Introduction aux systèmes GNU/Linux

Module IntroLinux – S26E03

Philippe Latu / Université de Toulouse
inetdoc.net

Cours 3
Le plan

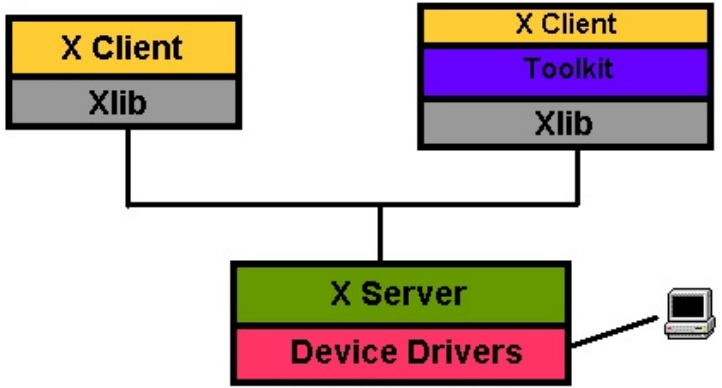
- Environnements graphiques
- Shell Bash
- Permissions Unix
- Initialisation système

Environnements graphiques

Environnements graphiques – Historique

X Window System (MIT, X Consortium)

- 1986 : première version diffusée par le MIT
- 1987 : protocole X11, toujours en usage
- 1992 : projet XFree86
  - Au départ : PC à processeur Intel i386
  - Ensuite : portage sur de nombreuses architectures



X = Protocole réseau

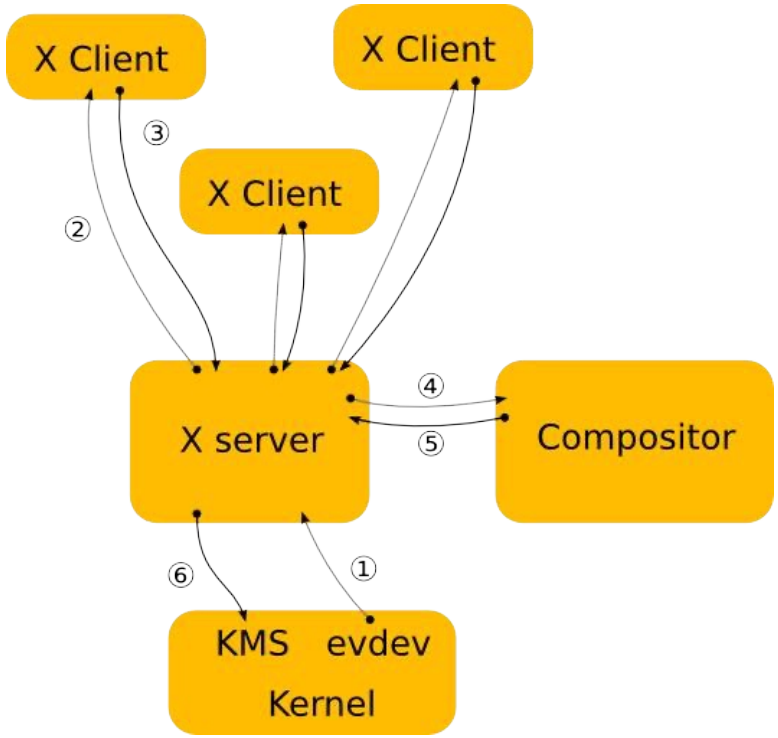
Fondation X.Org

- 2004 : création, x.org
- 2005 : X11R7, architecture modulaire
- 2008 : début du projet Wayland

Environnements graphiques – X.Org

Architecture X.Org – freedesktop.org

- Modèle client/serveur
- Le serveur X sert d'intermédiaire entre les clients et le compositeur
- Le compositeur assemble l'image finale et la renvoie au serveur X



Solution modulaire

- ⊕ Ajout de fonctions supplémentaires
- ⊖ Accumulation de code au fil du temps

Fonctions réseau

- Applications graphiques accessibles à distance : `ssh -X`

Environnements graphiques – Wayland

Architecture Wayland – freedesktop.org

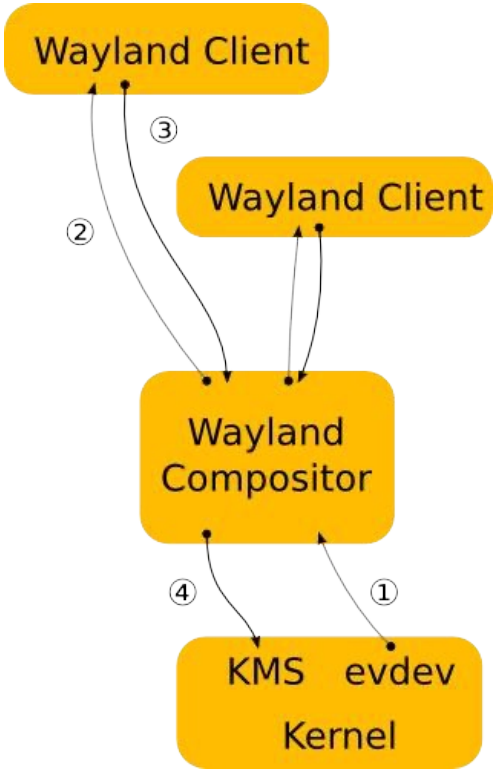
- Architecture « simplifiée »
- Le compositeur est le serveur d'affichage : plus d'intermédiaire
- XWayland : compatibilité des applications X11

Adoption

- Protocole par défaut de GNOME, de KDE Plasma et des grandes distributions
- Pilotes graphiques : Mesa (Intel, AMD) et NVIDIA

Sécurité

- Isolement des clients : pas d'écoute globale du clavier
- Capture d'écran et partage via des portails autorisés



Wayland en 2026 : fin des sessions X11

GNOME 50 « Tokyo » (mars 2026)

- Session X11 supprimée de Mutter, GNOME Shell et GDM
- Bureau d'Ubuntu 26.04 LTS et de Fedora 44

KDE Plasma 6.8 (14 octobre 2026)

- Session Wayland seule ; les applications KDE restent compatibles X11

Xfce

- xfwl4 : nouveau compositeur Wayland écrit en Rust, 2e préversion en août 2026

Accès à distance

- RDP intégré à GNOME et KDE ; `waypipe` pour une seule application

Sources : 9to5Linux, mars 2026 ; heise, oct. 2026 ; blogs.kde.org ; blog.xfce.org, août 2026

Environnements graphiques – Développement

- Un écosystème (beaucoup plus structuré qu'il n'y paraît)
  - Un environnement graphique est aussi une chaîne de développement
  - Démonstrateur de l'état courant des avancées techniques
- Très (trop) grande diversité des développements et des bibliothèques
- Modèle open source : avantages et inconvénients
  - ⊕ Introduction facile de nouvelles fonctionnalités
  - ⊖ Coordination difficile entre projets « concurrents »
  - Exemple : la transition X11 → Wayland (2008-2026)

Environnements graphiques – Exemples

Bureaux légers : LXQt et Xfce

- Faibles capacités de traitement, systèmes embarqués
- Xfce : bureau par défaut de Kali Linux et de Xubuntu

GNOME et GTK

- RAD : Cambalache, successeur de Glade

KDE Plasma et Qt

- RAD : Qt Creator

RAD : Rapid Application Development

Source : Wikipédia, « Glade Interface Designer »

Bilan environnements graphiques

- Wayland
  socle commun des bureaux en 2026
- Cloud et Virtual Desktop Infrastructure (VDI)
- Compromis performances / consommation énergétique
- Rapid Application Development

Shell Bash

Shell Bash – Historique

- 1979 : Unix version 7 → Bourne shell (sh)
- 1989 : GNU Bash → Bourne Again Shell (« born again »)
  - Norme : POSIX.1-2024 (IEEE Std 1003.1-2024), volume « Shell and Utilities »
  - Debian 13 : Bash 5.2 ; Bash 5.3 publié en juillet 2025
- Interpréteur de commandes : historique, complétion, édition de ligne
- Shell : environnement de développement
  - Algorithmique : tests, conditions, boucles, fonctions, arithmétique, tableaux
  - Développements : accolades, tilde, paramètres, commandes, noms de fichiers
- Documentation : manuel de référence GNU, Bash Guide

Sources : Wikipédia, « Bourne shell » ; IEEE SA, 1003.1-2024 ; packages.debian.org ; LWN, juillet 2025

Shell Bash – Ligne de commande

- Correction des fautes de frappe
  - `shopt -s cdspell` (commande cd)
  - `shopt -s dirspell` (complétion)
- Complétion automatique : touche Tab (paquet `bash-completion`)
- Historique et rappel des commandes antérieures
  - Commande `history`, rappel avec `!n` ou `!!`
- Raccourcis clavier (bibliothèque Readline)
  - Recherche dans l'historique : Ctrl+R puis caractères de la commande
  - Curseur en début de ligne : Ctrl+A ; en fin de ligne : Ctrl+E
  - Défilement de l'historique : flèches haut et bas

Shell Bash – Séquencement

- Tubes (pipes) : `ip nei ls | grep de:b4:8a:f8:f1:4d`
- Redirection avec création ou écrasement du fichier : `echo "syntax on" > ~/.vimrc`
  - Protection contre l'écrasement : `set -o noclobber`
- Redirection avec ajout en fin de fichier : `echo "syntax on" >> ~/.vimrc`
- Enchaînements selon le code de retour (`echo $?` ; 0 = succès)
  - Inconditionnel : `commande1 ; commande2`
  - Si commande1 réussit : `commande1 && commande2`
  - Si commande1 échoue : `commande1 || commande2`

Shell Bash – Here document

- Formater un texte sur plusieurs lignes

```
cat << EOF
The current working directory is: $PWD
You are logged in as: $(whoami)
EOF
```

- Créer un script sans éditeur
  - `$i`, `$HOME` et `$((…))` sont développés à la génération
  - Le délimiteur `EOF` doit commencer la ligne

```
for i in {0..2}
do
  cat << EOF >> start3vms.sh
echo "Start vm $i ..."
$HOME/vm/scripts/ovs-startup.sh \
  vm$i.qcow2 512 $((i + 10))
EOF
done
```

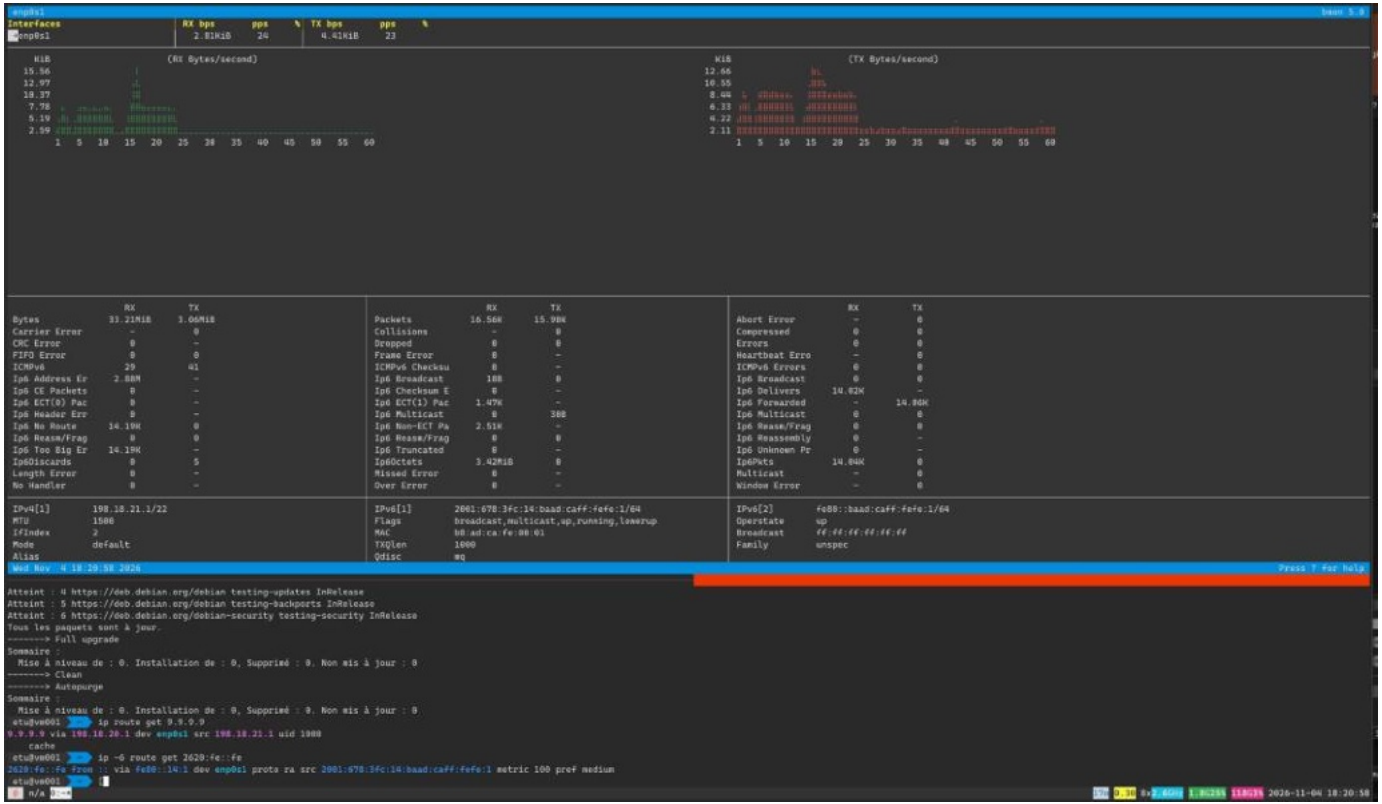
- Envoyer une suite d'instructions via SSH
  - `'EOF'` entre apostrophes : aucun développement local

```
ssh etu@srv.example.com sh -s << 'EOF'
echo "My own VM list"
cd $HOME/vm
ls -lAh *.qcow2
echo "That's all folks!"
EOF
```

Shell Bash – Multiplexeurs de terminal

- tmux
- Byobu, surcouche de tmux
- Zellij

Conserver les sessions ouvertes sur un serveur malgré une coupure SSH



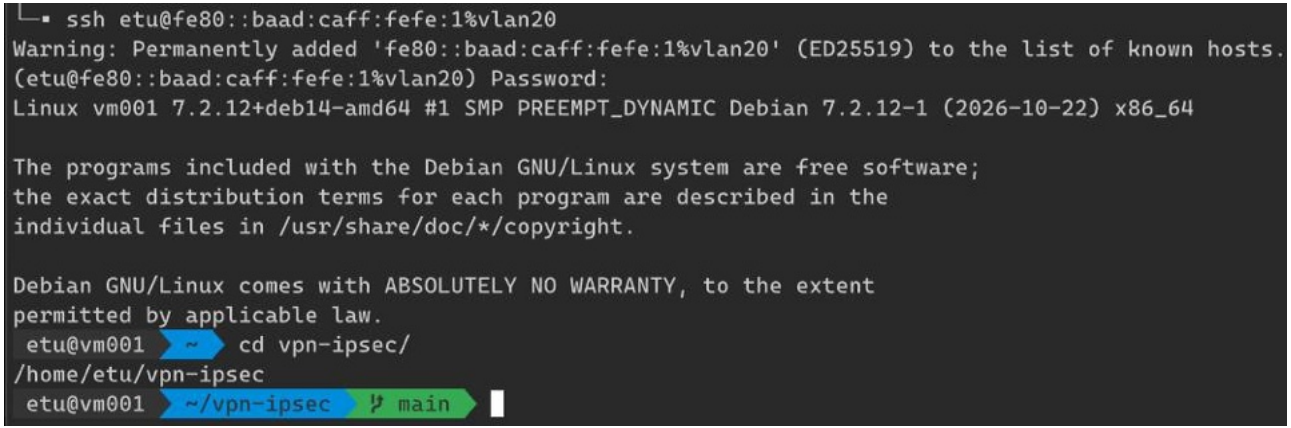
Shell Bash – Personnalisation

Outil : Oh My Bash

- dépôt GitHub du projet

Paramètres prédéfinis cohérents

- Thème, alias
- Invite de commandes détaillée
- Intégration Git
- Environnements virtuels Python



Shell Bash – Qualité des scripts

Vérifier avant d'exécuter

- `bash -n script.sh` : vérification de la syntaxe
- `shellcheck script.sh` : analyse statique, avertissements documentés

Mode strict

- `set -euo pipefail` : arrêt sur erreur, variable non définie, échec dans un tube
  - Limite : `set -e` ne s'applique pas dans les conditions `if`, `&&`, `||`

Bash 5.3 (juillet 2025)

- `${ commande; }` : substitution sans sous-processus ; variable `GLOBSORT`

Analyse critique

- Script proposé par une IA : passer ShellCheck, puis justifier chaque avertissement

Sources : shellcheck.net ; LWN, « Bash 5.3 released », juillet 2025

Bilan shell Bash
Carrefour du monde DevOps

- Dev : algorithmique de base
  l'idempotence doit être programmée
- Préalable aux outils d'automatisation (Ansible)
  traitements idempotents
- Ops : pilier essentiel de l'administration système

Idempotence : f(f(x)) = f(x) pour tout x

Préparation aux travaux pratiques

Analysez le code du script ci-contre

- Que signifie la ligne « shebang » `#!` ?
- Pourquoi `set -euo pipefail` ? Quelles limites ?
- Qu'est-ce qui distingue les conditions des deux tests `if [ … ]` ?
- À quel répertoire correspond ~ ?
- Pourquoi `cat << EOT` au lieu de `printf` ?
- Ce script est-il idempotent ? Que se passe-t-il à la 2e exécution ?
- Pourquoi `exit 0` ?

```
#!/usr/bin/env bash
set -euo pipefail
GREEN='\e[32m'
NC='\e[0m' # No Color
printf '%bPréparation des machines virtuelles%b\n' \
  "$GREEN" "$NC"
if [ ! -L ~/masters ]; then
  cat << EOT
Création du lien vers le catalogue des
fichiers images de machines virtuelles.
EOT
  ln -s /var/cache/kvm/masters ~
fi
if [ ! -d ~/vm ]; then
  cat << EOT
Création du dossier de dépôt des copies
des fichiers images de machines virtuelles.
EOT
  mkdir ~/vm
  ln -s ~/masters/scripts ~/vm/
fi
printf '%bArborescence prête%b\n' "$GREEN" "$NC"
exit 0
```

Permissions Unix

un héritage des années 1970

Permissions Unix – postulat de départ

Paradigme Unix : « Everything is a file »

- Objectif : une interface uniforme (`open`, `read`, `write`, `close`)
  - pour les applications et pour les utilisateurs

Objets représentés sous forme de fichiers

- Fichiers, répertoires, périphériques (`/dev`), tubes nommés, sockets Unix
- Processus : visibles sous forme de fichiers dans `/proc`
- Sockets réseau : pas de nom dans l'arborescence, seulement des descripteurs

Permissions Unix – classes d’utilisateurs

Un fichier a un propriétaire et un groupe ; 3 classes d’utilisateurs

- Propriétaire – **Owner/User** : le propriétaire du fichier
- Groupe – **Group** : les membres du groupe propriétaire
- Autres – **Other** : tous les autres utilisateurs

Une seule classe s’applique

- Ordre de test : propriétaire, puis groupe, puis autres
- Un propriétaire limité à `r--` ne profite pas du `rw-` accordé au groupe

Permissions Unix – droits sur un fichier

Chaque classe dispose de 3 permissions

- Lecture – **Read** : lire le contenu du fichier
- Écriture – **Write** : modifier le contenu du fichier
- Exécution – **eXecute** : exécuter le fichier (programme ou script)

Attention

- Supprimer ou renommer un fichier dépend des droits sur le répertoire

Permissions Unix – droits sur un répertoire

Un répertoire est une liste de noms (entrées)

- **r** : lister les noms des entrées (`ls`)
- **w** : créer, supprimer, renommer des entrées (avec **x**)
- **x** : traverser le répertoire, accéder à une entrée connue (`cd`)

Cas pratiques

- `r--` sans `x` : les noms s'affichent, mais pas leurs attributs
- `--x` sans `r` : accès à un fichier dont on connaît le nom
- `w` sur le répertoire : suppression possible d'un fichier en lecture seule

Permissions Unix – exemple

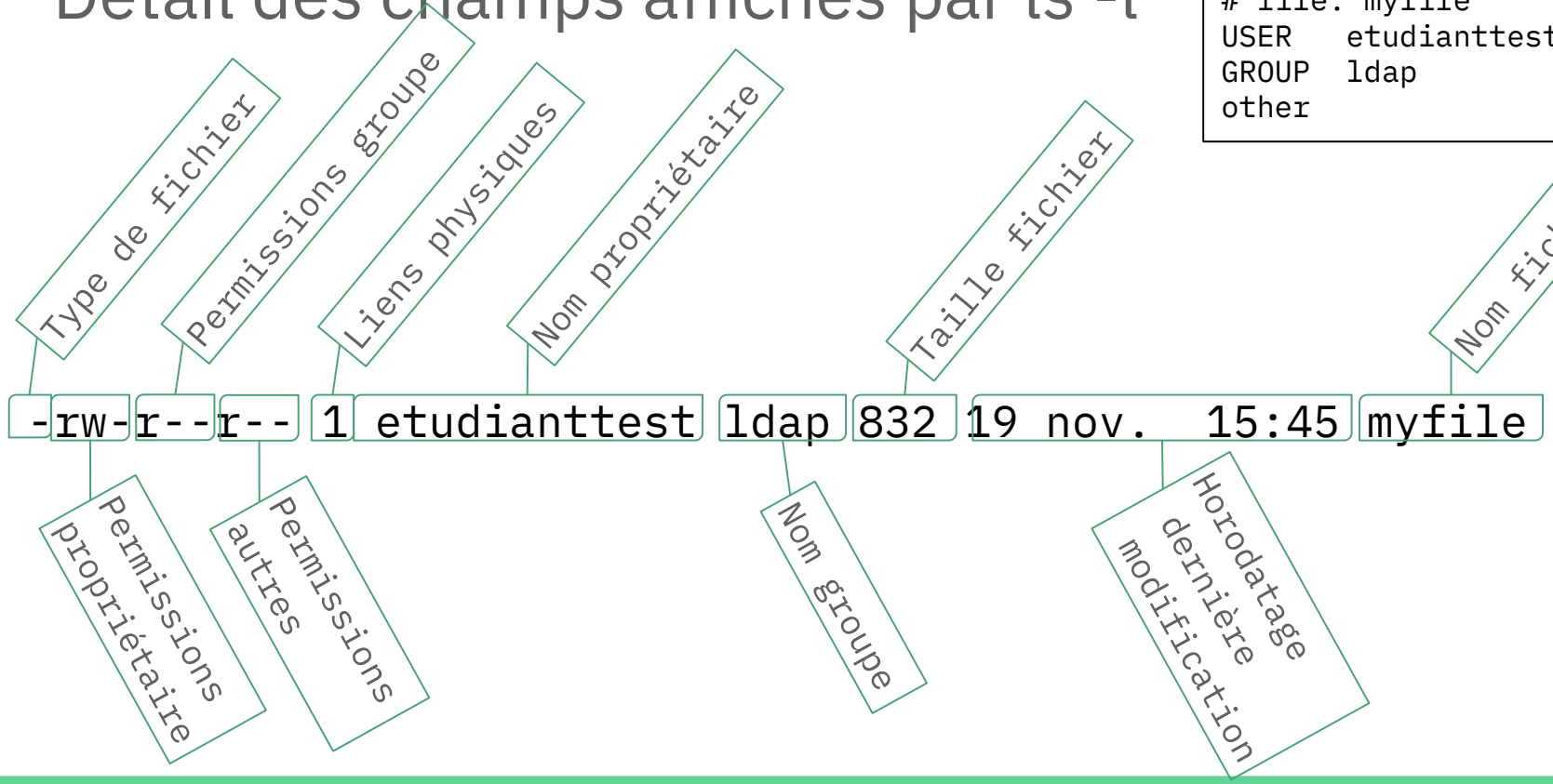
Visualisation des permissions avec la commande ls -lAh

```
┌─[etudianttest][bob][~]
└─▪ ls -lAh
total 48K
-rw-------  1 etudianttest ldap 1,2K 19 nov.  15:18 .bash_history
-rw-r--r--  1 etudianttest ldap  220 27 oct.  10:55 .bash_logout
-rw-r--r--  1 etudianttest ldap 4,4K 12 nov.  16:32 .bashrc
lrwxrwxrwx  1 etudianttest ldap   22 19 nov.  15:17 masters -> /var/cache/kvm/masters
drwxr-xr-x 15 etudianttest ldap 4,0K 12 nov.  16:32 .oh-my-bash
-rw-r--r--  1 etudianttest ldap   17 12 nov.  16:32 .osh-update
-rw-r--r--  1 etudianttest ldap  807 27 oct.  10:55 .profile
-rw-r--r--  1 etudianttest ldap    0 19 nov.  15:32 .sdirs
drwx------  2 etudianttest ldap 4,0K 27 oct.  10:58 .ssh
-rw-------  1 etudianttest ldap 8,0K 19 nov.  15:18 .viminfo
-rwxr-xr-x  1 etudianttest ldap  609 19 nov.  15:18 virt-tree.sh
drwxr-xr-x  2 etudianttest ldap 4,0K 19 nov.  15:17 vm
```

Permissions Unix – métadonnées d'un fichier

Détail des champs affichés par ls -l

```
getfacl -t myfile
# file: myfile
USER   etudianttest  rw-
GROUP  ldap          r--
other                r--
```



Permissions Unix – types de fichiers

-	fichier ordinaire
b	fichier spécial de périphérique en mode bloc (disques)
c	fichier spécial de périphérique en mode caractère (terminaux, /dev/null)
d	répertoire
l	lien symbolique
p	tube nommé (FIFO) : communication entre processus
s	socket Unix : communication entre processus

Permissions Unix – notations

Notation symbolique ou numérique (octale)

Chaque bit de permission d'une classe correspond à une puissance de 2

Le masque des permissions s'écrit sous deux formes équivalentes :

- notation symbolique : `rwxr-x---`
- notation numérique : `750`

r	w	x
2^2	2^1	2^0
4	2	1

`rw- r-- --- = 4+2 4 0` : masque des permissions = 640

`rwx r-x r-x = 4+2+1 4+1 4+1` : masque des permissions = 755

Permissions Unix – exemple

Manipulations simples sur les permissions d'un script Bash

Commande `chmod` : `+x` sans classe ajoute x à toutes les classes, filtré par `umask` (ici 022)

```
cat << EOF > hello.sh
#!/bin/bash

echo "Hello, World!"

exit 0
EOF
```

```
ls -l hello.sh
-rw-r--r-- 1 etudianttest ldap 42 19 nov.  16:57 hello.sh

chmod +x hello.sh

ls -l hello.sh
-rwxr-xr-x 1 etudianttest ldap 42 19 nov.  16:57 hello.sh

chmod 750 hello.sh

ls -l hello.sh
-rwxr-x--- 1 etudianttest ldap 42 19 nov.  16:57 hello.sh
```

Permissions Unix – permissions étendues

3 permissions supplémentaires (valeur octale) :

SETUID (4000)	Le processus prend l'UID effectif du propriétaire du fichier
SETGID (2000)	Exécutable : le processus prend le GID effectif du groupe du fichier Répertoire : les nouveaux fichiers héritent du groupe du répertoire
Sticky bit (1000)	Répertoire partagé : seul le propriétaire d'un fichier (ou du répertoire, ou root) peut le supprimer ou le renommer

Exemples

```
ls -lAh /usr/bin/passwd
-rwsr-xr-x 1 root root 67K 11 nov.  09:28 /usr/bin/passwd

ls -ld /var/local
drwxrwsr-x 2 root staff 4096  3 oct.  23:30 /var/local

ls -ld /var/tmp
drwxrwxrwt  5 root root  4,0K 19 nov.  09:25 /var/tmp
```

Permissions Unix – au-delà du SETUID

Capacités de fichiers

- Un privilège ciblé au lieu de tous les droits root : `getcap`, `setcap`
  - Exemple : `setcap cap_net_raw+ep` pour un outil réseau

Élever ses privilèges en 2026

- sudo-rs : `sudo` par défaut d'Ubuntu depuis 25.10, l'original devient `sudo.ws`
- `run0` (systemd ≥ 256) : sans binaire SETUID, autorisation par polkit

Debian 13 : /tmp en tmpfs

- Monté avec `mode=1777,nosuid,nodev` : sticky bit, SETUID neutralisé
- Nettoyage : fichiers de /tmp après 10 jours, de /var/tmp après 30 jours

Sources : notes de version Ubuntu 26.04 ; man7.org, run0(1) ; notes de publication Debian 13

Bilan permissions Unix

- Paradigme Unix
  Everything is a file
- Permissions : savoir essentiel de l'administration système
- Utilisateur, groupe et permissions sont indissociables
- Piliers de la sécurité système
  capacités, run0 : moins de SETUID

Métadonnées : stockées dans l'inode

Préparation aux travaux pratiques

À l'aide de la commande `ls`, rechercher et afficher les permissions d'un « objet » du type :

- Programme exécutable
- Lien symbolique
- Périphérique en mode caractère
- Périphérique en mode bloc
- Socket de la famille Unix
- Exécutable SETUID (`find / -perm -4000 -type f`)

Quels sont les traitements assurés par les commandes suivantes ?

- `chmod`
- `chown`
- `chgrp`
- `setfacl`
- `umask`
- `stat`

Initialisation système

Initialisation système – les étapes

POST + UEFI/BIOS	Power-On Self-Test Unified Extensible Firmware Interface Basic Input Output System	Test des composants à la mise sous tension
MBR/GPT	Master Boot Record GUID Partition Table	Table des partitions du disque d'amorçage
GRUB	GRand Unified Bootloader	Chargeur d'amorçage
Noyau	Linux	Pilotage de l'architecture matérielle
initramfs	Initial RAM file system	Accès au système de fichiers racine
systemd	System and service manager	Pilotage de l'initialisation
Cible	Target unit (graphical.target...)	État attendu du système après initialisation

Initialisation système – POST

POST = *Power-On Self-Test*

- Premiers codes exécutés à la mise sous tension
- Tests de fonctionnement des composants et détection de pannes
- Initialisation de la carte mère : mémoire, processeurs, bus

Micrologiciels (*firmwares*)

- Carte mère : UEFI, stocké en mémoire flash
- Périphériques : chargés par les pilotes du noyau depuis `/lib/firmware`
  - Wi-Fi, GPU (Graphics Processing Unit), NPU (Neural Processing Unit)
  - Debian : paquets `firmware-*` de la section non-free-firmware (« binary blobs »)

Tendance : multiplication des micrologiciels de moins en moins **« micro »**

Initialisation système – BIOS/UEFI

BIOS = *Basic Input Output System* (IBM PC, 1981)

- Service « d'abstraction du matériel » à partir de zones mémoire prédéfinies
- Ordre des périphériques d'amorçage ; table de partitions MBR

UEFI = *Unified Extensible Firmware Interface*

- Table de partitions GPT, partition système EFI (ESP, FAT32)
- Shell UEFI, pilotes et applications EFI
- Amorçage réseau : PXE (hérité du BIOS) et HTTP(S) Boot
- Secure Boot : vérification de la signature du chargeur (shim, GRUB) et du noyau

Source : BBC News, « Twenty five years of the IBM PC », 2006

Initialisation système – Secure Boot en 2026

Chaîne de confiance

- UEFI (base db) → shim signé par Microsoft → GRUB → noyau signé

Renouvellement des certificats

- Microsoft UEFI CA 2011 : expiré le 27 juin 2026
  - Les shims déjà signés continuent de démarrer
  - Nouveaux shims signés par la Microsoft UEFI CA 2023
  - Micrologiciel sans le certificat 2023 : refus d'un shim signé seulement en 2023

Vérifier une machine

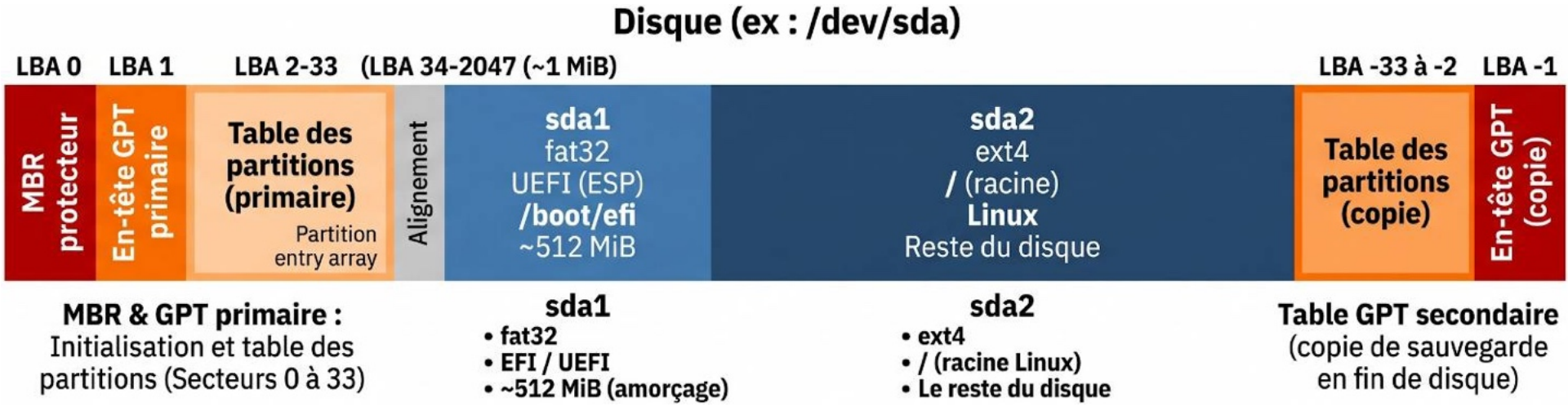
- `mokutil --sb-state`
- `mokutil --db | grep 'UEFI CA 2023'` ; mise à jour avec `fwupdmgr`

Sources : Microsoft Tech Community, 2026 ; AlmaLinux, juillet 2026

Initialisation système – GRUB

GRUB = *GRand Unified Bootloader*

- Chargeur d'amorçage : sélection du noyau et de ses paramètres
- Identification des partitions et des systèmes de fichiers
- GRUB 2.14 (janvier 2026) : BLS, images noyau unifiées (UKI), TPM2, EROFS
- Alternatives UEFI : systemd-boot, UKI lancée directement



Sources : Phoronix, « GRUB 2.14 Released », janvier 2026 ; schéma : disque GPT avec partition EFI (sda1) et BIOS boot (sda2)

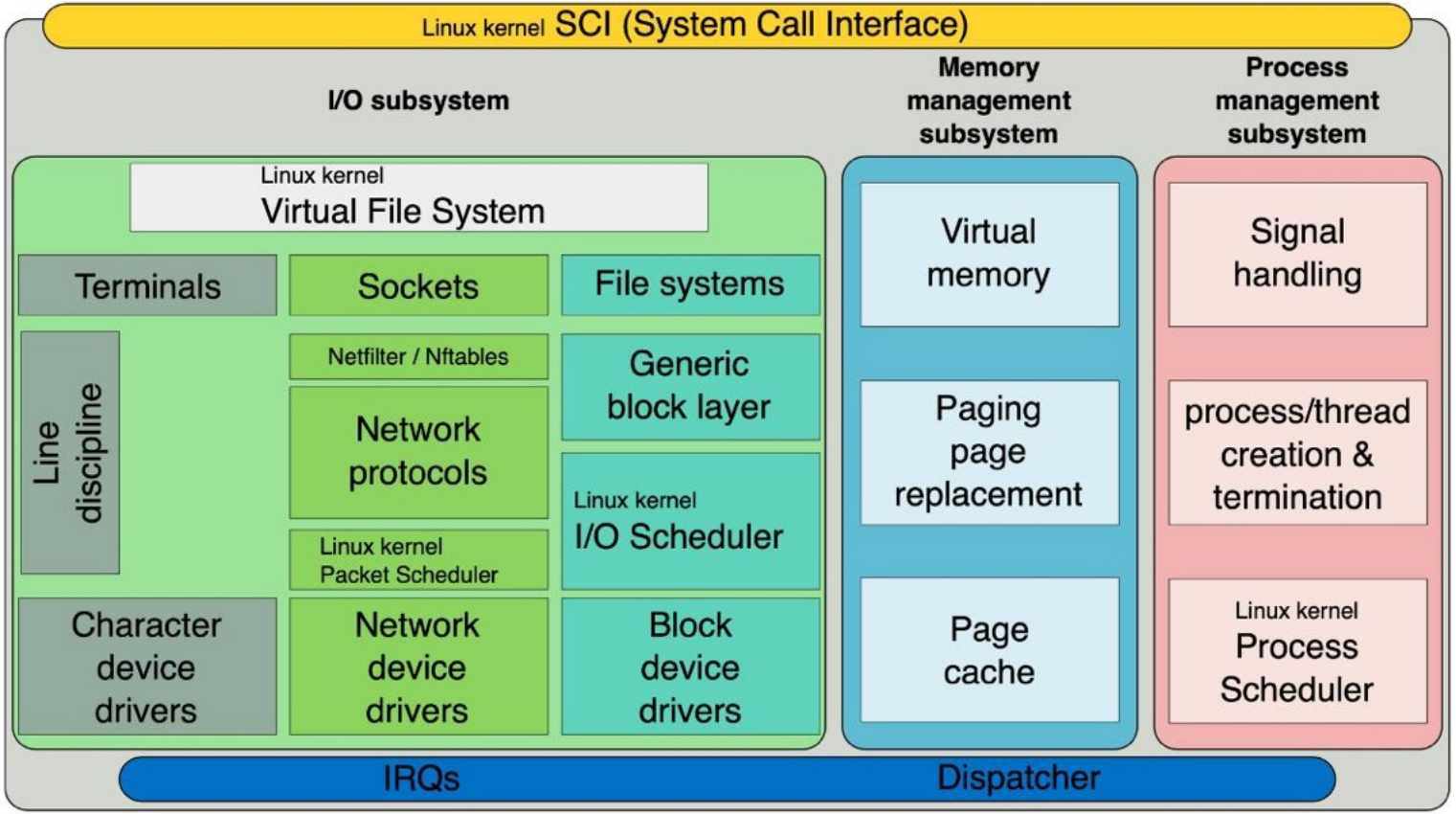
Initialisation système – noyau Linux

Séquence d'initialisation

1. Détection de l'architecture matérielle : ARM64 / AMD64 / RISC-V
2. Mémoire virtuelle : pagination, contraintes d'ordre (memory barrier)
3. Ressources de l'ordonnanceur : horloges, interruptions
4. Paramètres de la ligne de commande (`/proc/cmdline`)
5. Décompression de l'initramfs : BusyBox, modules propres au matériel
   - Debian 13 : initramfs-tools ; Ubuntu 26.04 LTS : dracut
6. Montage du système de fichiers racine, arborescence FHS
7. Lancement du processus init (PID 1) : systemd

Source : notes de version Ubuntu 26.04 LTS

Initialisation système – architecture du noyau



Initialisation système – systemd

Gestion des services et des processus de l'espace utilisateur (PID 1)

- Démarrage parallèle : dépendances entre unités, activation par socket
- Gestion dynamique des périphériques matériels : udev, hotplug
- Contraintes unifiées par les cgroups v2
  - CPU, mémoire, entrées-sorties disque, nombre de processus
- Découpage hiérarchique : `system.slice` ≠ `user.slice`
- Alternatives aux services historiques
  - Planification : cron ou timers systemd
  - Journalisation : journald (rsyslog n'est plus installé par défaut depuis Debian 12)

Source : man7.org, cgroups(7)

Initialisation système – schéma de systemd

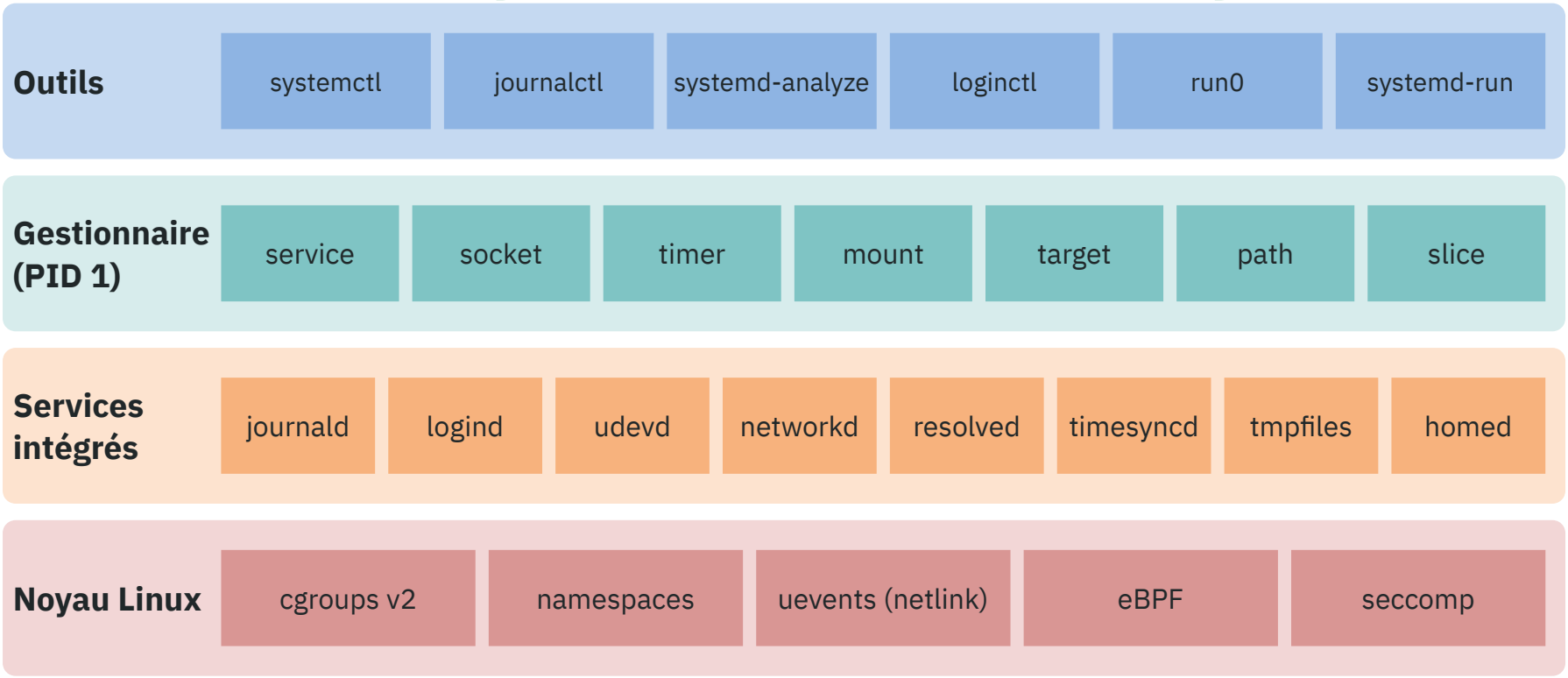


Schéma : d'après la documentation systemd (systemd.io) et les pages de manuel, systemd 257 (Debian 13)

Initialisation système – systemd en 2026

Versions

- Debian 13 : systemd 257
- systemd 258 : fin des cgroups v1
- systemd 260 (mars 2026) : fin des scripts System V
  - Plus de `systemd-sysv-generator` ni de `rc-local.service`
  - Les scripts `/etc/init.d` sans unité native ne sont plus lancés

Explorer le démarrage

- `systemd-analyze blame` ; `systemd-analyze critical-chain`
- `journalctl -b -p warning` : avertissements du dernier démarrage
- `systemctl list-dependencies graphical.target`

Sources : packages.debian.org ; GitHub systemd, v260 ; systemd.io

Bilan initialisation système

- Classer les étapes de l'initialisation dans le bon ordre
- Gestion de l'amorçage
  UEFI, Secure Boot, GRUB
- Noyau Linux
  applications et événements matériels
- systemd
  socle de la gestion des services