`, `fish` (déjà vus précédemment)

Pour ouvrir un shell :

- Lancer le **Terminal** (interface graphique)
- Utiliser un **TTY** (Ctrl + Alt + F2)

◆ 2. Introduction à l'interface graphique

L'interface graphique (ou **GUI**) rend l'utilisation plus intuitive.

Un environnement graphique Linux comprend :

Composant	Rôle
Serveur d'affichage	Gère l'écran (X11 ou Wayland)
Gestionnaire de fenêtres	Contrôle la disposition et bordures
Environnement de bureau	Ensemble d'outils graphiques (GNOME, KDE, XFCE)
Gestionnaire de fichiers	Explorateur (Nautilus , Dolphin , etc.)

Utilisation :

- Ouvrir des applications
- Utiliser des menus, des fenêtres
- Configurer (écran, imprimante, wifi...)

◆ 3. Utiliser le terminal et les applications

Terminal

Le terminal est une interface **en ligne de commande**.

Exemples :

```
ls  
mkdir projet  
sudo apt install firefox
```

Applications

Lancement possible via :

- Le **menu** (Activités, Applications...)
- Une commande (`firefox &`)
- Un **raccourci sur le bureau**

◆ 4. Navigation et interaction avec fichiers/dossiers

Méthode graphique :

- Double-clic pour ouvrir un dossier
- Glisser-déposer
- Clic droit pour plus d'options

Méthode terminal :

```
cd ~/Documents  
ls -lh  
cp monfichier.txt /tmp/  
mv /tmp/monfichier.txt ~/Bureau/
```

Astuces :

-  = votre dossier personnel
-  = répertoire courant
-  = répertoire parent

Utilisez la **touche Tab** pour compléter automatiquement les noms.

◆ 5. Gérer les comptes utilisateurs et administrateurs

👤 Utilisateurs

Chaque utilisateur a un compte et un dossier perso (`/home/nom`).

Créer un utilisateur :

```
sudo adduser alice
```

🔒 Administrateur (root)

L'utilisateur `root` a tous les droits.

Exécuter une commande en tant que root :

```
sudo commande
```

Exemple :

```
sudo apt update  
sudo rm -rf /dossier/dangereux
```

◆ 5. Gérer les comptes (suite)

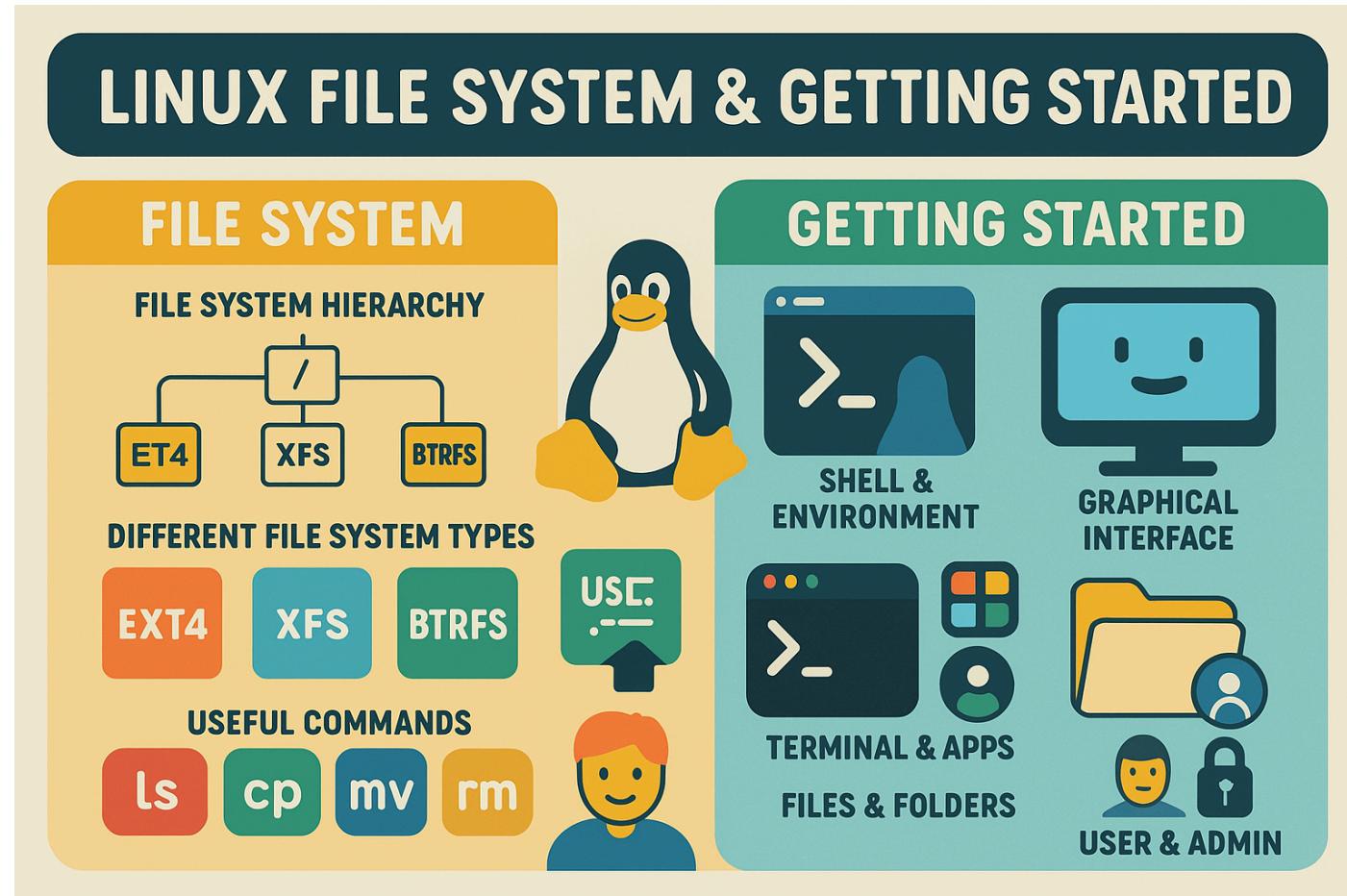
👤 Gestion des utilisateurs :

```
sudo useradd john  
sudo passwd john  
sudo deluser john
```

Ajouter à un groupe (ex. sudo) :

```
sudo usermod -aG sudo john
```

 Résumé





Gestion des fichiers

📁 1. Gestion des dossiers

🛠️ Créer des dossiers - `mkdir`

```
mkdir mondossier
```

Crée un dossier dans le répertoire actuel.

```
mkdir dossier1 dossier2 dossier3
```

Crée **plusieurs dossiers** d'un coup.

```
mkdir -p parent/enfant/petit-enfant
```

`-p` = crée **les dossiers parents si nécessaires**.

Parfait pour des scripts ou des structures imbriquées.

“  `mkdir` ne remplace pas les dossiers existants. Il affiche une erreur sauf si `-p` est utilisé. ”

📁 Afficher les dossiers - `ls` et `tree`

```
ls
```

Liste le contenu du dossier courant (hors fichiers cachés).

```
ls -l
```

Affichage en mode détaillé :

- Permissions (`rwx`)
- Nombre de liens
- Propriétaire / groupe
- Taille
- Date de modification

```
ls -a
```

Affiche **tous les fichiers**, y compris cachés (`.bashrc`, `.gitignore`)

```
ls -lh
```

- Affiche des tailles **lisibles** (Ko, Mo...).

```
ls -R
```

- Affichage **récursif** du contenu des sous-dossiers.

```
ls -la /etc
```

- Combine plusieurs options.

```
tree
```

- Affiche la structure comme un **arbre visuel**.

À installer : `sudo apt install tree`

⌚ Se déplacer - `cd`, `pwd`

```
cd /chemin/vers/dossier  
cd ~                      # Dossier personnel  
cd ..                     # Remonter d'un niveau  
cd -                      # Revenir au dossier précédent  
pwd                       # Afficher le chemin actuel
```

✎ Renommer / Déplacer - `mv`

```
mv ancien nom nouveau nom
```

Renomme un dossier.

```
mv dossier /tmp/
```

Déplace un dossier ailleurs.

Si le dossier cible existe → **fusion** du contenu.

Copier un dossier - `cp -r`

```
cp -r dossier1 dossier2
```

 `-r` = récursif (obligatoire pour les dossiers)

Optionnel :

```
cp -rv dossier1 dossier2
```

- `-v` = verbose (affiche les fichiers copiés)

✗ Supprimer un dossier - `rmdir`, `rm -r`

```
rmdir dossier
```

Supprime uniquement si vide.

```
rm -r dossier
```

Supprime un dossier et son contenu.

```
rm -rf dossier
```

- `-f` = force, **aucune confirmation**

⚠️ Dangereux ! À manipuler avec prudence.

2. Lire & interagir avec le contenu des fichiers

00 Lire un fichier texte

```
cat fichier.txt
```

Affiche tout le contenu (pour les petits fichiers).

```
less fichier.txt
```

Lecteur paginé (défilement, / pour chercher, q pour quitter).

```
head -n 20 fichier.txt
```

Affiche les **20 premières lignes** (par défaut = 10)

```
tail -n 15 fichier.txt
```

Affiche les **15 dernières lignes**

```
tail -f journal.log
```

Suivi en temps réel (pratique pour logs).

Ctrl + C pour quitter.

🔍 Rechercher dans un fichier - grep

```
grep "erreur" /var/log/syslog
```

- ✓ Affiche les lignes contenant "erreur"

```
grep -i "ERREUR" fichier.txt
```

- ✓ `-i` = insensible à la casse

```
grep -r "conflit" /etc
```

- ✓ `-r` = recherche récursive dans les sous-dossiers

```
grep -n "bonjour" fichier.txt
```

- ✓ `-n` = affiche le numéro de ligne

```
grep -v "DEBUG" fichier.txt
```

- ✓ `-v` = **exclut** les lignes contenant "DEBUG"

12 34 Compter lignes, mots, caractères – wc

```
wc fichier.txt
```

✓ Affiche :

- Nombre de **lignes**
- Nombre de **mots**
- Nombre de **caractères**

```
wc -l fichier.txt      # Juste les lignes
wc -w fichier.txt      # Juste les mots
wc -c fichier.txt      # Juste les caractères
```

 **Type de fichier - file**

```
file fichier.txt
```

Déetecte le type : texte, binaire, script, image, etc.

3. Gestion des fichiers

Créer des fichiers

```
touch nouveau.txt
```

Crée un fichier vide

```
echo "Bonjour monde" > salut.txt
```

Écrit une ligne dans un fichier (remplace le contenu)

```
nano fichier.txt      # Éditeur simple (terminal)
vi fichier.txt        # Éditeur avancé (Vim)
gedit fichier.txt     # Éditeur graphique (GNOME)
```

⟳ Déplacer / Renommer - mv

```
mv fichier.txt /tmp/  
mv fichier.txt renomme.txt
```

📎 **Copier des fichiers - cp**

```
cp fichier.txt sauvegarde.txt  
cp fichier.txt /home/utilisateur/sauvegarde/
```

Avec options :

```
cp -uvi fichier.txt /home/utilisateur/sauvegarde/
```

- `-u` : copie **seulement si plus récent**
- `-v` : affiche ce qui est copié
- `-i` : demande confirmation

✖ Supprimer des fichiers - `rm`

```
rm fichier.txt  
rm -i fichier.txt      # Confirme avant suppression  
rm -f fichier.txt      # Supprime sans confirmation (!)
```

 **Permissions - chmod**

```
chmod 644 fichier.txt      # rw-r--r--  
chmod 755 script.sh        # rwxr-xr-x  
chmod +x script.sh         # Rend exécutable
```

 **Propriétaire - chown**

```
sudo chown alice:alice fichier.txt
```

Change le **propriétaire** et le **groupe** d'un fichier.

Octal Representation

0	000	- - -	No permissions
1	001	- - x	Only Execute
2	010	- w -	Only Write
3	011	- w x	Write and Execute
4	100	r - -	Only Read
5	101	r - x	Read and Execute
6	110	r w -	Read and Write
7	111	r w x	Read, Write and Execute

JULIA EVANS
@b0rk

unix permissions

4

There are 3 things you can do to a file

↓ read ↓ write ↓ execute

ls -l file.txt shows you permissions. Here's how to interpret the output:

r w-	r w-	r --	b0rk staff
↑ b0rk (user)	↑ staff (group)	↑ ANYONE	
can read & write	can read & write	can read	

File permissions are 12 bits

setuid setgid
↓
000 user group all
sticky rwx rwx rwx rwx
= 110 110 110 100
= 6 4 4 4

For files:

r = can read
w = can write
x = can execute

For directories, it's approximately:

r = can list files
w = can create files
x = can cd into & access files

110 in binary is 6
So rw- r-- r--
= 110 100 100
= 6 4 4

chmod 644 file.txt
means change the permissions to:
rW- r-- r--
Simple!

setuid affects executables
\$ls -l /bin/ping

rwS r-x r-x root root
this means ping always runs as root

setgid does 3 different unrelated things for executables, directories, and regular files.



- Comment représenter des droits d'accès sous forme numérique plutôt que sous la forme d'une chaîne de caractères telle que rw-r--r-- ?
- La représentation des droits d'accès sous la forme d'une chaîne de caractères a deux caractéristiques remarquables :
 - chaque droit d'accès a une position bien précise;
 - chaque droit d'accès n'a que deux états possibles, accordé ou refusé.
- On peut donc envisager de représenter des droits d'accès sous la forme d'un nombre binaire comprenant neuf chiffres, un par droit d'accès, chaque chiffre correspondant au droit situé à la même position dans la représentation des droits d'accès sous la forme d'une chaîne de caractères et, pour chaque chiffre, 1 représentant l'accord d'un droit et 0 son refus.
- Ainsi, rw- r-- r--peut être représenté par 110 100 100

- Par définition, le nombre binaire 110100100 se décompose en puissances de 2 ainsi :

$$110100100 = 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$

- soit, en factorisant la puissance de 2 la plus petite de chaque ligne :

$$110100100 = (1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0) \times 2^6 + (1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0) \times 2^3 + (1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0) \times 2^0$$

- ce qui correspond au nombre 644 en base 8.

- Attention à ne pas dire « six cent quarante-quatre » car il ne s'agit pas de 644 en base 10. En l'occurrence, on ne peut pas vraiment dire mieux que « six quatre quatre »...

Change mode

- commande : chmod
- La commande chmod permet de modifier les droits d'accès de fichiers ou de répertoires.
- Son premier argument indique quels droits d'accès modifier dans les fichiers dont les chemins d'accès figurent dans les arguments suivants.

- Les modifications à apporter aux droits d'accès peuvent être indiquées :

- de manière symbolique :

```
$ chmod o-r fichier.txt
```

- de manière numérique :

```
$ chmod 644 fichier.txt
```

- L'option -R permet d'agir récursivement.

Modification symbolique

- Il est possible de modifier les droits d'accès indépendamment grâce à l'écriture symbolique :
 - u propriétaire (user)
 - g groupe (group)
 - o les autres (others)
 - a tous (all)
- + ou - pour ajouter ou retirer un droit
- Suivi du/des droit(s) à ajouter ou retirer
- ex: chmod u+x fichier (ajoute l'exécution à l'utilisateur)

User file creation mask

- commande : umask
- La commande umask affiche (sans argument) ou modifie (avec un argument) le masque de création des fichiers de l'interpréteur de commandes :
- affichage

```
$ umask
```

- modification

```
$ umask 077
```

Permission	Fichier			Répertoire		
	user	group	other	user	group	other
Prédéfini	6	6	6	7	7	7
Umask	0	0	2	2	0	2
Defaut	6	4	4	7	5	5

 **Comparer deux fichiers - diff, cmp**

```
diff fichier1.txt fichier2.txt  
cmp fichier1.bin fichier2.bin
```

🔗 4. Alias et liens

📌 Alias

◆ Temporaire :

```
alias ll='ls -lh'  
alias clr='clear'
```

◆ Permanent :

Dans `.bashrc` ou `.zshrc` :

```
alias maj='sudo apt update && sudo apt upgrade'
```

Puis recharger :

```
source ~/ .bashrc
```

🔗 Liens symboliques - `ln -s`

```
ln -s /chemin/reel/fichier.txt raccourci.txt
```

- Fonctionne comme un **raccourci**
- Peut pointer vers un autre disque
- Si la cible est supprimée → **lien brisé**

Vérification :

```
ls -l raccourci.txt
```

Affiche :

```
raccourci.txt -> /chemin/reel/fichier.txt
```

brick Liens physiques - ln

```
ln fichier.txt lien.txt
```

- Partage le **même inode**
- Si l'original est supprimé, l'autre fonctionne encore
- Ne fonctionne **que sur le même système de fichiers**

Vérification :

```
ls -li fichier.txt lien.txt
```

Même inode = lien dur.



Scripting Shell

Utopios® Tous droits réservés

Scripting Shell – Concepts détaillés

Un **script shell** est un fichier texte contenant une **suite de commandes** exécutées **en séquence** par un interpréteur (`bash`, `sh`, `zsh`, etc.).

“ C'est comme une **recette de cuisine automatisée** que la machine exécute ligne par ligne. ”

🎯 Pourquoi utiliser des scripts ?

- Automatiser les tâches répétitives
- Définir une logique complexe (conditions, boucles, fonctions)
- Gérer des fichiers, utilisateurs, réseaux
- Centraliser la logique dans des scripts modulaires

📁 Comment fonctionne un script shell ?

Pour exécuter un script :

```
./monscript.sh
```

En haut du script, on précise l'interpréteur à utiliser :

```
#!/bin/bash
```

➡ Ceci s'appelle le **shebang**, et il indique à Linux que le fichier doit être exécuté avec Bash.

✓ 1. Utiliser des variables

Une variable est un **nom** associé à une **valeur**.

```
nom="Alice"  
echo "Bonjour $nom"
```

- Pas d'espace autour du `=`
- On accède avec `$nom`

Types supportés :

- Chaînes
- Nombres
- Tableaux
- Tableaux associatifs (en Bash uniquement)

✍ Exemple :

```
couleurs=("rouge" "vert" "bleu")  
echo ${couleurs[1]} # Affiche "vert"
```

✓ 2. Conditions (logique)

```
if [ $age -ge 18 ]; then
    echo "Adulte"
else
    echo "Mineur"
fi
```

Explications :

- `-ge` = supérieur ou égal
- `then` = commence le bloc
- `fi` = termine le bloc

Autres tests :

- Fichier : `[-f monfichier]`
- Texte : `["$a" = "$b"]`
- Nombres : `-eq`, `-lt`, `-gt`, etc.

Liste des conditions `if` en Bash

1. Comparaison numérique

Test	Signification	Exemple
<code>-eq</code>	égal à	["\$a" -eq "\$b"]
<code>-ne</code>	différent de	["\$a" -ne "\$b"]
<code>-lt</code>	inférieur à	["\$a" -lt "\$b"]
<code>-le</code>	inférieur ou égal à	["\$a" -le "\$b"]
<code>-gt</code>	supérieur à	["\$a" -gt "\$b"]
<code>-ge</code>	supérieur ou égal à	["\$a" -ge "\$b"]

2. Comparaison de chaînes de caractères

Test	Signification	Exemple
=	égalité	["\$a" = "\$b"]
==	égalité (aussi valable dans [[]])	[["\$a" == "\$b"]]
!=	différent	["\$a" != "\$b"]
<	inférieur (ordre alphabétique, dans [[]] seulement)	[["\$a" < "\$b"]]
>	supérieur (ordre alphabétique, dans [[]] seulement)	[["\$a" > "\$b"]]
-z	chaîne vide	[-z "\$a"]
-n	chaîne non vide	[-n "\$a"]

3. Tests de fichiers

Test	Signification	Exemple
-e	existe	[-e fichier.txt]
-f	fichier régulier	[-f fichier.txt]
-d	dossier (répertoire)	[-d mon_dossier]
-r	lisible	[-r fichier.txt]
-w	inscriptible	[-w fichier.txt]
-x	exécutable	[-x script.sh]
-s	fichier non vide	[-s fichier.txt]
-L	lien symbolique	[-L lien]
-b	fichier bloc spécial	[-b /dev/sda]
-c	fichier caractère spécial	[-c /dev/tty]

4. Combinaison logique

Syntaxe	Signification	Exemple
-a	ET logique (deprecated, préfère &&)	["\$a" -gt 0 -a "\$b" -gt 0]
-o	OU logique (deprecated, préfère)	["\$a" -gt 0 -o "\$b" -gt 0]
&&	ET logique dans [[]]	[["\$a" -gt 0 && "\$b" -gt 0]]
	OU logique dans [[]]	[["\$a" -gt 0 "\$b" -gt 0]]
!	NON logique (négation)	[! -e fichier.txt]

5. Comparaison entre fichiers

Test	Signification	Exemple
file1 -nt file2	file1 est plus récent que file2	[fichier1.txt -nt fichier2.txt]
file1 -ot file2	file1 est plus ancien que file2	[fichier1.txt -ot fichier2.txt]
file1 -ef file2	file1 et file2 sont le même fichier (lien dur/symbolique)	[fichier1.txt -ef fichier2.txt]

Syntaxe rapide

- Simple crochets : [condition]
- Doubles crochets : [[condition]] → plus puissant (et supporte &&, ||, <, >, regex, etc.)
- Parenthèses doubles pour arithmétique : ((expression))

Exemples :

```
if [ "$a" -eq 5 ]; then
    echo "a vaut 5"
fi

if [[ "$nom" == "admin" || "$nom" == "root" ]]; then
    echo "Utilisateur privilégié"
fi

if (( a > b )); then
    echo "a est plus grand que b"
fi
```

✓ 3. Boucles

🔁 Boucle `for` :

```
for fichier in *.txt
do
    echo "$fichier"
done
```

Parcourt tous les fichiers `.txt`.

🔁 Boucle `while` :

```
compteur=1
while [ $compteur -le 5 ]; do
    echo $compteur
    compteur=$((compteur+1))
done
```

✓ 4. Fonctions

```
saluer() {  
    echo "Salut, $1"  
}  
saluer "Alice"
```

- `$1` = premier argument
- Utiles pour rendre le code **modulaire**

✓ 5. Paramètres en ligne de commande

Exécution :

```
./bonjour.sh Alice 25
```

Script :

```
echo "Bonjour $1, âge $2"
```

Autres :

- `$@` = tous les paramètres
- `$#` = nombre de paramètres

✓ 6. Include d'autres fichiers

```
source config.sh
```

Permet d'utiliser les variables ou fonctions définies dans un autre fichier.

✓ 7. Codes de retour et erreurs

Chaque commande retourne un code :

- 0 = succès
- 1-255 = erreur

À tester avec :

```
echo $?
```

Ou :

```
if ls monfichier; then
    echo "Trouvé"
else
    echo "Introuvable"
fi
```

Flux standards

Chaque commande utilise 3 canaux :

Flux	N°	Rôle
stdin	0	Entrée (clavier, pipe...)
stdout	1	Sortie standard (résultats)
stderr	2	Erreurs

Rediriger la sortie

Objectif	Syntaxe	Effet
Rediriger stdout	commande > fichier.txt	Sauvegarde la sortie
Ajouter à un fichier	commande >> fichier.txt	Ajoute à la fin
Rediriger stderr	commande 2> erreur.log	Sauvegarde les erreurs
stdout + stderr vers un seul	commande > tout.log 2>&1	Combine les deux dans un fichier
Raccourci	commande &> fichier.log	Même chose

 **Exemple réel**

```
ls bon.txt mauvais.txt > sortie.txt 2> erreur.txt
```

- Si `bon.txt` existe : dans `sortie.txt`
- Si `mauvais.txt` manque : erreur dans `erreur.txt`

📘 Rediriger l'entrée (stdin)

```
cat < fichier.txt
```

Utile avec les boucles :

```
while read ligne; do
    echo "Ligne : $ligne"
done < monfichier.txt
```

✎ Chaine avec des pipes |

Un **pipe** relie deux commandes :

```
ps aux | grep firefox | wc -l
```

- `ps aux` = liste des processus
- `grep` = filtre
- `wc -l` = compte les lignes

Schéma des flux

Clavier → [stdin] → commande → [stdout] → Écran ou fichier
➡ [stderr] → Erreurs visibles/log

Avec un **pipe** :

[stdout de cmd1] → [stdin de cmd2]

Avec redirection :

commande > sortie.txt 2> erreur.txt

➡ Sortie normale + erreurs = séparées

Cas d'usage pratiques

1. Journaliser succès et erreurs séparément :

```
./build.sh > succes.log 2> erreur.log
```

2. Lire depuis un fichier :

```
./bonjour.sh < noms.txt
```

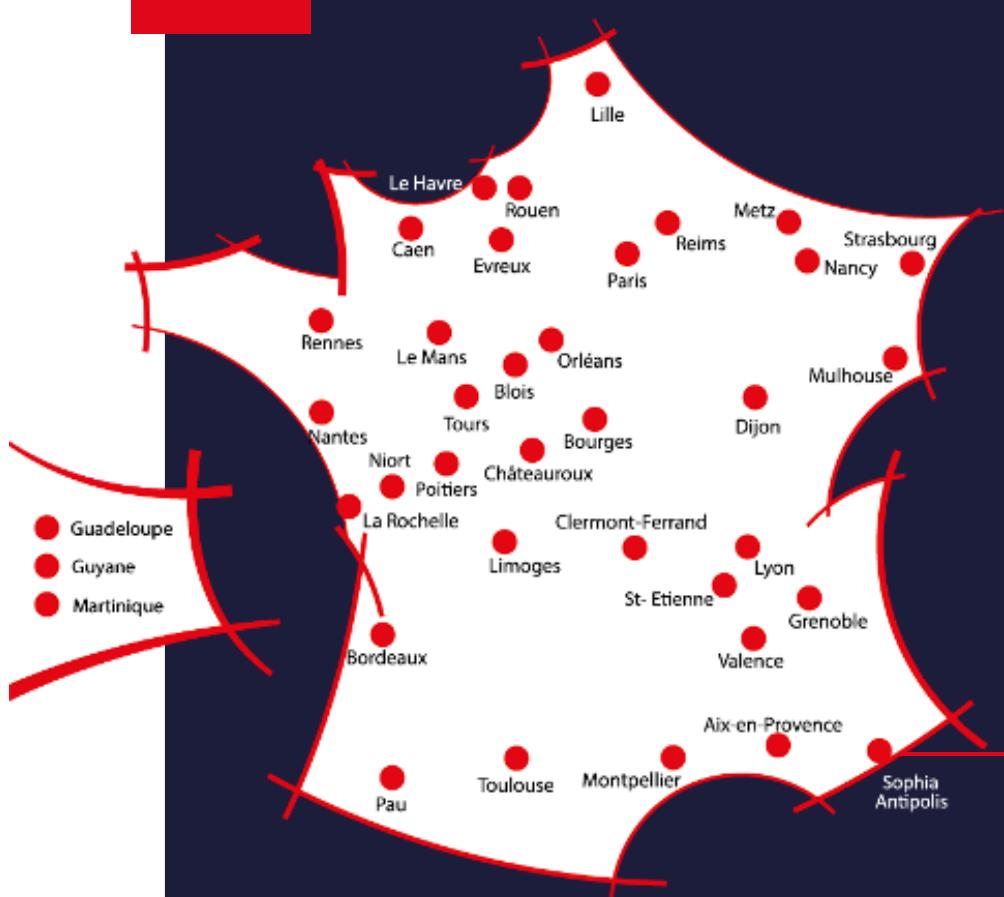
```
while read nom; do
    echo "Salut $nom"
done
```

3. Ignorer les erreurs :

```
commande 2>/dev/null
```

4. Tout cacher (silencieux) :

```
commande > /dev/null 2>&1
```

Découvrez également
l'ensemble des stages à votre disposition
sur notre site m2iformation.fr

m2iformation.fr

