GARY NKWEKA

Maîtriser Ubuntu 24.04

LTS

Du premier démarrage à l'administration de serveurs



Maîtriser Ubuntu 24.04 LTS

GARY NKWEKA

Maîtriser Ubuntu 24.04 LTS

Du premier démarrage à l'administration de serveurs

WSC DIGITAL · 2026

Maîtriser Ubuntu 24.04 LTS
Du premier démarrage à l'administration de serveurs

© 2026 Gary Nkweka. Édité par WSC Digital, wscdigital.net.

Livre gratuit, qui ne peut pas être vendu. Il est diffusé sous la licence Creative Commons Attribution, Pas d'utilisation commerciale, Pas de modification 4.0 International (CC BY-NC-ND 4.0).

Vous pouvez le copier, l'imprimer et le partager librement, sous toute forme, à trois conditions : citer l'auteur et la source (Gary Nkweka, wscdigital.net) ; ne pas le vendre ni en tirer un profit commercial, seul ou avec autre chose ; ne pas le modifier, ni en diffuser une version modifiée.

Texte de la licence : creativecommons.org/licenses/by-nc-nd/4.0/deed.fr. Pour tout autre usage, écrivez à contact@wscdigital.net.

Ubuntu et Canonical sont des marques déposées de Canonical Ltd. Linux® est une marque déposée de Linus Torvalds aux États-Unis et dans d'autres pays. Les autres noms de produits cités appartiennent à leurs propriétaires. Ce livre n'est ni publié ni approuvé par Canonical.

Les commandes et les sorties d'écran ont été vérifiées sur Ubuntu 24.04.5 LTS. L'auteur et l'éditeur ont apporté le plus grand soin à leur exactitude, mais ne peuvent être tenus pour responsables des conséquences de leur utilisation. Essayez d'abord sur une machine de test.

Composé en Source Serif 4, Figtree et JetBrains Mono. Achevé de composer en octobre 2026.

Sommaire

Avant-propos vii

PARTIE I FONDATIONS

1 Linux, Ubuntu et vous 3
2 Installer Ubuntu 24.04 27
3 Premiers pas dans l'environnement 51

PARTIE II L'ENVIRONNEMENT UBUNTU

4 Le système de fichiers 77
5 Les commandes essentielles du shell 97
6 Utilisateurs, groupes et permissions 123
7 La gestion des paquets 145

PARTIE III MAÎTRISE DU SHELL BASH

8 Le shell en profondeur 169
9 Traiter du texte en ligne de commande 189
10 Écrire des scripts Bash 211

PARTIE IV SYSTEMD ET GESTION DU SYSTÈME

11 Le démarrage du système 237
12 Processus et services 253
13 Journaux et planification 285

PARTIE V STOCKAGE ET SYSTÈMES DE FICHIERS

14 Disques, partitions et systèmes de fichiers 311
15 LVM, RAID et chiffrement 331
16 Sauvegarde et restauration 351

PARTIE VI RÉSEAU SOUS UBUNTU

17 Les fondamentaux du réseau 373
18 Configurer le réseau sous Ubuntu 393
19 SSH 407

PARTIE VII SÉCURITÉ DU SYSTÈME

20 Durcir le système 427

21 Pare-feu et protection du réseau 445
22 AppArmor et le confinement 461

PARTIE VIII SERVICES SERVEUR ESSENTIELS

23 Serveurs web 479
24 Bases de données 509
25 Partager des fichiers sur le réseau 533
26 Superviser un serveur 551

PARTIE IX AUTOMATISATION ET DEVOPS

27 Git et le suivi des versions 577
28 Les conteneurs avec Docker 597
29 Automatiser avec Ansible 619

PARTIE X DIAGNOSTIC, PERFORMANCE, RÉCUPÉRATION

30 Diagnostiquer un incident 645
31 Récupérer et sauver 665

PARTIE XI PROJETS GUIDÉS

32 Projet : un serveur personnel 687
33 Projet : un poste de travail 711

ANNEXES

Annexe A — Aide-mémoire des commandes 729
Annexe B — Référence systemd 737
Annexe C — Glossaire 753
Annexe D — Ressources pour continuer 767
Annexe E — Installer vite 773

Table des matières 781

Avant-propos

Pourquoi ce livre

Les livres francophones sur Linux tombent souvent dans l'un de trois travers. Les uns survolent toutes les distributions sans en approfondir aucune. D'autres s'adressent à des administrateurs déjà formés. Beaucoup, enfin, décrivent des versions d'Ubuntu qui ne sont plus suivies. Ce livre fait un autre choix : une seule version, Ubuntu 24.04 LTS, suivie du premier démarrage jusqu'à l'administration de serveurs exposés à Internet.

Quatre principes l'ont guidé. Chaque chapitre s'appuie sur les précédents : rien n'est supposé au départ. Tout est propre à Ubuntu 24.04 : Netplan, Snap, systemd-resolved, AppArmor, les sources APT au nouveau format. Le poste de travail et le serveur sont traités l'un et l'autre. Enfin, la pratique passe avant la théorie : à partir du chapitre 2, chaque chapitre se termine par un atelier, et les deux derniers construisent un serveur personnel complet et un poste de travail professionnel.

À qui il s'adresse

- À ceux qui débutent sous Linux : étudiants, curieux, personnes en reconversion.
- Aux développeurs qui utilisent Linux tous les jours sans vraiment le connaître.
- Aux administrateurs Windows qui passent à Linux.
- À ceux qui visent un métier d'administrateur ou d'ingénieur DevOps, et veulent des bases solides avant Docker et Ansible.
- À ceux qui hébergent eux-mêmes leurs services, et veulent comprendre ce qu'ils font tourner.

Comment il est construit

Le livre compte trente-trois chapitres, répartis en onze parties, et cinq annexes. Les trois premières parties posent les fondations : l'installation, l'organisation du système, la ligne de commande et les scripts Bash. Les cinq suivantes couvrent l'admi-

nistration : systemd, le stockage, le réseau, la sécurité et les services d'un serveur. La partie IX initie à Git, à Docker et à Ansible, la partie X apprend à diagnostiquer et à réparer, et la partie XI rassemble tout dans deux projets.

Chaque chapitre suit le même plan. Il annonce ses objectifs et ce qu'il suppose connu, développe son sujet, le résume, puis propose un atelier guidé (sauf le premier chapitre, qui présente le paysage), des exercices et des pistes « pour aller plus loin ». Si vous débutez, lisez dans l'ordre. Si la ligne de commande vous est familière, vous pouvez commencer à la partie IV : l'encadré « Avant de commencer » de chaque chapitre dit ce qu'il faut savoir, et où l'apprendre.

Les annexes servent au quotidien : un aide-mémoire des commandes, une référence des fichiers d'unité systemd, un glossaire, des ressources pour continuer et un aide-mémoire d'installation.

Ce qui a été vérifié

Sauf mention contraire dans le chapitre, les commandes de ce livre ont été exécutées sur Ubuntu 24.04 LTS, en 2026, sur des machines virtuelles et des conteneurs montés pour l'occasion. Les sorties reproduites sont celles qui se sont affichées. Les seules retouches tiennent à la page : une sortie trop longue est coupée, et la coupure est marquée `[…]` ; une ligne trop large continue, en retrait, sur la ligne suivante ; quelques commandes ont été rejouées sous une forme plus compacte pour que leur sortie tienne dans la largeur. Quand un sujet n'a pas pu être démontré, comme le bureau graphique au chapitre 33, le chapitre le dit.

Les versions des logiciels avancent avec les mises à jour. Les numéros affichés chez vous différeront parfois de ceux du livre ; la démarche, elle, reste la même tout au long de la vie de la 24.04.

Les machines du livre

Les exemples se déroulent sur un petit réseau local, `192.168.1.0/24`. L'invite indique sur quelle machine taper la commande :

Invite	Machine
`gary@serveur:~$`	Le serveur Ubuntu 24.04 que l'on construit et administre au fil des chapitres, à l'adresse `192.168.1.50`
`gary@vostro:~$`	Le poste de travail d'où on l'administre
`gary@perso:~$`	Le serveur personnel du chapitre 32
`gary@portable:~$`	Le poste de travail du chapitre 33
`$`	N'importe quelle machine

Les adresses de la forme `203.0.113.42` désignent un serveur sur Internet. Elles appartiennent à une plage réservée à la documentation : remplacez-les par celle de votre machine.

Conventions

Dans une transcription, ce qui suit l'invite est ce que vous tapez ; le reste est la réponse du système. Une commande qui se termine par `\` continue sur la ligne suivante. Les touches du clavier s'écrivent Ctrl+C.

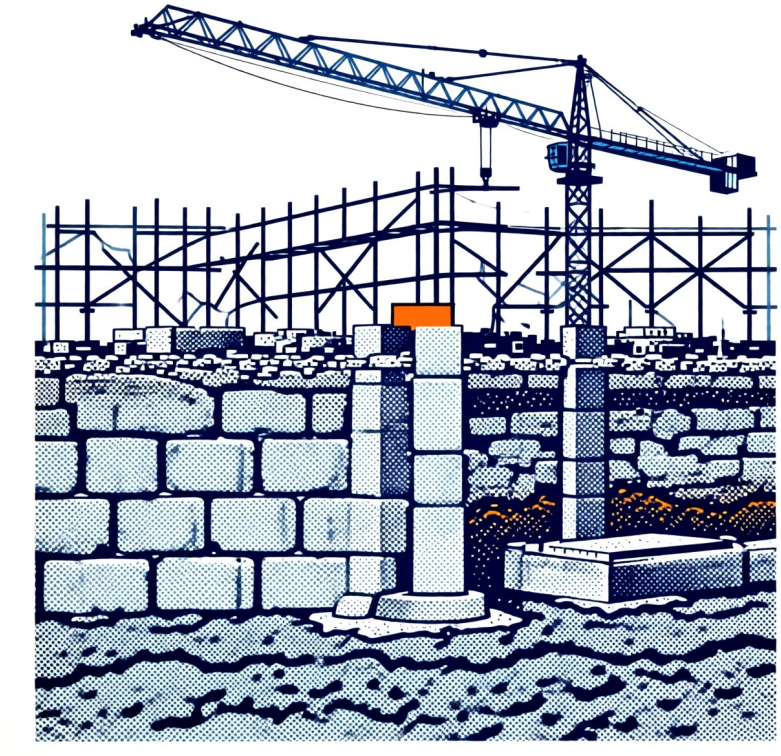
Des encadrés ponctuent le texte :

Encadré	Contenu
À retenir	L'essentiel d'une section, à garder en tête
Note	Une précision utile, qui peut se sauter en première lecture
Astuce	Un raccourci, une façon plus commode de faire
Attention	Un piège courant, une erreur qui coûte du temps
Danger	Une commande qui peut détruire des données ou couper l'accès à une machine
Repère historique	D'où vient un outil, pourquoi il est ainsi
Sur le terrain	Ce qui se passe en vrai, en production

PARTIE I

Fondations

Poser le décor. À la fin de cette partie, Ubuntu 24.04 tourne sur votre machine et vous savez ce que vous avez entre les mains.





CHAPITRE 1

Linux, Ubuntu et vous

D'où vient Linux, pourquoi Ubuntu existe, et ce que la version 24.04 LTS met entre vos mains.

DANS CE CHAPITRE

- Expliquer ce que fait un système d'exploitation et nommer ses composants : noyau, bibliothèques, shell, environnement de bureau.
- Retracer le chemin qui mène d'Unix à Ubuntu, en passant par GNU, Linux et Debian.
- Distinguer logiciel libre, open source et logiciel gratuit, et reconnaître les licences les plus courantes.
- Situer Ubuntu parmi les autres distributions Linux.
- Dire ce qu'Ubuntu 24.04 LTS apporte et jusqu'à quand elle est maintenue.
- Choisir l'édition d'Ubuntu adaptée à votre usage : Desktop, Server, images cloud ou Core.

AVANT DE COMMENCER

Aucun. Si vous savez allumer un ordinateur et ouvrir un navigateur, vous pouvez lire ce chapitre. Vous n'y taperez aucune commande : il pose le vocabulaire dont tous les autres chapitres ont besoin. Comptez une heure de lecture, que vous regagnerez largement par la suite.

1.1 Qu'est-ce qu'un système d'exploitation ?

Vous utilisez un système d'exploitation tous les jours, le plus souvent sans y penser. Quand vous ouvrez un navigateur sur un portable, quand vous lancez une application sur un téléphone, quand vous consultez votre messagerie au bureau, il y en a un qui travaille derrière. On ne le remarque que lorsqu'il tombe en panne.

À RETENIR · Définition

Un **système d'exploitation** (en anglais *operating system*, abrégé OS) est le logiciel qui fait l'intermédiaire entre le matériel d'un ordinateur et les applications. Il répartit les ressources de la machine (processeur, mémoire, disques, réseau), fournit des services communs à tous les programmes et donne à l'utilisateur un moyen de piloter l'ensemble.

Le chef d'orchestre

Prenez un orchestre symphonique. Chaque musicien sait jouer de son instrument, mais si chacun jouait quand bon lui semble, on n'entendrait que du bruit. Le chef, lui, ne joue de rien. Il indique qui entre, à quel moment, et à quel volume.

Dans un ordinateur, les musiciens sont les composants matériels : processeur, mémoire vive, disque, carte réseau, écran, clavier. Les partitions sont les applications. Le système d'exploitation tient la baguette. C'est lui qui décide quel programme obtient le processeur et pour combien de temps, qui distribue la mémoire, qui transmet chaque frappe du clavier à la bonne fenêtre, et qui veille à ce que deux programmes n'écrivent pas au même endroit du disque.

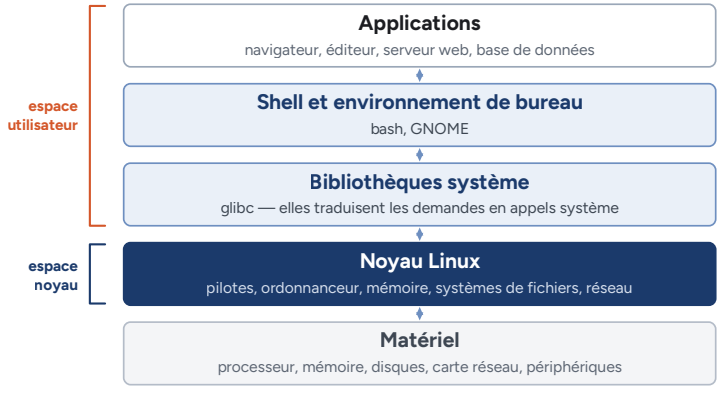


Figure 1.1 — Les couches d'un système informatique. Chaque couche ne s'adresse qu'à celle du dessous, à travers une interface bien définie.

Trois responsabilités

L'image est parlante, mais elle reste vague. Concrètement, un système d'exploitation fait trois choses, et presque tout ce que vous apprendrez à administrer dans ce livre se rattache à l'une d'elles.

Il masque la diversité du matériel. Un disque dur à plateaux, un SSD et une clé USB ne fonctionnent pas du tout de la même façon. Si chaque application devait connaître chaque modèle de chaque fabricant, on n'écrirait plus aucun logiciel. Le système présente donc à tous les programmes une notion unique, le *fichier*. Le programme demande à écrire dans un fichier, le système se charge du reste. On parle de couche d'abstraction, et vous en croiserez beaucoup d'autres.

Il partage les ressources. Une machine dispose d'un nombre fixe de cœurs de processeur et d'une quantité fixe de mémoire. Or des dizaines de programmes tournent en même temps : le navigateur, la messagerie, l'interface graphique, les services en arrière-plan. Le système accorde un peu de processeur à chacun, à tour de rôle (c'est l'*ordonnancement*). Il prête de la mémoire puis la reprend, et il empêche un programme défaillant d'entraîner les autres dans sa chute.

Il vous donne les commandes. À l'origine, on pilotait un ordinateur en tapant du texte dans un terminal. Aujourd'hui la plupart des gens passent par une interface graphique, avec des fenêtres et une souris. Les deux coexistent. Sur un serveur, le terminal reste l'outil principal, et ce n'est pas par nostalgie : une commande se note, se répète à l'identique, s'automatise et s'envoie à un collègue, ce qu'une suite de clics ne permet pas.

Les pièces d'un système Linux

Un système Linux n'est pas un bloc. C'est un assemblage de pièces développées par des équipes différentes, que vous apprendrez à reconnaître.

Le **noyau** (*kernel*) est chargé en mémoire au démarrage et y reste jusqu'à l'extinction. Lui seul parle directement au matériel. Il contient les pilotes, l'ordonnanceur, la gestion de la mémoire, les systèmes de fichiers et la pile réseau. Ce noyau s'appelle Linux, au sens strict du mot. Ubuntu 24.04 est livrée avec sa version 6.8.

Les **bibliothèques système** regroupent des fonctions que tous les programmes réutilisent. La plus importante est la `glibc`, la bibliothèque C du projet GNU. Presque tous les logiciels du système en dépendent.

Le **système d'init** est le premier programme que le noyau lance. Il démarre tout le reste : réseau, écran de connexion, services. Sur Ubuntu, c'est `systemd`, auquel la partie IV est consacrée.

Le **shell** est le programme qui lit ce que vous tapez dans un terminal et l'exécute. Celui d'Ubuntu s'appelle `bash`. Ce n'est pas le système d'exploitation, c'est une application comme une autre, dont le métier est de lancer les autres.

Les **utilitaires** sont les centaines de petits programmes dont on se sert au quotidien : `ls` pour lister des fichiers, `cp` pour les copier, `grep` pour chercher du texte. La plupart viennent du projet GNU.

L'**environnement de bureau** est la partie graphique. Ubuntu 24.04 utilise GNOME 46. Il en existe d'autres (KDE Plasma, Xfce, Cinnamon), et un serveur s'en passe très bien.

Noyau, système, distribution : ne plus confondre

Ces trois mots sont employés les uns pour les autres, y compris dans des documentations sérieuses. Autant les fixer maintenant.

Linux est un noyau, rien de plus. On ne peut pas « ouvrir une session sur Linux » : seul, il n'offre ni écran de connexion, ni gestionnaire de fichiers, ni même une ligne de commande.

GNU est un projet lancé en 1983 pour écrire un système d'exploitation entièrement libre. Il a produit le compilateur, le shell, les utilitaires et les bibliothèques. Il lui a toujours manqué un noyau utilisable : le sien, Hurd, n'a jamais atteint la maturité nécessaire.

GNU/Linux désigne la réunion des deux, le noyau de l'un et les outils de l'autre. C'est cette combinaison qui donne un système complet.

Une **distribution** est un produit fini construit sur cette base : un noyau, les outils GNU, un système d'init, un gestionnaire de paquets, des applications, une configuration par défaut et un mécanisme de mise à jour. Derrière, il y a une équipe, un calendrier de publication et des choix assumés. Ubuntu, Debian, Fedora et Arch Linux sont des distributions.

> **REPÈRE HISTORIQUE · Linux ou GNU/Linux ?**
>
> Richard Stallman, fondateur du projet GNU, tient à l'appellation « GNU/Linux » : sans les outils GNU, le noyau ne servirait à rien. Linus Torvalds, l'auteur du noyau, estime que « Linux » est entré dans l'usage et que la querelle n'apporte rien. L'usage lui a donné raison. Ce livre dit « Linux », comme tout le monde, mais gardez en tête que la moitié des commandes que vous taperez viennent de GNU.

1.2 D'Unix à Ubuntu : une brève histoire

Beaucoup de choses vous sembleront arbitraires dans Linux : le nom des répertoires, les commandes de deux lettres, l'habitude de tout stocker dans des fichiers texte. Elles ne le sont pas. Elles découlent de décisions prises il y a plus de cinquante ans, et qui ont si bien résisté qu'on ne les a jamais remises en cause.

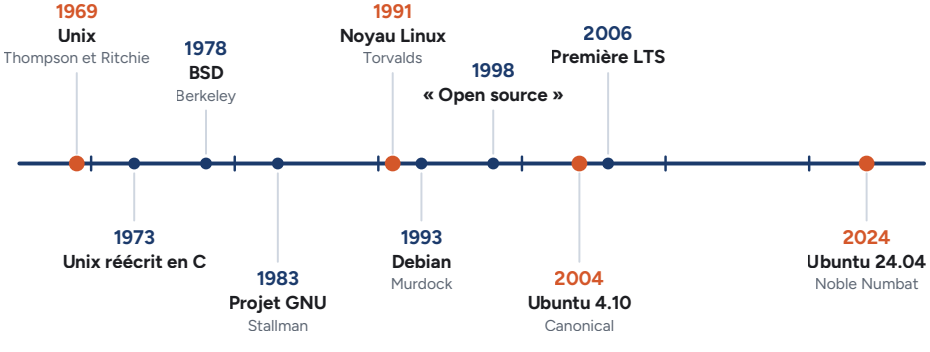


Figure 1.2 — D'Unix à Ubuntu : les jalons à retenir.

Multics, puis Unix

Au milieu des années 1960, le MIT, General Electric et les laboratoires Bell d'AT&T s'associent pour construire Multics, un système très ambitieux : plusieurs utilisa-

teurs à la fois, un accès à distance, une sécurité soignée. Trop ambitieux pour le matériel de l'époque. Le projet s'enlise et les laboratoires Bell s'en retirent en 1969.

Deux de leurs chercheurs, Ken Thompson et Dennis Ritchie, en gardent des idées et une certaine frustration. Sur un mini-ordinateur PDP-7 dont personne ne se sert, ils écrivent un système bien plus modeste. Thompson y tenait d'autant plus qu'il voulait continuer à faire tourner *Space Travel*, un jeu de sa composition. Un collègue baptise le résultat « Unics », par dérision envers Multics. L'orthographe devient vite Unix.

Le langage C et la portabilité

Le premier Unix est écrit en assembleur, un langage propre à chaque modèle de processeur. Changer de machine oblige à tout réécrire. En 1972, Dennis Ritchie met au point le langage C, presque aussi efficace que l'assembleur mais indépendant du matériel. L'année suivante, Unix est réécrit en C.

La conséquence est considérable. Pour porter Unix sur une nouvelle machine, il suffit désormais d'y disposer d'un compilateur C. C'est ce qui explique qu'on trouve aujourd'hui des descendants d'Unix partout, de la box Internet au supercalculateur. Le noyau Linux est toujours écrit en C, pour l'essentiel.

Les universités et BSD

AT&T détient alors le monopole du téléphone aux États-Unis. En contrepartie, un accord passé avec la justice en 1956 lui interdit de vendre autre chose que du téléphone. L'entreprise ne peut donc pas commercialiser Unix. Elle le cède aux universités pour une somme symbolique, code source compris.

Toute une génération d'étudiants apprend ainsi l'informatique en lisant le code d'Unix, en le modifiant et en s'échangeant les améliorations. À Berkeley, Bill Joy rassemble les siennes et les diffuse à partir de 1978 sous le nom de BSD (*Berkeley Software Distribution*). On doit à BSD l'éditeur `vi`, la mémoire virtuelle et l'implémentation de TCP/IP sur laquelle Internet s'est construit. FreeBSD, OpenBSD et NetBSD en descendent directement.

La guerre des Unix

En 1982, AT&T accepte d'être démantelée, ce qui sera effectif le 1er janvier 1984. Elle y gagne le droit de vendre du logiciel et commercialise aussitôt Unix sous le nom de System V, à un prix élevé. Sun, HP, IBM et d'autres constructeurs vendent chacun leur variante, incompatible avec celle du voisin.

Cet éparpillement a coûté cher à Unix. Il a laissé le champ libre à Windows NT sur les serveurs d'entreprise dans les années 1990. Et il a fait naître l'envie d'un Unix unique, libre et accessible à tous.

GNU et le logiciel libre

Le 27 septembre 1983, Richard Stallman, programmeur au laboratoire d'intelligence artificielle du MIT, annonce le projet GNU. Il veut un système complet, compatible avec Unix, que chacun puisse utiliser, étudier, modifier et redistribuer. Le nom est une plaisanterie d'informaticien : *GNU's Not Unix*, un sigle qui se contient lui-même.

Stallman fonde la Free Software Foundation en 1985 et publie en 1989 la première version de la licence publique générale GNU, la GPL. L'idée est habile. La GPL s'appuie sur le droit d'auteur, non pour restreindre la copie mais pour la garantir : quiconque distribue un logiciel sous GPL, modifié ou non, doit fournir le code source et accorder les mêmes droits à ses destinataires. On appelle ce mécanisme le *copyleft*.

En quelques années, GNU produit le compilateur GCC, l'éditeur Emacs, le shell Bash, la bibliothèque C et des centaines d'utilitaires. Au début des années 1990, il ne lui manque plus que le noyau.

Linux

Pendant ce temps, à Amsterdam, le professeur Andrew Tanenbaum écrit Minix, un petit Unix destiné à ses étudiants et publié en 1987 avec son manuel. Minix est simple et bien documenté, mais sa licence en limite l'usage à l'enseignement.

En 1991, Linus Torvalds, étudiant à l'université d'Helsinki, vient de s'offrir un PC à processeur 386. Minix le laisse sur sa faim. Il commence à écrire son propre noyau et l'annonce le 25 août sur le groupe de discussion `comp.os.minix` :

> *Hello everybody out there using minix — I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.*
>
> Linus Torvalds, 25 août 1991

« Juste un passe-temps, rien de gros ni de professionnel comme GNU. » On a rarement vu prévision plus fausse.

Les premières versions interdisent toute diffusion commerciale. Début 1992, avec la version 0.12, Torvalds adopte la GPL. Le noyau peut alors être distribué avec les ou-

tils GNU, publiés sous la même licence. GNU avait tout sauf un noyau, Linux n'était qu'un noyau : le système libre complet existe enfin.

> **REPÈRE HISTORIQUE · « Linux is obsolete »**
>
> En janvier 1992, Tanenbaum publie sur Usenet un message resté célèbre, intitulé *LINUX is obsolete*. Il reproche à Linux d'être un noyau monolithique, d'un seul tenant, alors que l'avenir appartiendrait selon lui aux micro-noyaux. Torvalds répond sans ménagement. Trente ans plus tard, Linux fait tourner l'essentiel des serveurs de la planète et tous les téléphones Android. Les micro-noyaux, eux, ont trouvé leur place dans l'embarqué. Chacun avait un peu raison.

Les distributions

Un noyau et des outils éparpillés sur des serveurs FTP, cela ne fait pas un système qu'on installe. Dès 1992, des passionnés assemblent le tout sur des jeux de disquettes. SLS ouvre la voie, suivi en 1993 par Slackware, qui existe toujours.

En août 1993, Ian Murdock lance Debian. Le projet apporte ce qui fait encore la force des distributions : des logiciels découpés en *paquets*, des dépendances résolues automatiquement, et une organisation entièrement bénévole, encadrée par un contrat social qui l'engage envers le logiciel libre. En 1994 paraît Red Hat Linux, avec son propre format de paquets, RPM, et une ambition commerciale assumée.

Presque toutes les distributions actuelles descendent de l'une de ces deux lignées. Ubuntu est une fille de Debian.

Dix ans pour s'imposer

Entre 1995 et 2004, Linux quitte les chambres d'étudiants pour les salles de serveurs.

- **1995.** Naissance du serveur web Apache, qui devient l'année suivante le plus utilisé au monde, le plus souvent sur Linux.
- **1996-1997.** Lancement de KDE puis de GNOME, les deux grands environnements de bureau.
- **1998.** Netscape publie le code source de son navigateur, qui donnera Firefox. Le terme *open source* apparaît la même année.
- **1999.** Red Hat entre en Bourse. Le logiciel libre devient une affaire sérieuse aux yeux des financiers.
- **2000.** IBM annonce un investissement d'un milliard de dollars dans Linux.

- **2003.** Red Hat sépare son offre en deux : Red Hat Enterprise Linux pour les entreprises, Fedora pour la communauté.
- **2004.** Première version d'Ubuntu.

1.3 Logiciel libre et open source

Ubuntu est un logiciel libre. Ce n'est pas un slogan : ce statut détermine ce que vous avez le droit d'en faire, y compris dans un cadre professionnel. Vous pouvez l'installer sur autant de machines que vous voulez, le modifier et vendre un service construit dessus. Voyons pourquoi.

Les quatre libertés

La Free Software Foundation définit un logiciel libre par quatre libertés accordées à l'utilisateur. Les informaticiens comptant à partir de zéro, elles sont numérotées de 0 à 3 :

- **liberté 0** : faire fonctionner le programme comme on veut, pour n'importe quel usage ;
- **liberté 1** : étudier son fonctionnement et le modifier, ce qui suppose l'accès au code source ;
- **liberté 2** : en redistribuer des copies ;
- **liberté 3** : distribuer ses versions modifiées, pour en faire profiter tout le monde.

Qu'une seule manque, et le logiciel n'est pas libre.

> **À RETENIR · Libre ne veut pas dire gratuit**
>
> En anglais, *free* signifie à la fois « libre » et « gratuit », d'où une confusion tenace. La Free Software Foundation la dissipe ainsi : « "logiciel libre" fait référence à la liberté, pas au prix (pour comprendre ce concept, vous devez penser à "liberté d'expression", pas à "entrée libre") ».
>
> Un logiciel libre peut se vendre : Red Hat facture cher l'abonnement à sa distribution. À l'inverse, WhatsApp et Gmail sont gratuits sans être libres, car vous ne pouvez ni lire leur code ni le modifier.

Open source : la même chose, vue autrement

Début 1998, Netscape vient d'annoncer l'ouverture du code de son navigateur. Quelques figures du logiciel libre se réunissent le 3 février à Palo Alto pour en tirer parti. Elles jugent l'expression *free software* trop ambiguë et trop militante pour convaincre les entreprises. Christine Peterson propose *open source*. Eric Raymond et Bruce Perens fondent dans la foulée l'Open Source Initiative (OSI), qui se charge de définir le terme.

Les deux définitions recouvrent, à peu de chose près, les mêmes logiciels. La différence tient à l'argumentaire. Le mouvement du libre parle d'éthique et de droits de l'utilisateur. Les partisans de l'open source parlent de qualité du code et d'efficacité du développement en commun. Dans ce livre, les deux termes sont employés indifféremment.

Deux familles de licences

Une licence est le texte juridique qui accompagne un logiciel et précise ce que vous pouvez en faire. Il en existe des dizaines, qui se rangent en deux familles.

Les licences à **copyleft** exigent que toute version modifiée, si elle est distribuée, le soit sous la même licence. Reprenez du code sous GPL dans votre programme, distribuez ce programme, et vous devez le publier lui aussi sous GPL. Le code reste libre, quoi qu'il arrive.

Les licences **permissives** n'imposent presque rien. Vous pouvez reprendre du code sous licence MIT ou BSD dans un produit fermé et commercial, sans rien publier. Il suffit de conserver la mention des auteurs.

Licence	Famille	Ce qu'elle impose	Exemples
GPL v2	Copyleft fort	Toute version modifiée et distribuée reste sous GPL, code source fourni.	Noyau Linux, Git
GPL v3	Copyleft fort	Comme la v2, avec des protections contre les brevets et contre le matériel verrouillé.	Bash, GCC, GIMP
LGPL	Copyleft faible	Un programme fermé peut utiliser la bibliothèque ; les modifications de la bibliothèque elle-même restent libres.	glibc, GTK
AGPL v3	Copyleft réseau	Comme la GPL v3, étendue aux logiciels proposés sous forme de service en ligne.	Nextcloud, Mastodon, Grafana (depuis 2021)

Licence	Famille	Ce qu'elle impose	Exemples
MPL 2.0	Copyleft faible	Seuls les fichiers modifiés doivent rester libres.	Firefox, Thunderbird, LibreOffice
Apache 2.0	Permissive	Conserver les mentions ; comporte une clause explicite sur les brevets.	Kubernetes, Android (en grande partie)
MIT	Permissive	Conserver la mention de copyright, rien d'autre.	Node.js, Ruby on Rails, jQuery
BSD	Permissive	Proche de MIT ; la version à trois clauses interdit d'utiliser le nom des auteurs à des fins de promotion.	FreeBSD, Nginx, Go

ATTENTION · Lisez la licence avant de réutiliser du code

Intégrer une bibliothèque sous AGPL dans un service en ligne commercial peut vous obliger à en publier tout le code. Et deux licences libres ne sont pas toujours compatibles entre elles : du code placé sous « GPL v2 uniquement », comme le noyau Linux, ne peut pas être mélangé à du code sous GPL v3. Les entreprises tiennent des listes de licences autorisées pour cette raison.

Ce que cela change pour vous

Ubuntu rassemble des milliers de logiciels, chacun sous sa propre licence. Canonical les répartit en quatre dépôts, que vous retrouverez au chapitre 7 : `main` (logiciels libres maintenus par Canonical), `universe` (logiciels libres maintenus par la communauté), `restricted` (pilotes propriétaires, par exemple pour les cartes Nvidia) et `multiverse` (logiciels soumis à des restrictions de licence).

Au quotidien, ces libertés ont des effets très concrets.

Le code est consultable. Quand une commande se comporte de façon inattendue, vous pouvez aller lire comment elle est écrite. Peu de gens le font. Savoir que c'est possible change pourtant la manière de chercher une panne.

Rien ne vous retient. Si l'orientation prise par Canonical cesse de vous convenir, vous passez à Debian ou à une autre distribution, et vos connaissances vous suivent. Il n'y a ni clé d'activation ni contrat.

Les correctifs sont publics. Quand une faille est corrigée, le correctif est publié et chacun peut le vérifier. Cela ne rend pas le logiciel libre invulnérable, mais on n'a pas à croire l'éditeur sur parole.

Il y a une contrepartie, et il faut la connaître avant de se lancer : certains logiciels n'existent pas sous Linux. La suite Adobe et Microsoft Office en version de bureau sont les exemples habituels. On trouve presque toujours une solution de rechange, mais pas toujours sans effort.

1.4 L'écosystème Linux aujourd'hui

On compte plusieurs centaines de distributions Linux actives. Vue de loin, cette abondance décourage. De près, elle s'organise en quelques familles, et les différences se ramènent à un petit nombre de choix.

Ce qui distingue deux distributions

Toutes partagent le noyau Linux et les outils GNU. Elles se séparent sur quatre points :

- **le gestionnaire de paquets**, c'est-à-dire l'outil qui installe et met à jour les logiciels : APT chez Debian et Ubuntu, DNF chez Fedora et Red Hat, pacman chez Arch ;
- **le rythme de publication** : versions espacées et longuement testées, ou flux continu de mises à jour (on parle alors de *rolling release*) ;
- **les choix par défaut** : environnement de bureau, système d'init, outils de sécurité ;
- **le modèle** : projet bénévole, entreprise, ou les deux à la fois.

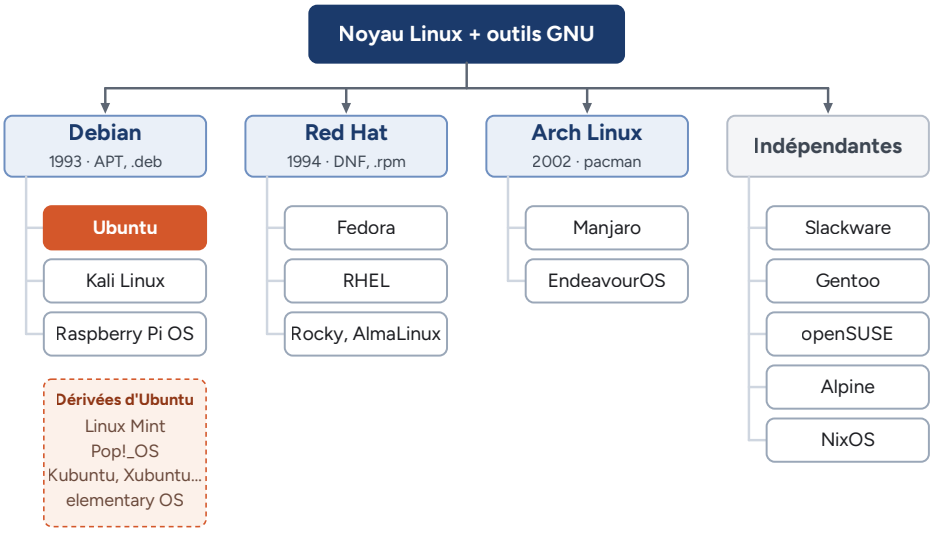


Figure 1.3 — Les grandes familles de distributions Linux. Ubuntu descend de Debian et a elle-même de nombreux dérivés.

La famille Debian

Debian existe depuis 1993. Aucune entreprise ne la dirige et son attachement au logiciel libre est inscrit dans ses textes fondateurs. Elle publie une version stable tous les deux ans environ, après de longs mois de gel et de tests. C'est une valeur sûre sur un serveur. Son gestionnaire de paquets, APT, et son format `.deb` sont aussi ceux d'Ubuntu.

Parmi ses descendants directs ou indirects : **Ubuntu**, bien sûr ; **Linux Mint**, construite sur Ubuntu et appréciée des personnes qui viennent de Windows ; **Pop!_OS**, de l'assembleur américain System76 ; **Kali Linux**, spécialisée dans les tests d'intrusion ; **Raspberry Pi OS**, pour les cartes du même nom.

La famille Red Hat

Red Hat Enterprise Linux (RHEL) est la référence des grandes entreprises, des banques et des opérateurs télécoms. On y accède par abonnement. Son format de paquets est RPM, son gestionnaire DNF.

Autour d'elle gravitent **Fedora**, la distribution communautaire où Red Hat met au point ses nouveautés, avec une version tous les six mois, et **CentOS Stream**, qui préfigure la prochaine version mineure de RHEL. Jusqu'en 2020, CentOS était une co-

pie gratuite de RHEL, très répandue chez les hébergeurs. Quand Red Hat a changé son rôle, deux projets ont pris le relais en 2021 : **Rocky Linux** et **AlmaLinux**.

La famille Arch

Arch Linux, née en 2002, prend le contre-pied des précédentes. Elle n'a pas de versions : les mises à jour arrivent en continu. L'installation part d'un système presque vide que l'on complète soi-même. Sa documentation, le wiki Arch, est l'une des meilleures qui soient, et elle sert souvent aux utilisateurs d'autres distributions. **Manjaro** et **EndeavourOS** en proposent des variantes plus faciles à installer.

Les indépendantes

Quelques distributions ne se rattachent à aucune de ces lignées : **Slackware**, la plus ancienne encore maintenue ; **Gentoo**, où chaque logiciel est compilé sur place ; **openSUSE**, d'origine allemande ; **Alpine**, minuscule, devenue la base de très nombreuses images de conteneurs ; **NixOS**, où tout le système se décrit dans un fichier de configuration.

Gestionnaire	Format	Distributions	Particularité
APT	`.deb`	Debian, Ubuntu et leurs dérivées	Le plus vaste catalogue de paquets
DNF	`.rpm`	Fedora, RHEL, Rocky Linux, AlmaLinux	Le standard du monde Red Hat
zypper	`.rpm`	openSUSE, SUSE Linux Enterprise	Retour arrière grâce aux instantanés Btrfs
pacman	`.pkg.tar.zst`	Arch, Manjaro, EndeavourOS	Rapide, mises à jour en continu
Portage	sources	Gentoo	Tout est compilé sur la machine
apk	`.apk`	Alpine	Minimal, pensé pour les conteneurs
Nix	dérivations	NixOS, utilisable ailleurs	Système décrit dans un fichier, reproductible
Snap	`.snap`	Ubuntu surtout	Applications isolées, mises à jour automatiques
Flatpak	Flatpak	Toutes, bureau surtout	Applications isolées, indépendant de la distribution

> **ASTUCE · Par où commencer ?**
>
> Pour un poste de travail : Ubuntu, Linux Mint ou Fedora. Pour un serveur : Ubuntu Server, Debian, ou Rocky Linux si votre entreprise travaille avec Red Hat. Pour comprendre comment un système est construit, pièce par pièce : Arch. Ce livre a fait son choix, mais ce que vous y apprendrez s'applique pour l'essentiel aux autres.

1.5 Pourquoi Ubuntu ?

Canonical et Mark Shuttleworth

Mark Shuttleworth est un entrepreneur sud-africain. En 1999, il vend à VeriSign sa société Thawte, spécialisée dans les certificats numériques, pour 575 millions de dollars. Ancien développeur Debian, il fonde Canonical en 2004 avec un objectif simple à formuler : partir de la base technique de Debian et en faire un système que n'importe qui puisse installer et utiliser, avec une entreprise derrière pour le support.

Le mot *ubuntu* vient des langues nguni d'Afrique australe. Il exprime l'idée qu'on ne devient pleinement humain qu'à travers les autres, ce que l'on résume souvent par « je suis parce que nous sommes ». Le slogan des premières années en découlait : *Linux for human beings*.

Les noms de code et le rythme des versions

La première version sort le 20 octobre 2004. Elle porte le numéro 4.10, pour l'année et le mois, et le nom de code *Warty Warthog*, le phacochère verruqueux. La convention est restée : un adjectif et un animal qui commencent par la même lettre. À partir de la 6.06 (*Dapper Drake*), ces lettres se suivent dans l'ordre alphabétique. Arrivé à Z avec la 17.04 (*Zesty Zapus*), le compteur est reparti de A. La 24.04 s'appelle *Noble Numbat*, la 26.04 *Resolute Raccoon*.

Le calendrier n'a pas changé depuis l'origine : une version tous les six mois, en avril et en octobre.

Le modèle LTS

En juin 2006, Canonical publie la 6.06 avec une mention nouvelle : **LTS**, pour *Long Term Support*. Tous les deux ans, une version est maintenue bien plus longtemps que les autres. Les entreprises n'en demandaient pas davantage. On ne réinstalle pas un serveur de production tous les six mois.

Les règles actuelles tiennent en quatre points :

- une version LTS sort en avril des années paires : 20.04, 22.04, 24.04, 26.04 ;
- elle reçoit gratuitement cinq ans de correctifs de sécurité ;
- l'abonnement **Ubuntu Pro** porte cette durée à dix ans et l'étend aux milliers de paquets du dépôt `universe`. Il est gratuit jusqu'à cinq machines, pour un usage personnel ou une petite structure ;
- les versions intermédiaires (24.10, 25.04, 25.10…) ne sont maintenues que neuf mois. Elles servent à préparer la LTS suivante.

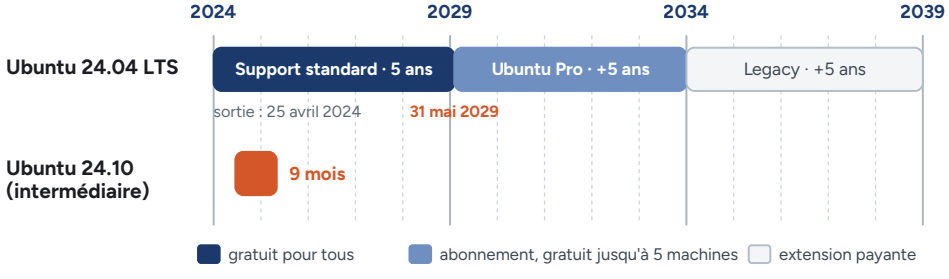


Figure 1.4 — Cycle de vie d'Ubuntu 24.04 LTS, comparé à celui d'une version intermédiaire.

Version	Nom de code	Sortie	Support standard jusqu'en	Avec Ubuntu Pro jusqu'en
14.04 LTS	Trusty Tahr	avril 2014	2019	2024
16.04 LTS	Xenial Xerus	avril 2016	2021	2026
18.04 LTS	Bionic Beaver	avril 2018	2023	2028
20.04 LTS	Focal Fossa	avril 2020	2025	2030
22.04 LTS	Jammy Jellyfish	avril 2022	2027	2032
24.04 LTS	Noble Numbat	avril 2024	2029	2034
26.04 LTS	Resolute Raccoon	avril 2026	2031	2036

> **NOTE · Et Ubuntu 26.04 ?**
>
> Ubuntu 26.04 LTS est sortie le 23 avril 2026. Ce livre reste consacré à la 24.04, pour une raison pratique : c'est elle qui équipe la plupart des serveurs en service, et elle sera maintenue jusqu'au 31 mai 2029. Presque tout ce que vous lirez ici vaut aussi pour la 26.04. Les commandes, l'organisation du système et les outils d'administration sont les mêmes d'une LTS à l'autre.

Qui fait vivre Ubuntu

Canonical est une société privée, non cotée. Elle ne vend pas Ubuntu. Elle vend ce qui l'entoure : les abonnements Ubuntu Pro, le support technique, les images certifiées chez les fournisseurs de cloud, des services d'infogérance, et Ubuntu Core aux fabricants d'objets connectés.

Autour d'elle, une communauté nombreuse répond aux questions et documente le système. Trois adresses à connaître :

- `askubuntu.com`, un site de questions et réponses sur le modèle de Stack Overflow ;
- `discourse.ubuntu.com`, le forum officiel, qui a remplacé en 2025 les anciens Ubuntu Forums ;
- `doc.ubuntu-fr.org` et `forum.ubuntu-fr.org`, la documentation et le forum de la communauté francophone.

Devant un message d'erreur, copiez-le tel quel dans un moteur de recherche. Neuf fois sur dix, quelqu'un a déjà posé la question.

Les critiques, sans détour

Ubuntu a ses détracteurs, et ils n'ont pas toujours tort.

La recherche Amazon. En 2012, la version 12.10 envoie à Amazon ce que l'utilisateur tape dans le lanceur d'applications, pour afficher des produits à acheter. Le tollé est général. La fonction est désactivée par défaut dans la 16.04, puis retirée.

Unity. De 2011 à 2017, Canonical développe sa propre interface de bureau au lieu de GNOME. La communauté se divise, le projet est abandonné en 2017 et Ubuntu revient à GNOME. Unity survit dans une saveur communautaire.

Snap. Canonical pousse son format de paquets, les *snaps*, isolés du reste du système et mis à jour automatiquement. Ses adversaires lui reprochent un démarrage plus lent des applications et, surtout, un magasin central dont le logiciel serveur n'est

pas libre et que seule Canonical exploite. Sur Ubuntu 24.04, Firefox et Thunderbird ne sont fournis que sous cette forme. Le chapitre 7 présente les arguments des deux camps.

Rien de tout cela n'empêche Ubuntu d'être un excellent choix. Mieux vaut simplement le savoir : Canonical est une entreprise, elle prend des décisions d'entreprise, et il lui arrive de revenir dessus.

1.6 Ubuntu 24.04 LTS « Noble Numbat »

Ubuntu 24.04 LTS est sortie le 25 avril 2024. Le numbat qui lui donne son nom est un petit marsupial rayé d'Australie-Occidentale, dont il est l'emblème. Il se nourrit presque uniquement de termites. C'est lui qui ouvre ce chapitre.

Sa préparation a été mouvementée. Fin mars 2024, une porte dérobée est découverte dans `xz`, un outil de compression présent sur tous les systèmes Linux. Canonical retarde la version bêta d'une semaine, le temps de recompiler les paquets concernés. Puis la première mise à jour intermédiaire, la 24.04.1, glisse du 15 au 29 août 2024 pour corriger des défauts de la mise à niveau depuis la 22.04.

Ce qui change par rapport à la 22.04

Le noyau 6.8. Il prend en charge le matériel sorti depuis 2022. Les installations Desktop reçoivent ensuite des noyaux plus récents par le mécanisme HWE (*Hardware Enablement*) : à l'automne 2026, c'est le noyau 7.0, celui d'Ubuntu 26.04.

GNOME 46. L'application Fichiers gagne une recherche globale, les Paramètres sont réorganisés et les notifications peuvent se déplier pour afficher leur contenu.

Un nouvel installateur pour le poste de travail. Il remplace Ubiquity, en service depuis 2006, et repose sur le même moteur que l'installateur serveur. L'installation par défaut est désormais minimale : un navigateur et les outils de base. LibreOffice et Thunderbird font partie d'une « sélection étendue », à cocher si on les veut. L'installateur sait aussi chiffrer le disque en confiant la clé à la puce TPM de la machine, mais cette option est encore expérimentale.

Une logithèque refaite. L'App Center remplace Ubuntu Software.

Netplan sur le poste de travail. La configuration réseau passe par Netplan sur toutes les éditions. Sur le bureau, NetworkManager reste aux commandes, mais ses réglages sont enregistrés au format Netplan. Le chapitre 18 y est consacré.

Les sources APT au format deb822. La liste des dépôts quitte `/etc/apt/sources.list` pour `/etc/apt/sources.list.d/ubuntu.sources`, dans un format plus lisible. Détails au chapitre 7.

AppArmor 4.0. Le noyau d'Ubuntu restreint désormais les *espaces de noms utilisateur non privilégiés*, un mécanisme dont beaucoup de failles récentes se sont servies. Certaines applications tierces doivent être adaptées. Le chapitre 22 explique comment.

Le tableau suivant résume les versions des principaux composants. Elles ont été relevées sur un système à jour.

Composant	Ubuntu 22.04	Ubuntu 24.04
Noyau Linux	5.15	6.8
GNOME	42	46
systemd	249	255
OpenSSH	8.9	9.6
Python	3.10	3.12
PHP	8.1	8.3
PostgreSQL	14	16
AppArmor	3.0	4.0
Sources APT	`sources.list`	`ubuntu.sources` (deb822)
Installation Desktop par défaut	complète	minimale
Fin du support standard	2027	31 mai 2029

> **ASTUCE · Vous venez de la 22.04 ?**
>
> La mise à niveau se fait avec la commande `do-release-upgrade`. Sur un serveur en production, sauvegardez d'abord, lisez les notes de publication et répétez l'opération sur une copie. Pour apprendre, une installation neuve est plus simple et plus propre.

1.7 Quelle édition choisir ?

Ubuntu se décline en plusieurs éditions. Elles partagent le même socle et les mêmes dépôts. Seuls changent l'installateur et les logiciels installés au départ.

Ubuntu Desktop

C'est l'édition pour ordinateur personnel : GNOME, Firefox, l'App Center et les outils courants. Canonical demande au minimum un processeur double cœur à 2 GHz, 4 Go de mémoire et 25 Go de disque. Avec 8 Go de mémoire, on travaille confortablement.

Ubuntu Server

Pas d'interface graphique, un installateur en mode texte et le strict nécessaire : SSH pour l'administration à distance, Netplan pour le réseau, `cloud-init` pour l'installation automatisée. Ni serveur web ni base de données. Vous ajoutez ce dont vous avez besoin, et seulement cela. Une machine virtuelle de 2 Go de mémoire suffit pour tous les exercices de ce livre.

Les images cloud

Chez AWS, Azure, Google Cloud, OVHcloud, Hetzner ou Contabo, on n'installe rien. On choisit une image Ubuntu déjà préparée, et la machine démarre en moins d'une minute. Ces images sont des Ubuntu Server allégés, configurés au premier démarrage par `cloud-init`.

Ubuntu Core

Ubuntu Core vise les objets connectés et les équipements industriels. Le système y est en lecture seule et composé uniquement de snaps, noyau compris. Les mises à jour s'appliquent en bloc, avec retour automatique à la version précédente en cas d'échec. Son fonctionnement s'écarte trop d'un Linux classique pour servir de terrain d'apprentissage. Ce livre ne le traite pas.

Les saveurs

Les *saveurs* sont des variantes officielles d'Ubuntu Desktop, maintenues par la communauté, avec un autre environnement de bureau. Elles utilisent les mêmes dépôts.

Saveur	Environnement	Pour qui
Kubuntu	KDE Plasma	Ceux qui aiment tout régler ; proche des habitudes de Windows
Xubuntu	Xfce	Machines modestes, bureau classique
Lubuntu	LXQt	Machines anciennes
Ubuntu MATE	MATE	Les nostalgiques de GNOME 2
Ubuntu Budgie	Budgie	Bureau épuré
Ubuntu Cinnamon	Cinnamon	L'ergonomie de Linux Mint
Ubuntu Unity	Unity	Les fidèles de l'ancienne interface d'Ubuntu
Ubuntu Studio	KDE Plasma	Création audio, vidéo et graphique
Edubuntu	GNOME	Écoles
Ubuntu Kylin	UKUI	Utilisateurs chinois

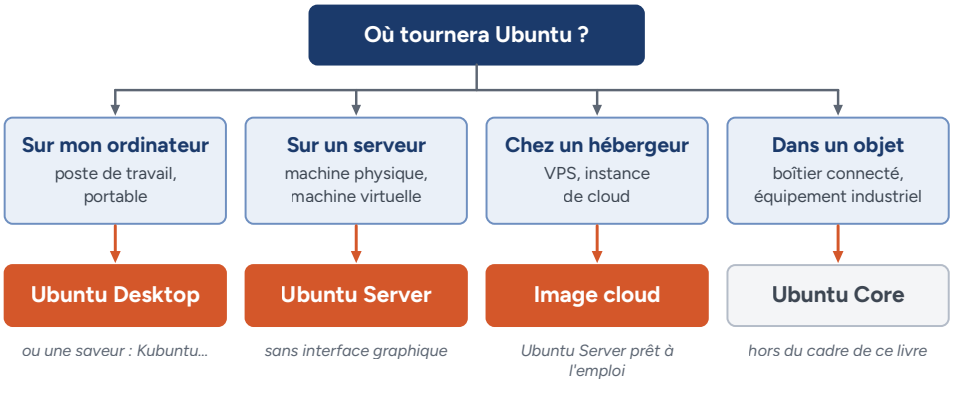


Figure 1.5 — Quelle édition pour quel usage ?

À RETENIR · Ce qu'il vous faut pour suivre ce livre

Deux machines virtuelles : l'une avec Ubuntu Desktop 24.04, l'autre avec Ubuntu Server 24.04. Vous aurez les deux visages du système sous la main et vous pourrez tout casser sans conséquence. Le chapitre 2 vous guide pour les installer.

En résumé

- Un système d'exploitation masque le matériel, partage les ressources et donne à l'utilisateur les moyens de piloter la machine.
- Linux est un noyau. Associé aux outils GNU, il forme un système complet, qu'une distribution assemble et maintient.
- Unix naît en 1969, GNU en 1983, Linux en 1991, Debian en 1993, Ubuntu en 2004.
- Un logiciel libre garantit quatre libertés : utiliser, étudier, redistribuer, modifier. Libre ne signifie pas gratuit.
- Les licences à copyleft (GPL) imposent de conserver la même licence ; les licences permissives (MIT, BSD, Apache) n'imposent presque rien.
- Ubuntu descend de Debian. Une version LTS sort tous les deux ans et reçoit cinq ans de correctifs, dix avec Ubuntu Pro.
- Ubuntu 24.04 LTS est maintenue jusqu'au 31 mai 2029. Elle apporte le noyau 6.8, GNOME 46, un nouvel installateur, Netplan sur toutes les éditions et AppArmor 4.0.
- Desktop pour un poste de travail, Server pour tout le reste.

Exercices

Aucune manipulation ici. Ces exercices servent à vérifier que le vocabulaire est en place. Répondez par écrit, sans revenir en arrière, puis comparez avec le chapitre.

Exercice 1 — Vocabulaire

Définissez en une ou deux phrases : système d'exploitation, noyau, distribution, shell, environnement de bureau, copyleft, version LTS, saveur.

Exercice 2 — Filiation

Retracez en dix lignes le chemin qui va de Multics à Ubuntu. Quel rôle ont joué Ken Thompson, Richard Stallman, Linus Torvalds, Ian Murdock et Mark Shuttleworth ?

Exercice 3 — Choisir une licence

Quelle licence choisiriez-vous dans chacun des cas suivants, et pourquoi ?

1. Une bibliothèque de chiffrement que vous voulez voir adoptée partout, y compris dans des produits fermés.
2. Un logiciel de réseau social que personne ne doit pouvoir exploiter en ligne sans publier ses modifications.
3. Un outil dont toute version modifiée doit rester libre, sans contrainte particulière sur l'usage en ligne.

Exercice 4 — Choisir une édition

Quelle édition d'Ubuntu, ou quelle autre distribution, recommanderiez-vous ?

1. Une personne qui vient de Windows et veut essayer Linux sur un portable de 2016 doté de 4 Go de mémoire.
2. Une PME qui installe un serveur pour son site web et ses fichiers partagés.
3. Une jeune entreprise qui déploie une application web chez un fournisseur de cloud.
4. Un industriel qui équipe des milliers de boîtiers connectés, mis à jour à distance.
5. Une développeuse qui veut un poste de travail moderne pour du développement web.

Exercice 5 — Et vous ?

Notez quelque part ce que vous voulez savoir faire en refermant ce livre, et dans quel cadre vous utiliserez Ubuntu. Vous relirez ces lignes à la fin.

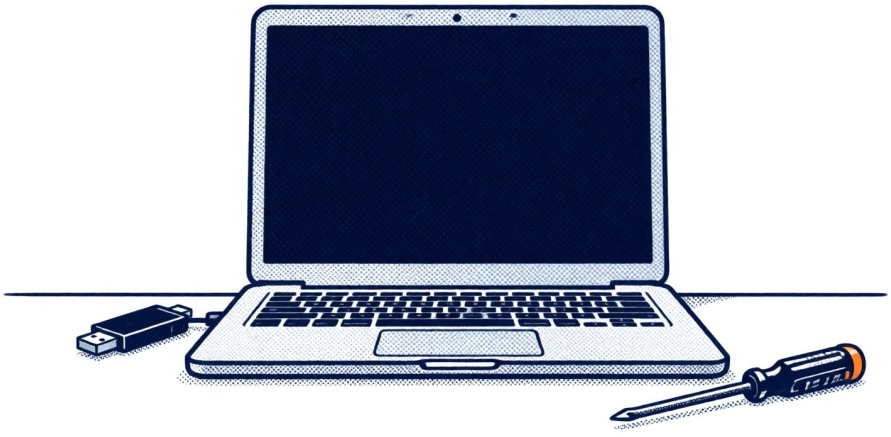
Pour aller plus loin

Histoire. *Just for Fun*, de Linus Torvalds et David Diamond : le récit de la naissance de Linux par son auteur, facile à lire. *La Cathédrale et le Bazar*, d'Eric Raymond : l'essai qui a théorisé le développement open source. *A Quarter Century of UNIX*, de Peter Salus, pour l'histoire d'Unix racontée par ceux qui l'ont faite.

Licences. `gnu.org/philosophy` pour les textes de référence du logiciel libre, en français. `opensource.org` pour la définition de l'open source et la liste des licences approuvées. `choosealicense.com` pour choisir une licence en cinq minutes.

Ubuntu. `ubuntu.com/about/release-cycle` pour le calendrier des versions et les durées de support. `documentation.ubuntu.com/release-notes/24.04/` pour les notes de publication. `doc.ubuntu-fr.org` pour la documentation francophone.

Actualité. `lwn.net`, la référence sur le noyau et l'écosystème. `omgubuntu.co.uk` pour l'actualité d'Ubuntu. `linuxfr.org` pour une veille en français.



CHAPITRE 2

Installer Ubuntu 24.04

Sur un PC, dans une machine virtuelle, sous Windows ou chez un hébergeur : choisir sa voie, puis passer à l'acte.

DANS CE CHAPITRE

- Choisir entre installation sur machine physique, machine virtuelle, WSL, serveur loué et double amorçage.
- Télécharger une image d'Ubuntu, vérifier son intégrité et préparer une clé USB.
- Installer Ubuntu Desktop, avec ou sans chiffrement du disque.
- Installer Ubuntu Server et s'y connecter en SSH.
- Créer une machine virtuelle avec VirtualBox, KVM, UTM ou Multipass.
- Louer un serveur chez un hébergeur et y ouvrir une première session.
- Comprendre le principe de l'installation automatisée.
- Faire les premiers gestes après l'installation.

> **AVANT DE COMMENCER**
>
> Un ordinateur de moins de dix ans et une connexion Internet. Selon la voie choisie : une clé USB de 8 Go, un logiciel de virtualisation ou un compte chez un hébergeur.
>
> Ce chapitre décrit plusieurs chemins. Lisez la section 2.1 pour choisir le vôtre, allez directement à la section correspondante, puis terminez par la section 2.10, qui concerne tout le monde. Si vous débutez, prenez la machine virtuelle (section 2.5) : on ne risque rien.

2.1 Préparer le terrain

Avant de télécharger quoi que ce soit, il faut décider où Ubuntu va tourner. La décision engage la suite, et certaines erreurs à cette étape ne se rattrapent pas : un disque effacé reste effacé.

Cinq façons de faire tourner Ubuntu

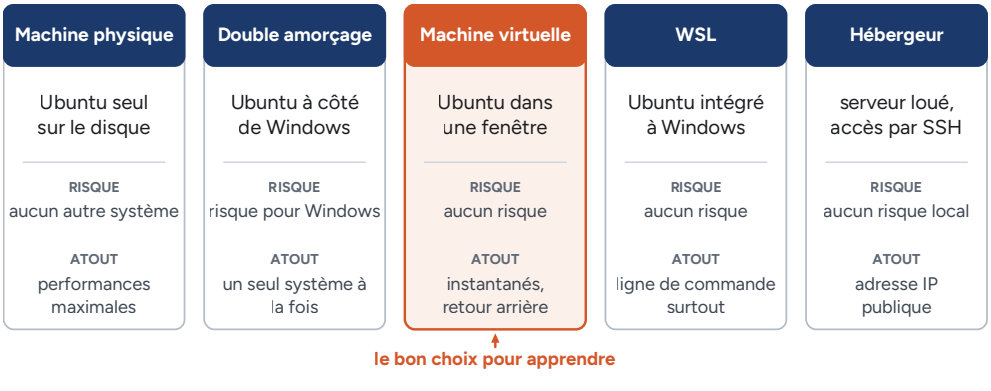


Figure 2.1 — Les cinq façons d'installer Ubuntu. Pour apprendre, la machine virtuelle est le meilleur compromis.

Sur une machine physique, seul. Ubuntu occupe tout le disque. Les performances sont maximales et vous vivez sous Linux au quotidien. C'est le bon choix pour un vieux portable à recycler, et c'est le cas normal d'un serveur.

En double amorçage avec Windows. Les deux systèmes cohabitent sur le même disque et vous choisissez au démarrage. C'est utile quand un logiciel vous retient sous Windows. C'est aussi la manipulation la plus risquée de ce chapitre, car une erreur de partitionnement peut effacer Windows. Elle est traitée en section 2.9.

Dans une machine virtuelle. Ubuntu tourne dans une fenêtre de votre système actuel. Vous pouvez prendre un instantané, tout casser, revenir en arrière en quelques secondes et recommencer. Il faut seulement assez de mémoire pour deux systèmes à la fois : 8 Go au minimum, 16 Go pour être à l'aise.

Avec WSL, sous Windows. Windows 10 et 11 savent faire tourner un Ubuntu en ligne de commande, intégré au système. C'est pratique pour apprendre le shell ou pour développer. Ce n'est pas tout à fait un Ubuntu ordinaire, comme on le verra en section 2.6.

Chez un hébergeur. Vous louez un serveur déjà installé, auquel vous accédez à distance par SSH. Pour quelques euros par mois, vous disposez d'une vraie machine sur Internet, avec une adresse IP publique. C'est la voie normale pour héberger un site ou un service.

Ce qu'il faut comme matériel

Ressource	Desktop : minimum	Desktop : confortable	Server
Processeur	64 bits, double cœur, 2 GHz	quatre cœurs	64 bits, un cœur suffit
Mémoire	4 Go	8 Go ou plus	1 Go pour démarrer, 2 Go pour être à l'aise
Disque	25 Go	50 Go sur SSD	10 Go, 25 Go conseillés
Réseau	Wi-Fi ou câble	câble pendant l'installation	câble

Les valeurs minimales pour Desktop sont celles que publie Canonical. Les autres viennent de la pratique.

Ubuntu 24.04 existe pour six architectures : amd64 (les PC et la plupart des serveurs), arm64 (Raspberry Pi 4 et 5, Mac récents en virtualisation, serveurs ARM), armhf, riscv64, ppc64el et s390x. Une image arm64 ne démarre pas sur un PC, et inversement. La commande `uname -m` affiche l'architecture d'une machine Linux ou macOS : `x86_64` pour amd64, `aarch64` ou `arm64` pour ARM.

Pour connaître votre matériel avant de commencer :

```
# Voir les informations système
msinfo32
```

```
# Ou dans PowerShell :
systeminfo | Select-String "OS Name","System Type","Total Physical Memory"
Get-ComputerInfo | Select CsProcessors, BiosBIOSVersion
```

```
# Dans le Terminal
system_profiler SPHardwareDataType
sysctl -n machdep.cpu.brand_string
sysctl -n hw.memsize
```

```
# Informations sur le CPU
lscpu

# Quantité de RAM
free -h

# Informations générales sur le matériel
sudo lshw -short

# Type d'architecture
uname -m   # renvoie x86_64, aarch64, etc.
```

Sauvegardez d'abord

> **ATTENTION · Avant de toucher à un disque qui contient des données**
>
> Copiez sur un support externe tout ce que vous ne pouvez pas vous permettre de perdre : documents, photos, mots de passe et favoris du navigateur, clés de licence. Faites-le même si vous êtes sûr de vous. Les installateurs sont fiables. Les doigts qui cliquent sur « Effacer le disque » au mauvais endroit le sont moins.

Et Secure Boot ?

De vieux tutoriels demandent de désactiver Secure Boot avant d'installer Linux. Avec Ubuntu, c'est inutile depuis longtemps. Le premier maillon de son démarrage, un petit programme nommé `shim`, est signé par Microsoft. Il vérifie ensuite GRUB et le noyau, signés par Canonical. Laissez donc Secure Boot activé.

Il y a une situation où il se rappellera à vous : l'installation de modules noyau qui ne viennent pas d'Ubuntu, comme le pilote propriétaire Nvidia ou ceux de VirtualBox. Ubuntu vous fera alors créer un mot de passe, puis vous demandera de le confirmer au redémarrage suivant dans un écran bleu intitulé *MOK management*. Cette étape enregistre votre propre clé de signature. Elle ne se fait qu'une fois.

2.2 Créer une clé USB d'installation

Cette section concerne l'installation sur une machine physique. Pour une machine virtuelle, le fichier image suffit, sans clé.

Télécharger l'image

Les images officielles se trouvent sur `ubuntu.com/download` et sur `releases.ubuntu.com/24.04/`. N'en prenez jamais ailleurs.

- **Ubuntu Desktop :** `ubuntu-24.04.x-desktop-amd64.iso`, environ 6 Go.
- **Ubuntu Server :** `ubuntu-24.04.x-live-server-amd64.iso`, entre 3 et 4 Go.

Le dernier chiffre est celui de la *version ponctuelle*. Tous les six mois environ, Canonical republie l'image avec les correctifs parus entre-temps : 24.04.1 en août 2024, 24.04.2 en février 2025, et ainsi de suite jusqu'à la 24.04.5. Prenez toujours la plus récente. Le système installé sera le même, avec moins de mises à jour à télécharger ensuite.

Vérifier l'image

Un téléchargement de plusieurs gigaoctets peut être corrompu en route. Plus rarement, une image peut avoir été falsifiée. Canonical publie pour cette raison, à côté des images, un fichier `SHA256SUMS` qui contient leur empreinte. Téléchargez-le dans le même dossier que l'image, puis :

```
$ sha256sum -c SHA256SUMS --ignore-missing
ubuntu-24.04.5-live-server-amd64.iso: OK
```

La commande recalcule l'empreinte de l'image et la compare à celle du fichier. Sur un système en français, elle affiche `Réussi` au lieu de `OK`. Sous macOS, remplacez `sha256sum` par `shasum -a 256`. Sous Windows, PowerShell calcule l'empreinte, que vous comparez à l'œil avec la ligne correspondante de `SHA256SUMS` :

```
Get-FileHash -Algorithm SHA256 .\ubuntu-24.04.5-live-server-amd64.iso
```

Si les empreintes diffèrent, ne serait-ce que d'un caractère, supprimez le fichier et recommencez le téléchargement.

> **NOTE · Aller plus loin : la signature GPG**
>
> Le fichier `SHA256SUMS` est lui-même signé par Canonical (fichier `SHA256SUMS.gpg`). Vérifier cette signature prouve que les empreintes n'ont pas été modifiées. La procédure est décrite pas à pas sur `ubuntu.com/tutorials/how-to-verify-ubuntu`.

Écrire l'image sur la clé

Copier le fichier sur la clé ne suffit pas : il faut l'y écrire de façon à ce que la machine puisse démarrer dessus. Quatre outils s'en chargent. Tous effacent le contenu de la clé.

Outil	Systèmes	Pour qui
balenaEtcher	Windows, macOS, Linux	Le plus simple : on choisit l'image, la clé, on valide.
Rufus	Windows	Rapide, avec davantage de réglages. Gardez les options proposées.
Ventoy	Windows, Linux	On prépare la clé une fois, puis on y copie autant d'images que l'on veut ; un menu permet de choisir au démarrage.
dd	Linux, macOS	La ligne de commande, sans filet.

Avec `dd`, tout repose sur le nom du périphérique. Repérez la clé par sa taille et son modèle, jamais par habitude :

```
$ lsblk -o NAME,SIZE,MODEL,TRAN
NAME          SIZE MODEL                     TRAN
sda          14,9G Cruzer Blade              usb
nvme0n1     476,9G SAMSUNG MZVLB512HBJQ      nvme
├─nvme0n1p1     1G                           nvme
└─nvme0n1p2 475,9G                           nvme
```

Ici, la clé est `sda` et le disque du système `nvme0n1`. Sur une machine plus ancienne, c'est souvent l'inverse : le disque interne s'appelle `sda` et la clé `sdb`.

```
# Démonter la clé si le bureau l'a montée automatiquement
sudo umount /dev/sda*

# Écrire l'image sur la clé entière (sda, et non sda1)
sudo dd if=ubuntu-24.04.5.1-desktop-amd64.iso of=/dev/sda bs=4M status=progress oflag=sync
```

> **DANGER · dd ne demande jamais confirmation**
>
> Si vous indiquez le disque du système après `of=`, `dd` l'écrase sans un mot. Relisez la ligne avant de valider, et vérifiez une seconde fois la sortie de `lsblk`.

Démarrer sur la clé

Branchez la clé, allumez la machine et appuyez tout de suite sur la touche du menu de démarrage. Elle dépend du constructeur : F12 chez Dell et Lenovo, F9 chez HP, Échap ou F8 chez Asus. Choisissez la clé dans la liste.

L'image d'Ubuntu Desktop démarre un système complet en mémoire, sans rien écrire sur le disque. Profitez-en pour vérifier que le Wi-Fi, le son et l'écran fonctionnent avant d'installer.

2.3 Installer Ubuntu Desktop

La procédure est la même sur une machine physique et dans une machine virtuelle.



Figure 2.2 — Le menu qui s'affiche au démarrage sur la clé d'installation.

Gardez la première ligne, *Try or Install Ubuntu*. Si l'écran reste noir, redémarrez et prenez *Ubuntu (safe graphics)*, qui utilise un pilote graphique générique.

Les écrans de l'installateur

L'installateur pose ses questions dans cet ordre.

1. **Langue.** Choisissez Français.

2. **Accessibilité.** Lecteur d'écran, contraste, zoom. Passez si vous n'en avez pas besoin.
3. **Clavier.** Français, variante par défaut pour un clavier AZERTY. Une zone de saisie permet d'essayer : tapez `é è à ç @`.
4. **Réseau.** Branchez le câble ou choisissez votre Wi-Fi. On peut installer sans réseau, mais on se prive alors des pilotes et des codecs proposés plus loin.
5. **Essayer ou installer.** Choisissez *Installer Ubuntu*.
6. **Type d'installation.** *Installation interactive*. L'installation automatisée est décrite en section 2.8.
7. **Applications.** La *sélection par défaut* installe un système minimal : le navigateur et les utilitaires de base. La *sélection étendue* y ajoute la suite bureautique LibreOffice, la messagerie Thunderbird et quelques autres logiciels. Prenez la première. Vous installerez le reste quand vous en aurez besoin.
8. **Logiciels propriétaires.** Deux cases : les pilotes tiers pour la carte graphique et le Wi-Fi, et la prise en charge de formats multimédias supplémentaires. Cochez les deux sur un poste de travail.
9. **Disque.** C'est l'écran décisif, détaillé ci-dessous.
10. **Compte.** Votre nom, le nom de la machine, votre identifiant et un mot de passe.
11. **Fuseau horaire.** Il est proposé d'après votre connexion. Corrigez si besoin.
12. **Récapitulatif.** Relisez, puis lancez l'installation. Comptez dix à vingt minutes.

L'écran du disque

Trois possibilités.

Effacer le disque et installer Ubuntu. L'installateur s'occupe de tout. C'est le choix normal pour une machine consacrée à Ubuntu ou pour une machine virtuelle.

Installer Ubuntu à côté de Windows. Cette option n'apparaît que si un Windows est détecté. Voir la section 2.9.

Partitionnement manuel. Vous créez vous-même les partitions. C'est utile pour séparer `/home` du système, ce qui permet de réinstaller un jour sans toucher à vos fichiers.

Avec le premier choix, le bouton *Fonctions avancées* donne accès à quatre variantes :

- **LVM.** Le disque est géré par le gestionnaire de volumes logiques, qui permettra plus tard d'agrandir ou de réorganiser l'espace sans tout refaire. Le chapitre 15 lui est consacré.

- **LVM et chiffrement.** Tout le disque, sauf la petite partition de démarrage, est chiffré avec LUKS. Une phrase secrète est demandée à chaque démarrage. Sans elle, le contenu du disque est illisible, même démonté et branché sur une autre machine. Sur un ordinateur portable, c'est la protection qu'il faut.
- **ZFS**, avec ou sans chiffrement. Un système de fichiers puissant, présenté au chapitre 14.
- **Chiffrement appuyé sur le matériel.** La clé est confiée à la puce TPM de la machine et le disque se déverrouille sans rien taper. Canonical qualifie encore cette fonction d'expérimentale dans la 24.04, et elle ne s'accorde pas avec tous les matériels ni avec les pilotes tiers. Préférez la phrase secrète.

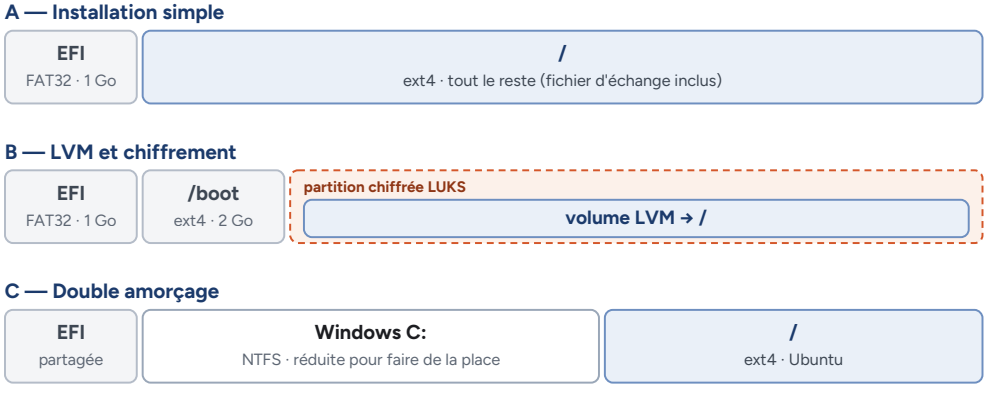


Figure 2.3 — Trois façons de découper le disque. La partition EFI, non chiffrée, est obligatoire sur les machines récentes.

> **À RETENIR · La phrase secrète ne se récupère pas**
>
> Si vous chiffrez le disque et oubliez la phrase secrète, les données sont perdues. Personne ne peut les récupérer, ni Canonical ni un réparateur. Notez-la dans un gestionnaire de mots de passe, sur un autre appareil.

Le compte utilisateur

- **Votre nom** : affiché sur l'écran de connexion. Accents et espaces autorisés.
- **Nom de l'ordinateur** : en minuscules, sans accent ni espace, par exemple `portable-gary`.
- **Nom d'utilisateur** : votre identifiant, court, en minuscules, par exemple `gary`.
- **Mot de passe** : vous le taperez plusieurs fois par jour, car c'est aussi lui que demandera la commande `sudo` pour toute opération d'administration.

Laissez cochée la case *Demander mon mot de passe pour ouvrir une session*.

Le premier démarrage

À la fin, l'installateur demande de retirer la clé puis d'appuyer sur Entrée. Si vous l'oubliez, la machine redémarre sur l'installateur.

Avec un disque chiffré, une invite demande la phrase secrète avant tout le reste. Rien ne s'affiche pendant la saisie, pas même des points. Vient ensuite l'écran de connexion, puis le bureau, que le chapitre 3 vous fera visiter.

2.4 Installer Ubuntu Server

L'installateur du serveur s'appelle Subiquity. Il fonctionne en mode texte et se pilote au clavier : Tab et les flèches pour se déplacer, Espace pour cocher, Entrée pour valider.

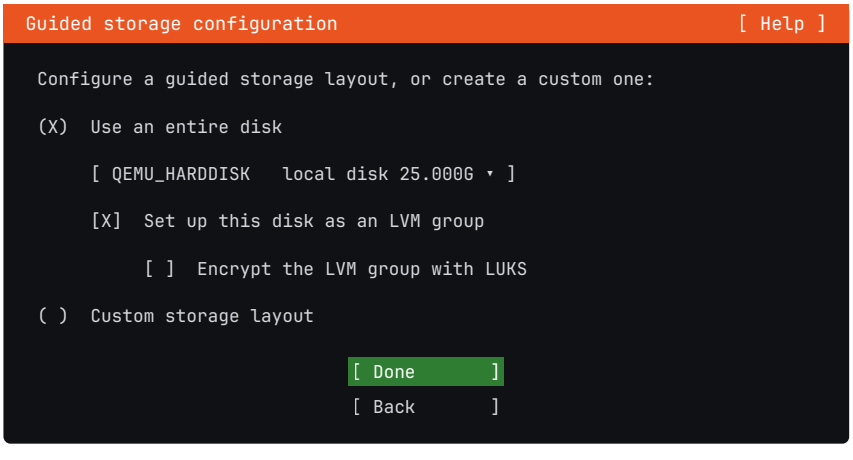


Figure 2.4 — L'écran de configuration du stockage de l'installateur serveur (reproduction).

Les écrans, un par un

1. **Language.** Beaucoup d'administrateurs gardent l'anglais sur un serveur : les messages d'erreur se retrouvent alors mot pour mot dans les moteurs de recherche. Ce livre fait de même pour l'édition Server.
2. **Keyboard configuration.** *French* pour la disposition et pour la variante.
3. **Type of install.** *Ubuntu Server*. La variante *minimized* retire la documentation et plusieurs outils. Elle convient aux machines qu'on ne touche jamais à la main.

4. **Network connections.** Avec DHCP, l'adresse est déjà là. Pour une adresse fixe, sélectionnez l'interface, puis *Edit IPv4* et *Manual*. Le champ *Subnet* attend le réseau en notation CIDR (par exemple `192.168.1.0/24`), *Address* l'adresse du serveur (`192.168.1.100`), *Gateway* celle de la passerelle (`192.168.1.1`) et *Name servers* les serveurs DNS.
5. **Proxy.** Laissez vide, sauf si votre réseau l'impose.
6. **Archive mirror.** Gardez le miroir proposé.
7. **Guided storage configuration.** *Use an entire disk*, avec la case *Set up this disk as an LVM group* cochée par défaut. On peut aussi chiffrer le groupe de volumes, mais un serveur chiffré attend sa phrase secrète à chaque redémarrage. Il faut donc pouvoir accéder à sa console.
8. **Storage configuration.** Le récapitulatif du découpage. Regardez la taille du volume `ubuntu-lv` : par défaut, l'installateur ne lui donne qu'une partie du disque et garde le reste en réserve. Le chapitre 15 montre comment l'agrandir.
9. **Profile configuration.** Votre nom, le nom du serveur, l'identifiant et le mot de passe.
10. **Upgrade to Ubuntu Pro.** *Skip for now*. L'abonnement s'active plus tard, en une commande.
11. **SSH configuration.** Cochez *Install OpenSSH server*. Sans lui, pas d'administration à distance.
12. **Featured server snaps.** Ne cochez rien. Vous installerez ce qu'il faut le moment venu.

L'installation dure de cinq à quinze minutes. Choisissez *Reboot Now* et retirez le support.

> **ASTUCE · Importer ses clés SSH depuis GitHub**
>
> À l'écran SSH, l'option *Import SSH key* récupère les clés publiques associées à un compte GitHub ou Launchpad. Vous pourrez ouvrir une session sans mot de passe dès le premier démarrage.

La première connexion

Sur la console, une invite `login:` attend votre identifiant puis votre mot de passe. L'écran est noir, sans fenêtre ni souris. C'est le cadre de travail normal d'un serveur.

On n'administre pas un serveur depuis sa console, mais depuis son poste, par SSH :

```
$ ssh gary@192.168.1.100
The authenticity of host '192.168.1.100 (192.168.1.100)' can't be established.
ED25519 key fingerprint is SHA256:4ySvJ7dF0kq2uYtGgC1m8sVbP6hXoR3nLwZeKa9TfUo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.100' (ED25519) to the list of known hosts.
gary@192.168.1.100's password:
Welcome to Ubuntu 24.04.5 LTS (GNU/Linux 6.8.0-146-generic x86_64)
```

La question posée à la première connexion est normale : votre poste ne connaît pas encore ce serveur. Répondez `yes`. Le chapitre 19 explique ce que cette empreinte protège.

2.5 Installer dans une machine virtuelle

Une machine virtuelle est un ordinateur simulé par un logiciel, l'*hyperviseur*. Elle a son disque (un gros fichier sur le vôtre), sa mémoire (prélevée sur la vôtre) et sa carte réseau. Ubuntu s'y installe exactement comme sur une machine physique, à partir du fichier image.

Hyperviseur	Système hôte	En deux mots
VirtualBox	Windows, macOS, Linux	Gratuit, le plus répandu pour apprendre.
VMware Workstation Pro, Fusion Pro	Windows, Linux, macOS	Gratuits pour tous les usages depuis novembre 2024.
KVM et virt-manager	Linux	La virtualisation native du noyau Linux.
UTM	macOS	Simple et gratuit, très bien adapté aux Mac à puce Apple.
Multipass	Windows, macOS, Linux	Des machines Ubuntu Server en une commande.

ATTENTION · Sur un Mac à puce Apple (M1 et suivants)

Le processeur est de type ARM. Téléchargez l'image **arm64** d'Ubuntu, pas l'image amd64 : elle tournera à pleine vitesse. UTM, VMware Fusion et VirtualBox à partir de la version 7.1 savent la faire fonctionner.

VirtualBox

1. Cliquez sur *Nouvelle*. Donnez un nom à la machine et indiquez le fichier image.

2. Cochez *Ignorer l'installation sans surveillance* (*Skip Unattended Installation*), pour dérouler vous-même les écrans de l'installateur.
3. Mémoire : 4096 Mo au moins. Processeurs : 2.
4. Disque : 40 Go, alloué dynamiquement. Le fichier ne grossira qu'au fur et à mesure.
5. Démarrez la machine et suivez la section 2.3 ou 2.4.

Une fois Ubuntu Desktop installé, le menu *Périphériques* de la fenêtre propose d'insérer l'image des *Additions invité*. Elles apportent le redimensionnement automatique de l'écran, le presse-papiers partagé et les dossiers partagés avec l'hôte.

KVM et virt-manager

Si votre poste tourne déjà sous Linux, KVM est le meilleur choix : il fait partie du noyau.

```
# Le processeur sait-il virtualiser ? La réponse doit être supérieure à 0
grep -E -c '(vmx|svm)' /proc/cpuinfo

# Installer l'hyperviseur et son interface graphique
sudo apt update
sudo apt install qemu-kvm libvirt-daemon-system virtinst virt-manager

# Autoriser votre compte à gérer les machines, puis rouvrir la session
sudo usermod -aG kvm,libvirt $USER
```

Si la première commande répond 0, activez la virtualisation (Intel VT-x ou AMD-V) dans les réglages du micrologiciel de la machine. Lancez ensuite `virt-manager` : un assistant demande l'image, la mémoire et la taille du disque.

Multipass

Multipass, de Canonical, crée des machines Ubuntu Server sans passer par l'installateur. Il télécharge une image prête à l'emploi et la démarre en moins d'une minute. Sous Ubuntu, il s'installe avec `sudo snap install multipass`.

```
# Après installation de Multipass

# Lancer une VM Ubuntu 24.04 en une commande
multipass launch 24.04 --name lab1 --cpus 2 --memory 4G --disk 20G

# Lister les VM
multipass list

# Se connecter en shell dans la VM
```

```
multipass shell lab1

# Arrêter / supprimer
multipass stop lab1
multipass delete lab1
multipass purge
```

> **ASTUCE · Une machine neuve par atelier**
>
> Pour les ateliers de ce livre qui se font sur un serveur, lancez une machine Multipass, faites l'exercice, puis supprimez-la. Vous repartez d'un système propre à chaque fois.

2.6 Ubuntu sous Windows avec WSL

WSL (*Windows Subsystem for Linux*) fait tourner un vrai noyau Linux dans une machine virtuelle légère que Windows gère pour vous. Vous obtenez un terminal Ubuntu qui s'ouvre aussi vite qu'une fenêtre PowerShell.

Il faut Windows 11, ou Windows 10 à partir de la version 2004. Ouvrez PowerShell en tant qu'administrateur :

```
# Voir les distributions proposées
wsl --list --online

# Installer WSL et Ubuntu 24.04
wsl --install -d Ubuntu-24.04
```

Après un redémarrage, lancez « Ubuntu 24.04 » depuis le menu Démarrer. Le premier lancement demande un identifiant et un mot de passe. Ils sont propres à cet Ubuntu et n'ont aucun lien avec votre compte Windows.

Quelques particularités :

- **Le noyau vient de Microsoft**, pas d'Ubuntu. La commande `uname -r` affiche un nom qui se termine par `microsoft-standard-WSL2`.
- **Les disques de Windows sont accessibles** sous `/mnt/c/`, `/mnt/d/`, etc. Dans l'autre sens, l'Explorateur de fichiers atteint les fichiers d'Ubuntu par le chemin `\\wsl$\Ubuntu-24.04\`.
- **Les applications graphiques Linux** s'ouvrent dans des fenêtres Windows.
- **Le matériel passe par Windows.** Pas d'accès direct aux disques ni aux ports USB.
- **Pas de démarrage :** ni GRUB, ni chargement du noyau à observer.

WSL convient très bien aux chapitres 3 à 10 : ligne de commande, fichiers, permissions, paquets, scripts. À partir du chapitre 11 (démarrage, services, disques, réseau), passez à une machine virtuelle. Ce que vous y verrez ressemblera à un vrai serveur.

2.7 Ubuntu chez un hébergeur

Ici, on n'installe rien. On commande une machine, on choisit « Ubuntu 24.04 » dans une liste, et une minute plus tard elle répond en SSH.

Les familles d'hébergeurs

Les grands fournisseurs de cloud (AWS, Microsoft Azure, Google Cloud) proposent des centaines de services, facturés à l'usage. Leur puissance a un prix : une complexité réelle, et une facture difficile à prévoir quand on débute.

Les hébergeurs européens généralistes (OVHcloud, Scaleway, Infomaniak) offrent un catalogue plus resserré, des prix lisibles et des données hébergées en Europe. OVHcloud et Scaleway assurent leur support en français.

Les spécialistes du serveur privé virtuel (Hetzner, Contabo, Netcup, DigitalOcean, Vultr) louent des VPS au mois, pour quelques euros. C'est le bon endroit pour apprendre et pour héberger ses premiers services.

Pour choisir un centre de données, un critère passe avant les autres : la distance par rapport à vos utilisateurs. Depuis l'Afrique centrale et de l'Ouest, ceux d'Europe de l'Ouest (Paris, Francfort, Londres) donnent en général les meilleurs temps de réponse. Mesurez-le vous-même avec `ping` avant de vous engager.

Créer une clé SSH

Les hébergeurs demandent une clé publique SSH à la commande. Créez-la avant, sur votre poste :

```
$ ssh-keygen -t ed25519 -C "gary@portable"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/gary/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
```

Validez l'emplacement proposé et choisissez une phrase secrète. Deux fichiers sont créés. `~/.ssh/id_ed25519` est la clé privée : elle ne quitte jamais votre poste.

`~/.ssh/id_ed25519.pub` est la clé publique : c'est elle que vous collez dans le formulaire de l'hébergeur.

```
$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIGq8v0sT1k4mJ2QxN7yLr5hB3dWc9eKfUaZoPnV6XtYs
gary@portable
```

Commander et se connecter

La commande se ressemble partout : on choisit un centre de données, l'image Ubuntu 24.04, un modèle de machine (2 vCPU et 4 Go de mémoire suffisent largement pour ce livre), on colle sa clé publique et on donne un nom au serveur. L'hébergeur affiche alors son adresse IP.

Le compte de première connexion dépend de l'hébergeur : `root` chez certains, `ubuntu` chez d'autres. Il est indiqué sur la page de la machine. S'il s'agit de `root`, créez tout de suite un compte personnel et ne travaillez plus qu'avec lui :

```
ssh root@203.0.113.42

# Mettre le système à jour
apt update && apt full-upgrade -y

# Créer un compte d'administration
adduser gary
usermod -aG sudo gary

# Lui donner la même clé SSH que root
rsync --archive --chown=gary:gary ~/.ssh /home/gary
```

Ouvrez un second terminal et vérifiez que `ssh gary@203.0.113.42` fonctionne avant de fermer la session de `root`.

> **ATTENTION · Un serveur sur Internet est attaqué en permanence**
>
> Dans les minutes qui suivent sa mise en ligne, des robots testent son port SSH avec des listes de mots de passe. Ce n'est pas vous qu'ils visent : ils parcourent tout Internet. La connexion par clé vous protège déjà de l'essentiel. La partie VII complète le dispositif avec l'interdiction de la connexion directe de `root`, un pare-feu et Fail2ban.

Ce que fait cloud-init

Comment le serveur connaissait-il son nom et votre clé avant même que vous y entriez ? Grâce à `cloud-init`, un programme présent dans toutes les images cloud d'Ubuntu. Au premier démarrage, il interroge un service de l'hébergeur, récupère le nom de la machine, les clés SSH et les éventuelles commandes à exécuter, puis applique le tout. C'est ce mécanisme qui permet de créer dix serveurs identiques en remplissant un formulaire.

2.8 L'installation automatisée

Installer Ubuntu à la main une fois, c'est instructif. La dixième fois, on cherche à s'en dispenser. Le mécanisme prévu pour cela s'appelle *autoinstall*. On écrit dans un fichier YAML les réponses qu'on aurait données à l'installateur, et celui-ci se déroule sans poser de question.

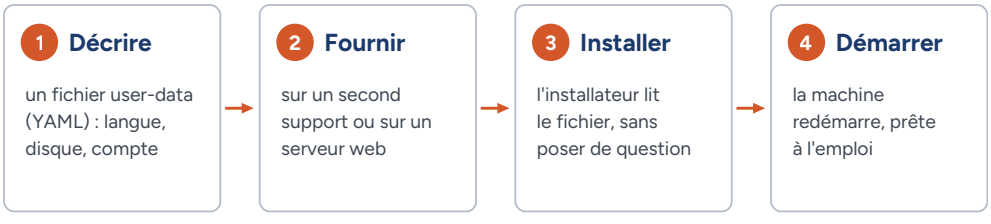


Figure 2.5 — Déroulement d'une installation automatisée.

Voici un fichier minimal :

```
#cloud-config
autoinstall:
  version: 1
  locale: fr_FR.UTF-8
  keyboard:
    layout: fr
  identity:
    hostname: serveur-auto
    username: gary
    # Mot de passe chiffré, produit par : mkpasswd -m sha-512
    password: "$6$exDY1mhS4KUYCE/2$zmn9ToZwTKLhCw.b4/
b.ZRTIZM30JZ4QrOQ2aOXJ8yk96xpcCof0kxKwuX1kqLG/ygbJ1f8wxED22bTL4F46P0"
  ssh:
    install-server: true
    authorized-keys:
      - ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIGq8v0sT1k4mJ2QxN7yLr5hB3dWc9eKfUaZoPnV6XtYs
```

```
gary@portable
  packages:
    - htop
    - tmux
```

Le mot de passe de cet exemple, tiré de la documentation de l'installateur, est `ubuntu`. La commande `mkpasswd`, fournie par le paquet `whois`, produit le vôtre.

L'installateur doit ensuite trouver ce fichier. Il y a deux moyens courants. Le premier consiste à le placer, sous le nom `user-data`, sur un second support dont le volume s'appelle `CIDATA`, accompagné d'un fichier `meta-data` vide. Le second consiste à le publier sur un serveur web du réseau et à ajouter son adresse aux paramètres du noyau, dans le menu de démarrage de l'image :

```
autoinstall ds=nocloud-net\;s=http://192.168.1.10/
```

La barre oblique inversée devant le point-virgule est indispensable. Sans elle, GRUB coupe la ligne à cet endroit.

Deux machines installées à partir du même fichier sont configurées de la même façon, et l'on sait toujours comment en refaire une. Le chapitre 32 s'en sert pour installer un serveur complet. Il montre aussi pourquoi y ajouter une section `network` : sans elle, une carte réseau qui change de nom une fois la clé d'installation retirée laisse la machine injoignable.

2.9 Le double amorçage avec Windows

Faire cohabiter Ubuntu et Windows sur un même disque est possible. C'est aussi la seule opération de ce chapitre qui peut vous coûter vos données. Si vous hésitez, restez sur la machine virtuelle.

Avant de commencer

- **Sauvegardez vos fichiers Windows** sur un disque externe.
- **Windows doit déjà être installé.** L'ordre inverse pose problème : l'installateur de Windows ignore les autres systèmes.
- **Libérez au moins 30 Go**, 50 Go si possible.
- **Suspendez BitLocker.** Sur les PC récents, le disque est souvent chiffré par défaut. Avant toute chose, enregistrez la clé de récupération BitLocker ailleurs que sur ce disque, puis suspendez le chiffrement dans les paramètres de Windows.

- **Désactivez le démarrage rapide** de Windows : *Panneau de configuration*, *Options d'alimentation*, *Choisir l'action des boutons d'alimentation*. Avec cette fonction, Windows ne s'éteint jamais complètement et laisse son disque dans un état qu'Ubuntu refusera de modifier.

La marche à suivre

1. **Faites de la place depuis Windows.** Clic droit sur le menu Démarrer, *Gestion des disques*, clic droit sur la partition `C:`, *Réduire le volume*. Indiquez la taille à libérer en mégaoctets : 50 000 pour 50 Go. Laissez cet espace « non alloué ».
2. **Démarrez sur la clé Ubuntu** et déroulez l'installateur jusqu'à l'écran du disque.
3. **Choisissez *Installer Ubuntu à côté de Windows***. L'installateur utilise l'espace libéré.
4. Si cette option n'apparaît pas, passez par le partitionnement manuel. Créez dans l'espace libre une partition `ext4` montée sur `/`. Ne modifiez ni les partitions NTFS de Windows ni la partition EFI existante : Ubuntu y ajoutera simplement son chargeur.

Au redémarrage, le menu de GRUB propose Ubuntu et *Windows Boot Manager*.

Si quelque chose manque

Windows démarre directement, sans menu. L'ordre de démarrage du micrologiciel a gardé Windows en tête. Entrez dans ses réglages (F2 ou Suppr à l'allumage, selon les machines) et placez l'entrée `ubuntu` en premier.

Le menu s'affiche, mais sans Windows. La détection des autres systèmes n'a pas eu lieu. Ouvrez le fichier `/etc/default/grub`, ajoutez-y la ligne `GRUB_DISABLE_OS_PROBER=false`, puis lancez `sudo update-grub`.

Une mise à jour de Windows a fait disparaître le menu. Cela arrive après certaines grosses mises à jour. Ubuntu est toujours sur le disque, c'est son chargeur qui a été évincé. Le chapitre 31 montre comment le rétablir depuis la clé d'installation.

2.10 Après l'installation

Quelle que soit la voie suivie, les premiers gestes sont les mêmes.

Vérifier où l'on est

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.5 LTS
Release:        24.04
Codename:       noble

$ uname -r
6.8.0-146-generic
```

La première commande donne la version d'Ubuntu, la seconde celle du noyau. Sur une installation Desktop, le noyau peut être plus récent que le 6.8.

Mettre à jour

```
# Rafraîchir la liste des paquets disponibles
sudo apt update

# Installer toutes les mises à jour
sudo apt full-upgrade -y

# Nettoyer les paquets devenus inutiles (anciens noyaux)
sudo apt autoremove --purge -y

# Redémarrer si le noyau a été mis à jour
sudo reboot
```

Faites-le même si l'image était récente. Des correctifs de sécurité paraissent chaque semaine.

Régler le fuseau horaire et la langue

```
# Liste des fuseaux disponibles
timedatectl list-timezones | grep -i paris

# Définir le fuseau
sudo timedatectl set-timezone Africa/Douala
# ou Europe/Paris, America/Montreal, Asia/Tokyo...

# Vérifier
timedatectl

# Installer les langues françaises (si installé en anglais)
sudo apt install -y language-pack-fr language-pack-gnome-fr
```

```
# Régénérer la locale
sudo locale-gen fr_FR.UTF-8
sudo update-locale LANG=fr_FR.UTF-8
```

Installer les pilotes propriétaires

Sur un poste équipé d'une carte graphique Nvidia, le pilote du constructeur donne de bien meilleurs résultats que le pilote libre installé par défaut.

```
# Quel matériel a besoin d'un pilote, et lequel est recommandé ?
ubuntu-drivers devices

# Installer les pilotes recommandés, puis redémarrer
sudo ubuntu-drivers install
sudo reboot
```

Quelques outils dont on se sert tout de suite

```
sudo apt install git curl vim htop tmux tree jq unzip
```

Chacun sera présenté le moment venu : `git` pour suivre les versions de vos fichiers, `curl` pour interroger des serveurs web, `vim` pour éditer du texte, `htop` pour observer les processus, `tmux` pour garder plusieurs terminaux dans une seule fenêtre, `tree` pour afficher une arborescence, `jq` pour lire du JSON.

Prendre un instantané

Dans une machine virtuelle, c'est le moment : le système est propre, à jour, et vous n'avez encore rien modifié. Éteignez la machine et créez un instantané (menu *Machine*, *Outils*, *Instantanés* dans VirtualBox ; bouton *Instantanés* dans virt-manager). Nommez-le « installation propre ». Vous y reviendrez chaque fois qu'un exercice aura mal tourné.

La liste de contrôle

- `lsb_release -a` annonce Ubuntu 24.04.
- `sudo apt update` se termine sans erreur.
- `ping -c 3 ubuntu.com` reçoit des réponses : le réseau et le DNS fonctionnent.
- `whoami` affiche votre identifiant, pas `root`.
- `sudo whoami` demande votre mot de passe et affiche `root`.
- Sur une machine virtuelle : l'instantané est pris.

- Sur un serveur distant : la connexion par clé fonctionne avec votre compte personnel.

En résumé

- Pour apprendre, la machine virtuelle est le terrain le plus sûr. Pour héberger, on loue un serveur.
- On vérifie toujours l'empreinte SHA-256 d'une image téléchargée.
- L'installateur Desktop propose LVM, le chiffrement LUKS et ZFS dans ses fonctions avancées. Un portable devrait toujours être chiffré.
- L'installateur Server, Subiquity, est en mode texte. La case à ne pas oublier est celle du serveur OpenSSH.
- WSL suffit pour la ligne de commande, pas pour l'administration d'un système complet.
- Chez un hébergeur, on se connecte par clé SSH et l'on crée son propre compte dès la première session.
- Le double amorçage exige une sauvegarde, la suspension de BitLocker et la désactivation du démarrage rapide.
- Après l'installation : mise à jour, vérifications, instantané.

Atelier — Une machine virtuelle chiffrée

Objectif : installer Ubuntu Desktop 24.04 dans VirtualBox avec un disque chiffré, puis prendre un instantané. Comptez une heure.

1. Téléchargez l'image Desktop et le fichier `SHA256SUMS`. Vérifiez l'empreinte.
2. Créez une machine virtuelle : 4 Go de mémoire, 2 processeurs, 40 Go de disque dynamique, installation sans surveillance ignorée.
3. Démarrez et déroulez l'installateur en français. À l'écran du disque, choisissez *Effacer le disque*, puis dans les fonctions avancées *LVM et chiffrement*. Notez la phrase secrète.
4. Au redémarrage, retirez l'image du lecteur virtuel (*Périphériques*, *Lecteurs optiques*), saisissez la phrase secrète et ouvrez votre session.
5. Ouvrez un terminal avec Ctrl+Alt+T. Mettez le système à jour et redémarrez.

6. Éteignez la machine. Prenez un instantané nommé « installation propre ».

Vous avez réussi si la machine demande la phrase secrète au démarrage, si `lsblk -f` montre une partition de type `crypto_LUKS` contenant un volume LVM, et si l'instantané apparaît dans VirtualBox.

Exercices

Exercice 1 — Choisir sa voie

Quelle façon d'installer Ubuntu conseilleriez-vous, et pourquoi ?

1. Une étudiante en informatique dont le portable Windows 11 doit rester utilisable pour ses cours.
2. Un développeur web équipé d'un MacBook à puce M2.
3. Un administrateur qui doit mettre en ligne le site de son entreprise, installée à Douala.
4. Un enseignant tenté d'abandonner Windows, mais qui n'est pas encore décidé.
5. Une personne qui veut transformer un vieux portable en serveur de sauvegarde pour la maison.

Exercice 2 — L'empreinte ne correspond pas

L'empreinte publiée pour votre image se termine par `…e8f9a0`. Celle que vous calculez se termine par `…e8f9a1`. Que faites-vous ? Quelles sont les deux explications possibles ?

Exercice 3 — Où héberger ?

Une PME de Yaoundé veut un site WordPress pour une clientèle locale. Comparez ces options selon le temps de réponse, le coût, la simplicité et la localisation des données : un grand fournisseur de cloud en Virginie, le même à Paris, un VPS en Allemagne, un VPS en France.

Exercice 4 — Trois niveaux de protection

Comparez un portable sans chiffrement, un portable chiffré avec phrase secrète et un portable chiffré avec déverrouillage par la puce TPM. Dans chaque cas :

1. Le voleur d'un portable éteint peut-il lire les données ?

2. Peut-il les lire en démontant le disque pour le brancher ailleurs ?
3. Que se passe-t-il si la carte mère tombe en panne ?
4. Quelle contrainte cela représente-t-il au quotidien ?

Exercice 5 — Lire un fichier autoinstall

Dans le fichier de la section 2.8, que faudrait-il modifier pour que le serveur s'appelle `web-01`, que le clavier soit en disposition américaine et que le paquet `git` soit installé ?

Pour aller plus loin

Documentation. `ubuntu.com/tutorials/install-ubuntu-desktop` et `ubuntu.com/tutorials/install-ubuntu-server` pour les guides illustrés de Canonical. `canonical-subiquity.readthedocs-hosted.com` pour la référence complète d'autoinstall. `ubuntu.com/certified` pour la liste du matériel certifié.

Outils. `etcher.balena.io`, `rufus.ie`, `ventoy.net` pour préparer une clé. `virtualbox.org`, `mac.getutm.app` et `canonical.com/multipass` pour la virtualisation. `learn.microsoft.com/windows/wsl` pour WSL.

Lectures. *Mastering Ubuntu Server*, de Jay LaCroix (Packt) : un bon compagnon pour la partie serveur, en anglais. *The Linux Command Line*, de William Shotts, disponible gratuitement sur `linuxcommand.org`.



CHAPITRE 3

Premiers pas dans l'environnement

Se repérer sur le bureau, ouvrir un terminal sans appréhension et taper ses premières commandes.

DANS CE CHAPITRE

- Se repérer dans le bureau GNOME : barre supérieure, dock, vue d'ensemble, paramètres.
- Ouvrir un terminal et lire ce que dit l'invite de commande.
- Utiliser les commandes de tous les jours : `pwd`, `ls`, `cd`, `cat`, `less`, `echo`, `date`, `history`.
- Trouver de l'aide seul avec `man`, `--help`, `apropos` et `tldr`.
- Modifier un fichier texte avec `nano`.
- Installer un logiciel avec APT, Snap ou Flatpak, en sachant ce qui les distingue.

> **AVANT DE COMMENCER**
>
> Un Ubuntu 24.04 installé (chapitre 2). La section 3.1 décrit le bureau de l'édition Desktop. Si vous travaillez sur Ubuntu Server ou sous WSL, commencez à la section 3.3.
>
> Ce chapitre présente plus de commandes que vous n'en retiendrez à la première lecture. Vous y reviendrez. Les commandes s'apprennent en les tapant, et vingt commandes tapées dix fois valent mieux que cent commandes lues une fois.

3.1 Le bureau GNOME

L'environnement de bureau d'Ubuntu s'appelle GNOME. La version 46 équipe Ubuntu 24.04. Deux choses déroutent en arrivant de Windows : il n'y a pas de menu Démarrer, et presque tout passe par une vue d'ensemble que l'on appelle d'une seule touche.

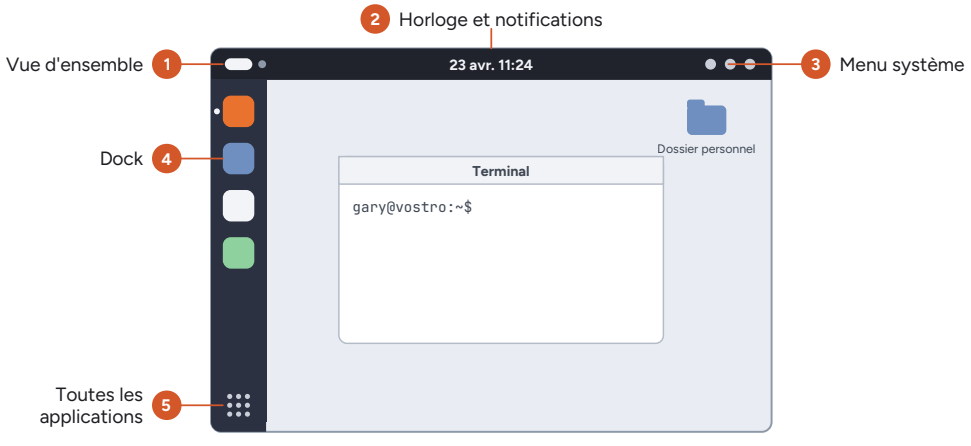


Figure 3.1 — Le bureau d'Ubuntu 24.04 à l'ouverture de session.

Ce que l'on voit à l'écran

La barre supérieure porte trois zones. À gauche, un indicateur des espaces de travail : un clic dessus ouvre la vue d'ensemble. Au centre, la date et l'heure : un clic affiche le calendrier et les notifications. À droite, le menu système, avec le réseau, le son, la batterie, le thème sombre, le verrouillage et l'extinction.

Le dock, à gauche, contient les applications favorites et celles qui sont ouvertes, signalées par un point. Tout en bas, un bouton à neuf points ouvre la grille de toutes

les applications. Pour garder une application dans le dock, lancez-la, faites un clic droit sur son icône et choisissez *Épingler au Dash*.

Le bureau affiche l'icône de votre dossier personnel et le contenu du dossier `Bureau`.

La vue d'ensemble

Appuyez sur la touche Super, celle qui porte le logo Windows sur la plupart des claviers. L'écran se réorganise : les fenêtres ouvertes se rangent côte à côte, les espaces de travail apparaissent en haut, et un champ de recherche attend votre saisie.

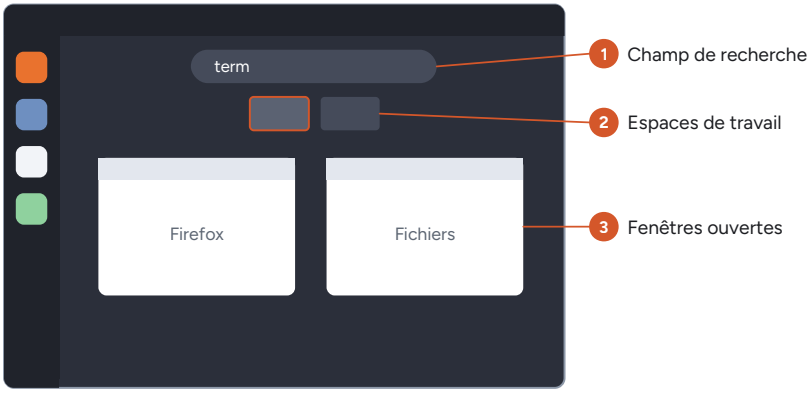


Figure 3.2 — La vue d'ensemble, que l'on ouvre avec la touche Super.

Tapez directement quelques lettres. La recherche porte sur les applications, les fichiers récents, les réglages et même les calculs : `term` propose le Terminal, `blue` le réglage du Bluetooth, `42*7` affiche 294. Entrée ouvre le premier résultat.

> **À RETENIR · Super, quelques lettres, Entrée**
> Ces trois gestes remplacent le menu Démarrer. Prenez l'habitude tout de suite : c'est la façon la plus rapide de lancer quoi que ce soit.

Les raccourcis qui servent tous les jours

Raccourci	Effet
Super	Ouvrir ou fermer la vue d'ensemble
Super + A	Afficher la grille des applications

Raccourci	Effet
Alt + Tab	Passer d'une fenêtre à l'autre
Super + Tab	Passer d'une application à l'autre
Super + ← ou →	Caler la fenêtre sur une moitié de l'écran
Super + ↑ ou ↓	Agrandir la fenêtre, puis la rétablir
Super + H	Réduire la fenêtre
Alt + F4	Fermer la fenêtre
Super + Page ↑ ou Page ↓	Changer d'espace de travail
Super + Maj + Page ↑ ou Page ↓	Emmener la fenêtre dans un autre espace de travail
Super + L	Verrouiller la session
Impr. écran	Capturer l'écran, une zone ou une fenêtre, ou filmer l'écran
Ctrl + Alt + T	Ouvrir un terminal

La liste complète, modifiable, se trouve dans *Paramètres*, *Clavier*, *Raccourcis clavier*.

Les Paramètres

L'application Paramètres regroupe tous les réglages. Les panneaux utiles au début :

- **Wi-Fi** et **Réseau** : connexions, VPN, proxy.
- **Écrans** : résolution, mise à l'échelle, écrans multiples.
- **Apparence** : style clair ou sombre, couleur d'accentuation, fond d'écran.
- **Bureau Ubuntu** : position et taille du dock, icônes du bureau.
- **Multi-tâches** : coin actif, espaces de travail.
- **Clavier** : dispositions et raccourcis.
- **Système** : pays et langue, date et heure, utilisateurs, et la page *À propos*, qui donne la version d'Ubuntu et le matériel détecté.

Les extensions

GNOME se complète par des extensions, de petits modules qui modifient l'interface. Ubuntu en livre quatre, actives d'office : le dock, les icônes du bureau, les icônes d'état de certaines applications et l'assistant de placement des fenêtres.

Pour en ajouter, installez le Gestionnaire d'extensions (`sudo apt install gnome-shell-extension-manager`). Son onglet *Parcourir* donne accès au catalogue de `extensions.gnome.org`. Deux exemples populaires : *Clipboard Indicator*, qui garde un historique du presse-papiers, et *Vitals*, qui affiche températures et charge dans la barre supérieure.

> **ATTENTION · Les extensions suivent mal les changements de version**
> Une extension écrite pour une version de GNOME peut cesser de fonctionner avec la suivante, le temps que son auteur l'adapte. N'en faites pas dépendre votre façon de travailler.

3.2 Ouvrir un terminal

Le terminal est l'outil principal de ce livre. Il impressionne au début, à tort. C'est une application dans laquelle on écrit une ligne de texte et qui répond par du texte.

Trois façons de l'ouvrir sur Ubuntu Desktop :

- le raccourci Ctrl+Alt+T, la plus rapide ;
- la touche Super, puis `term` et Entrée ;
- dans le gestionnaire de fichiers, un clic droit dans un dossier, puis *Ouvrir dans un terminal*. Le terminal s'ouvre alors directement dans ce dossier.

Les raccourcis du terminal

Raccourci	Effet
Tab	Compléter le mot en cours ; deux fois : afficher les possibilités
↑ et ↓	Parcourir les commandes précédentes
Ctrl+R	Chercher dans l'historique des commandes
Ctrl+C	Interrompre la commande en cours
Ctrl+D	Fermer la session (sur une ligne vide)
Ctrl+L	Effacer l'écran
Ctrl+A, Ctrl+E	Aller au début, à la fin de la ligne
Ctrl+U, Ctrl+K	Effacer avant, après le curseur
Ctrl+W	Effacer le mot qui précède

Raccourci	Effet
Ctrl+Maj+C, Ctrl+Maj+V	Copier, coller
Ctrl+Maj+T	Ouvrir un nouvel onglet

DANGER · Ctrl+C ne copie pas

Dans un terminal, Ctrl+C interrompt le programme en cours. Pour copier, c'est Ctrl+Maj+C. Tout le monde se trompe pendant la première semaine.

À RETENIR · La touche Tab

Tapez `cd Doc`, appuyez sur Tab : le terminal écrit `cd Documents/`. La complétion fonctionne pour les commandes, les fichiers, les dossiers et souvent les options. Elle évite les fautes de frappe et montre ce qui existe. Appuyez sur Tab après chaque début de mot, jusqu'à ce que cela devienne un réflexe.

3.3 Lire l'invite de commande

À l'ouverture, le terminal affiche une ligne qui se termine par un curseur. C'est l'*invite de commande* (en anglais *prompt*).



Figure 3.3 — Ce que dit l'invite de commande.

Elle suit toujours le même modèle : `utilisateur@machine:dossier$`.

- **L'utilisateur** est le compte avec lequel vous travaillez.
- **La machine** est le nom de l'ordinateur. Quand on administre plusieurs serveurs à distance, ce nom évite de taper une commande sur la mauvaise machine.
- **Le dossier** est celui où vous vous trouvez. Le tilde `~` désigne votre dossier personnel, ici `/home/gary`.
- **Le dernier caractère** indique vos droits : `$` pour un utilisateur ordinaire, `#` pour l'administrateur `root`.

L'invite change au fil de vos déplacements :

```
gary@vostro:~$ cd Documents
gary@vostro:~/Documents$ cd /etc
gary@vostro:/etc$ sudo -i
root@vostro:~# exit
gary@vostro:/etc$
```

> **ATTENTION · Le mot de passe ne s'affiche pas**
>
> Quand `sudo` demande votre mot de passe, rien n'apparaît à l'écran pendant la frappe, ni étoiles ni points. C'est voulu : quelqu'un qui regarde par-dessus votre épaule n'apprend même pas sa longueur. Tapez-le et validez.

L'apparence de l'invite se règle par une variable nommée `PS1`. Le chapitre 8 montre comment la personnaliser.

3.4 Les commandes de tous les jours

Dans ce livre, les lignes qui commencent par une invite sont celles que vous tapez. Celles qui suivent sont la réponse du système. Vous ne recopiez que ce qui se trouve après le `$`.

Où suis-je ? `pwd`

```
gary@vostro:~$ pwd
/home/gary
```

`pwd` (*print working directory*) affiche le chemin complet du dossier courant.

Qu'y a-t-il ici ? `ls`

```
gary@vostro:~$ ls
Bureau  Documents  Images  Modèles  Musique  Public  Téléchargements  Vidéos
```

`ls` liste le contenu d'un dossier. Trois options suffisent pour commencer : `-l` affiche une ligne détaillée par fichier, `-a` montre aussi les fichiers cachés (ceux dont le nom commence par un point), et `-h` exprime les tailles en kilo, méga et gigaoctets. On les combine librement.

```
gary@vostro:~$ ls -lah
total 76K
drwxr-x--- 15 gary gary 4,0K avril 23 11:02 .
drwxr-xr-x  3 root root 4,0K avril 20 09:14 ..
-rw-------  1 gary gary 1,2K avril 23 10:58 .bash_history
-rw-r--r--  1 gary gary  220 avril 20 09:14 .bash_logout
-rw-r--r--  1 gary gary 3,7K avril 20 09:14 .bashrc
drwxr-xr-x  2 gary gary 4,0K avril 20 09:20 Bureau
drwxr-xr-x  2 gary gary 4,0K avril 20 09:20 Documents
```

Chaque colonne a un sens : droits d'accès, nombre de liens, propriétaire, groupe, taille, date de modification, nom. Le chapitre 6 les explique en détail. D'autres variantes utiles : `ls -lt` trie par date, `ls -lS` par taille, et `ls /etc` liste un autre dossier que le dossier courant.

Ubuntu définit un raccourci, `ll`, qui équivaut à `ls -alF`.

Se déplacer : `cd`

```
# Aller dans un sous-répertoire
gary@vostro:~$ cd Documents
gary@vostro:~/Documents$

# Remonter d'un niveau (deux points = dossier parent)
gary@vostro:~/Documents$ cd ..
gary@vostro:~$

# Aller dans un chemin absolu
gary@vostro:~$ cd /etc
gary@vostro:/etc$

# Revenir à son home, deux équivalents
gary@vostro:/etc$ cd
gary@vostro:~$ cd ~

# Revenir au dossier où on était juste avant
gary@vostro:~$ cd -
/etc
gary@vostro:/etc$

# Aller deux niveaux au-dessus
gary@vostro:/etc/apt/sources.list.d$ cd ../..
gary@vostro:/etc$
```

Trois écritures à retenir : `cd` tout seul ramène au dossier personnel, `cd ..` remonte d'un niveau, `cd -` retourne au dossier précédent.

Lire un fichier : `cat`, `less`, `head`, `tail`

```
gary@vostro:~$ cat /etc/os-release
PRETTY_NAME="Ubuntu 24.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.5 LTS (Noble Numbat)"
VERSION_CODENAME=noble
```

`cat` envoie tout le fichier à l'écran. C'est parfait pour un fichier court. Pour un fichier long, utilisez `less`, qui l'affiche page par page : Espace avance, `b` recule, `/mot` cherche un mot, `q` quitte.

```
less /var/log/syslog          # parcourir le journal du système
head -n 5 /etc/passwd         # les cinq premières lignes
tail -n 20 /var/log/syslog    # les vingt dernières
tail -f /var/log/syslog       # suivre le fichier au fil de l'eau (Ctrl+C pour arrêter)
```

Afficher du texte : `echo`

```
gary@vostro:~$ echo "Bonjour"
Bonjour
gary@vostro:~$ echo $HOME
/home/gary
gary@vostro:~$ echo "Je suis $(whoami) sur $(hostname)"
Je suis gary sur vostro
```

`echo` répète ce qu'on lui donne. Cela semble inutile, jusqu'au moment où l'on s'en sert pour afficher la valeur d'une variable (`$HOME`) ou le résultat d'une autre commande (`$(whoami)`). Les chapitres 5 et 8 y reviennent.

Qui suis-je, et sur quelle machine ?

```
gary@vostro:~$ whoami
gary
gary@vostro:~$ hostname
vostro
gary@vostro:~$ uname -r
6.8.0-146-generic
gary@vostro:~$ uname -m
x86_64
```

`uname -r` donne la version du noyau, `uname -m` l'architecture, `uname -a` tout à la fois.

La date

```
gary@vostro:~$ date
jeu. 23 avril 2026 11:24:08 WAT
gary@vostro:~$ date "+%Y-%m-%d %H:%M"
2026-04-23 11:24
gary@vostro:~$ date -d "+7 days" +%F
2026-04-30
```

Le signe `+` introduit un format : `%Y` pour l'année, `%m` pour le mois, `%d` pour le jour, `%F` pour la date complète au format AAAA-MM-JJ. L'option `-d` calcule une date à partir d'une expression. Vous vous en servirez pour nommer des sauvegardes.

L'historique

Le shell garde la trace de vos commandes. La commande `history` les liste avec un numéro.

```
gary@vostro:~$ history | tail -n 4
  101  cd /etc
  102  cat os-release
  103  cd
  104  history | tail -n 4
gary@vostro:~$ !102
cat os-release
cat: os-release: Aucun fichier ou dossier de ce nom
```

`!102` relance la commande numéro 102. Ici elle échoue, et c'est instructif : la commande est rejouée telle quelle, mais depuis le dossier où l'on se trouve maintenant. `!!` relance la dernière commande, ce qui donne l'idiome `sudo !!` quand on a oublié `sudo`.

Le plus efficace reste Ctrl+R. Tapez quelques lettres, et le shell propose la dernière commande qui les contient. Un nouvel appui sur Ctrl+R remonte à la précédente.

Nettoyer et quitter

`clear`, ou Ctrl+L, efface l'écran. `exit`, ou Ctrl+D, ferme la session. Si l'affichage est brouillé, par exemple après avoir envoyé un fichier binaire à l'écran par erreur, tapez `reset`, même à l'aveugle.

Commande	Ce qu'elle fait	Exemple
`pwd`	Affiche le dossier courant	`pwd`
`ls`	Liste le contenu d'un dossier	`ls -lah`
`cd`	Change de dossier	`cd /etc`, `cd ..`, `cd -`
`cat`	Affiche un fichier en entier	`cat /etc/os-release`
`less`	Affiche un fichier page par page	`less /var/log/syslog`
`head`, `tail`	Affichent le début, la fin d'un fichier	`tail -f /var/log/syslog`
`echo`	Affiche un texte ou une variable	`echo $HOME`
`whoami`, `hostname`	Affichent l'utilisateur, la machine	`whoami`
`uname`	Renseigne sur le noyau et l'architecture	`uname -r`
`date`	Affiche ou calcule une date	`date +%F`
`history`	Liste les commandes déjà tapées	`history | tail`
`clear`, `reset`	Effacent l'écran, réinitialisent le terminal	`clear`
`exit`	Ferme la session	`exit`

3.5 Trouver de l'aide

Personne ne connaît toutes les options de toutes les commandes. Les administrateurs expérimentés ne les retiennent pas, ils savent où les trouver. Et l'aide est sur la machine, disponible même sans réseau.

Les pages de manuel : `man`



Figure 3.4 — Le début de la page de manuel de ls.

`man ls` affiche le manuel de `ls`. Toutes les pages ont la même structure : NAME (le nom et une ligne de description), SYNOPSIS (la syntaxe), DESCRIPTION (les options, une par une), puis souvent EXAMPLES et SEE ALSO. On s'y déplace comme dans `less` : Espace pour avancer, `/mot` pour chercher, `n` pour l'occurrence suivante, `q` pour quitter.

Dans le SYNOPSIS, les crochets signalent ce qui est facultatif et les points de suspension ce qui peut se répéter. `ls [OPTION]... [FILE]...` se lit donc : `ls`, suivi éventuellement d'options, suivi éventuellement de fichiers.

Le manuel est découpé en sections numérotées. Les trois qui vous serviront :

Section	Contenu	Exemple
1	Commandes pour tous les utilisateurs	`man 1 crontab`
5	Format des fichiers de configuration	`man 5 crontab`
8	Commandes d'administration	`man 8 useradd`

Quand un même nom existe dans deux sections, on précise le numéro. `man crontab` décrit la commande, `man 5 crontab` décrit le format du fichier qu'elle

édite. La notation `crontab(5)`, que vous croiserez dans les documentations, désigne la page de la section 5.

L'aide intégrée : `--help`

Presque toutes les commandes acceptent l'option `--help`, qui affiche un résumé de leurs options :

```
gary@vostro:~$ ls --help | head -n 6
Utilisation : ls [OPTION]... [FICHIER]...
Afficher des renseignements sur les FICHIERs (du répertoire actuel par défaut).
Trier les entrées alphabétiquement si aucune des options -cftuvSUX ou --sort
ne sont utilisées.

Les arguments obligatoires pour les options longues le sont aussi pour les
```

C'est plus court que le manuel, et souvent suffisant.

Trouver le nom d'une commande : `apropos`

Vous savez ce que vous voulez faire, mais pas avec quelle commande. `apropos` cherche un mot dans les descriptions de toutes les pages de manuel :

```
gary@vostro:~$ apropos compress
bzip2 (1)            - a block-sorting file compressor, v1.0.8
gzip (1)             - compress or expand files
xz (1)               - Compresser ou décompresser des fichiers .xz et .lzma
zstd (1)             - zstd, zstdmt, unzstd, zstdcat - Compress or decompress...
```

Les descriptions sont en anglais pour la plupart : cherchez avec des mots anglais. Dans l'autre sens, `whatis ls` donne la description d'une commande en une ligne.

Des exemples plutôt qu'une référence : `tldr`

Les pages de manuel sont complètes, parfois trop quand on cherche seulement la forme habituelle d'une commande. Le projet communautaire *tldr* (pour *too long; didn't read*) publie, pour chaque commande, une demi-douzaine d'exemples commentés, dont beaucoup sont traduits en français.

```
sudo apt install pipx
pipx install tldr
pipx ensurepath        # puis fermez et rouvrez le terminal
```

```
gary@vostro:~$ tldr -L fr tar
  tar
  Utilitaire d'archivage.  Souvent combiné avec une méthode de compression, telle
que gzip ou bzip2.

  - Crée une archive à partir de fichiers :
    tar cf chemin/vers/cible.tar chemin/vers/fichier1 chemin/vers/fichier2 ...

  - Crée une archive gzip à partir de fichiers :
    tar czf chemin/vers/cible.tar.gz chemin/vers/fichier1 chemin/vers/fichier2 ...

  - Extrait une archive (compressée) dans un répertoire cible :
    tar xf chemin/vers/source.tar.ext --directory répertoire
```

NOTE · Pourquoi pas `apt install tldr` ?

Le paquet `tldr` des dépôts d'Ubuntu 24.04 n'arrive plus à télécharger les pages du projet. La version installée par `pipx` fonctionne, et c'est pour cela qu'elle est recommandée ici. `pipx` installe les outils écrits en Python dans votre dossier personnel, sans toucher au système.

Quelle aide pour quelle question ?

Vous vous demandez…	Tapez
Comment s'utilise cette commande, en pratique ?	`tldr commande`
Quelle est cette option, déjà ?	`commande --help`
Que fait exactement cette option ?	`man commande`
Quelle commande fait cela ?	`apropos mot`
Comment s'écrit ce fichier de configuration ?	`man 5 fichier`

3.6 Les éditeurs de texte

Sous Linux, la configuration du système tient dans des fichiers texte. Administrer une machine, c'est en grande partie les modifier. Il faut donc un éditeur qui fonctionne dans un terminal.

nano, pour commencer

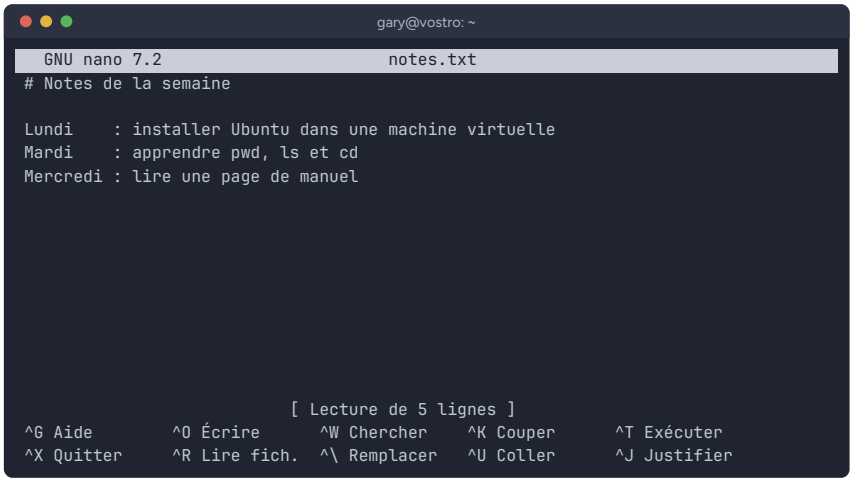


Figure 3.5 — L'éditeur nano. Les deux lignes du bas rappellent les raccourcis ; l'accent circonflexe représente la touche Ctrl.

`nano notes.txt` ouvre le fichier, ou le crée s'il n'existe pas. On tape son texte directement, sans mode à activer. Les raccourcis sont affichés en bas de l'écran.

Raccourci	Effet
Ctrl+O, puis Entrée	Enregistrer
Ctrl+X	Quitter (nano propose d'enregistrer si besoin)
Ctrl+W	Chercher
Ctrl+\	Chercher et remplacer
Ctrl+K, Ctrl+U	Couper la ligne, la coller
Ctrl+_	Aller à une ligne donnée

Pour modifier un fichier du système, on ajoute `sudo` : `sudo nano /etc/hosts`.

vim, pour plus tard

`vim` est présent sur à peu près tous les systèmes Unix, et c'est sa grande force. Sur un serveur inconnu, on est sûr de le trouver. Il est aussi d'une efficacité redoutable entre des mains exercées. Mais il déroute, car il fonctionne par *modes* : à l'ouverture, les touches ne produisent pas de texte, elles exécutent des commandes.

Retenez au moins de quoi en sortir, car vous y entrerez un jour sans l'avoir voulu :

Touche	Effet
i	Passer en mode insertion : on peut écrire
Échap	Revenir au mode commande
`:w` puis Entrée	Enregistrer
`:q` puis Entrée	Quitter
`:wq`	Enregistrer et quitter
`:q!`	Quitter sans enregistrer

Ubuntu installe une version réduite de vim, accessible par la commande `vi`. La version complète s'obtient avec `sudo apt install vim`. Elle fournit `vimtutor`, un tutoriel interactif d'une demi-heure.

> **REPÈRE HISTORIQUE · vi contre Emacs**
>
> `vi`, l'ancêtre de vim, a été écrit par Bill Joy en 1976. La même année, Richard Stallman et Guy Steele mettaient au point Emacs au MIT. Les deux éditeurs incarnent deux philosophies. `vi` mise sur des commandes brèves, tapées sans quitter le clavier. Emacs est un environnement entier, extensible à l'infini, dans lequel certains lisent aussi leur courrier. La rivalité entre leurs partisans dure depuis un demi-siècle et fait partie du folklore.

Et en mode graphique

Sur Ubuntu Desktop, l'**Éditeur de texte** de GNOME (commande `gnome-text-editor`) suffit pour les modifications simples. Pour écrire du code, **Visual Studio Code** est aujourd'hui l'éditeur le plus répandu : `sudo snap install code --classic`.

Mon conseil : `nano` dès maintenant, quelques séances de `vimtutor` quand vous administrerez des serveurs, et un éditeur graphique pour le code.

3.7 Les logiciels fournis

L'installation par défaut d'Ubuntu Desktop 24.04 est volontairement légère. Vous y trouvez :

- **Firefox**, le navigateur ;

- **Fichiers**, le gestionnaire de fichiers. Son entrée *Autres emplacements* permet de se connecter à un serveur distant (adresses en `sftp://`, `smb://`…) ;
- le **Terminal** et l'**Éditeur de texte** ;
- le **Moniteur système**, qui montre les processus et l'occupation du processeur et de la mémoire ;
- **Disques**, pour examiner, formater et tester disques et clés USB ;
- le **Visionneur d'images** et le **Visionneur de documents**, pour les PDF ;
- le **Centre d'applications**, la logithèque d'Ubuntu.

La sélection étendue de l'installateur ajoute la suite bureautique LibreOffice, la messagerie Thunderbird, le lecteur de musique Rhythmbox et quelques autres applications. Si vous ne l'avez pas choisie, chacune s'installe en une commande, comme on va le voir.

LibreOffice ouvre et enregistre les formats de Microsoft Office. Pour les documents courants, la compatibilité est bonne. Elle l'est moins pour les mises en page très élaborées et pour les macros.

3.8 Installer un premier logiciel

Sous Windows, on télécharge un installateur sur le site de l'éditeur. Sous Linux, on procède autrement : on demande au système d'aller chercher le logiciel dans un *dépôt*, un catalogue vérifié et signé. Il existe trois catalogues, donc trois outils.

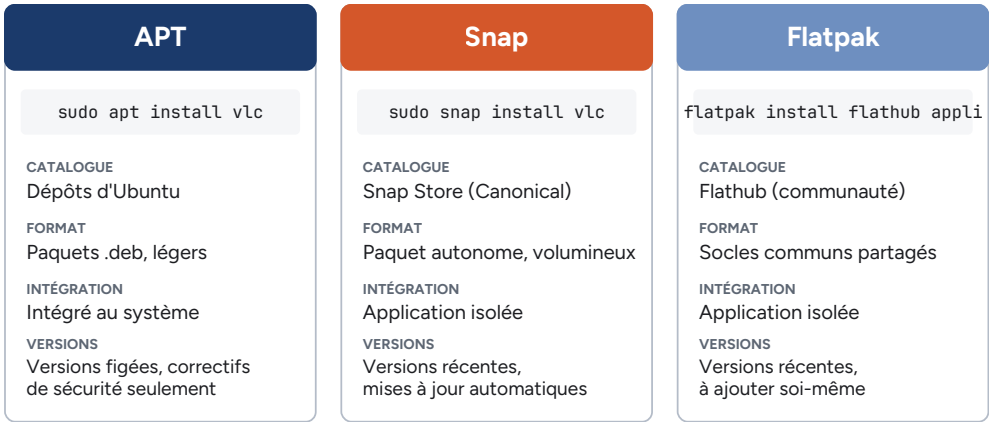


Figure 3.6 — Trois façons d'installer un logiciel sur Ubuntu.

APT

APT est le gestionnaire de paquets historique de Debian et d'Ubuntu. Il installe des fichiers `.deb` pris dans les dépôts d'Ubuntu, qui contiennent des dizaines de milliers de logiciels.

```
sudo apt update          # rafraîchir le catalogue
apt search htop          # chercher un paquet
apt show htop            # lire sa description
sudo apt install htop    # l'installer
sudo apt remove htop     # le désinstaller
sudo apt upgrade         # mettre à jour tout ce qui est installé
```

Les paquets sont petits, car ils partagent leurs bibliothèques, et ils s'intègrent parfaitement au système. En contrepartie, dans une version LTS, les logiciels restent à la version qu'ils avaient à la sortie d'Ubuntu. Ils reçoivent des correctifs de sécurité, pas de nouvelles fonctions.

Snap

Un *snap* est un paquet autonome. Il embarque tout ce dont l'application a besoin, s'exécute dans un espace isolé du reste du système et se met à jour tout seul. Le catalogue, le Snap Store, est géré par Canonical, et ce sont souvent les éditeurs eux-mêmes qui y publient leurs logiciels.

```
snap find vlc             # chercher
sudo snap install vlc     # installer
snap list                 # voir ce qui est installé
sudo snap remove vlc      # désinstaller
```

Les versions sont récentes et l'isolation est un vrai plus pour la sécurité. Le revers : des paquets volumineux, un premier lancement parfois lent, et un catalogue qui dépend d'une seule entreprise.

Flatpak

Flatpak repose sur le même principe que Snap, avec une gouvernance communautaire et un catalogue principal, Flathub. Ubuntu ne l'installe pas, mais rien n'empêche de l'ajouter :

```
sudo apt install flatpak
flatpak remote-add --if-not-exists flathub https://dl.flathub.org/repo/flathub.flatpakrepo
flatpak install flathub org.gimp.GIMP
flatpak run org.gimp.GIMP
```

	APT	Snap	Flatpak
Catalogue	Dépôts d'Ubuntu	Snap Store (Canonical)	Flathub (communauté)
Format	`.deb`	`.snap`	Flatpak
Taille	Petite : bibliothèques partagées	Grande : tout est embarqué	Moyenne : socles communs partagés
Isolation	Aucune	Oui (AppArmor)	Oui (bubblewrap)
Versions	Figées à la sortie d'Ubuntu	Récentes	Récentes
Mises à jour	Avec celles du système	Automatiques	À la demande
Idéal pour	Outils système, serveurs, ligne de commande	Applications de bureau, outils publiés par leur éditeur	Applications de bureau

Une règle simple pour ne pas se perdre : APT pour les outils du système et tout ce qui sert en ligne de commande, Snap ou Flatpak pour une application de bureau dont vous voulez la dernière version. Le chapitre 7 reprend tout cela en détail.

Votre première installation

`htop` est un moniteur de processus en mode texte. Installons-le.

```
gary@vostro:~$ sudo apt update
[sudo] password for gary:
gary@vostro:~$ sudo apt install htop
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  htop
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 171 kB of archives.
After this operation, 434 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble/main amd64 htop amd64 3.3.0-4build1 [171 kB]
Setting up htop (3.3.0-4build1) ...
gary@vostro:~$ htop
```

Sur un système installé en français, ces messages sont traduits. Dans `htop`, F3 cherche un processus, F6 change le tri et F10 (ou q) quitte.

> **À RETENIR · Ce que fait sudo**
>
> `sudo` exécute une commande avec les droits de l'administrateur. Il demande *votre* mot de passe, et il s'en souvient pendant quinze minutes. On ne le met que devant les commandes qui modifient le système : `apt install` en a besoin, `apt search` non.

Si vous préférez la souris, le Centre d'applications fait la même chose avec des fiches illustrées. Il propose surtout des snaps.

En résumé

- La touche Super ouvre la vue d'ensemble. On y tape le nom de ce que l'on cherche.
- Ctrl+Alt+T ouvre un terminal. Tab complète, Ctrl+R cherche dans l'historique, Ctrl+C interrompt.
- L'invite `utilisateur@machine:dossier$` dit qui vous êtes, sur quelle machine et dans quel dossier. Un `#` final signale l'administrateur.
- `pwd`, `ls`, `cd` pour se déplacer ; `cat`, `less`, `head`, `tail` pour lire.
- `man` pour la référence, `--help` pour le résumé, `apropos` pour trouver une commande, `tldr` pour des exemples.
- `nano` pour modifier un fichier. De `vim`, sachez au moins sortir : Échap, puis `:q!`.
- APT pour les outils du système, Snap ou Flatpak pour les applications de bureau récentes.

Atelier — Explorer sans rien casser

Objectif : mettre en pratique le chapitre. Comptez quarante-cinq minutes. Tout se fait en lecture seule, sauf la dernière partie, dans votre dossier personnel.

Se déplacer

1. Ouvrez un terminal. Tapez `pwd`, puis `ls -lah`.
2. Allez à la racine du système avec `cd /` et listez son contenu. Le chapitre 4 explique chacun de ces dossiers.

3. Allez dans `/etc`, affichez `os-release`, puis revenez dans votre dossier personnel avec `cd` seul.
4. Tapez `cd -`. Où êtes-vous ?

S'informer

5. Affichez votre identifiant, le nom de la machine, la version du noyau et la date au format `2026-04-23`.
6. Affichez vos dix dernières commandes.
7. Avec Ctrl+R, retrouvez la commande qui contenait `os-release`.

Chercher de l'aide

8. Dans `man ls`, trouvez l'option qui trie les fichiers par date de modification.
9. Avec `apropos`, trouvez une commande qui affiche l'espace disque disponible (le mot à chercher est *disk*).
10. Installez `tldr` et lisez la page de `cp`.

Modifier un fichier

11. Créez `premier.txt` avec nano. Écrivez trois lignes, enregistrez, quittez.
12. Affichez-le avec `cat`, puis comptez ses lignes avec `wc -l premier.txt`.
13. Supprimez-le avec `rm premier.txt` et vérifiez avec `ls`.

Exercices

Exercice 1 — Lire une invite

```
root@web-01:/var/log/nginx#
```

1. Quel utilisateur est connecté, et sur quelle machine ?
2. Dans quel dossier se trouve-t-il ?
3. À quoi voit-on qu'il a les droits d'administration ? Deux indices.
4. À votre avis, que vient-il faire là ?

Exercice 2 — Quelle commande ?

1. Afficher le nom de la machine.
2. Afficher la date sous la forme `2026-04-23`.

3. Afficher les vingt dernières commandes tapées.
4. Suivre en direct ce qui s'écrit dans `/var/log/syslog`.
5. Revenir dans son dossier personnel, où que l'on soit.
6. Retourner dans le dossier où l'on était juste avant.

Exercice 3 — Chercher seul

1. Quelle option de `grep` rend la recherche insensible à la casse ?
2. D'après `man 5 passwd`, combien de champs contient une ligne de `/etc/passwd` ?
3. Avec `apropos`, trouvez deux commandes qui servent à créer un utilisateur.

Exercice 4 — Une fiche d'identité de la machine

Créez un fichier `~/machine.txt` contenant votre identifiant, le nom de la machine, la version du noyau et la date. Faites-le d'abord avec nano, en recopiant le résultat de chaque commande. Puis essayez ceci, et observez le résultat avec `cat` :

```
echo "Utilisateur : $(whoami)" > ~/machine.txt
echo "Machine : $(hostname)" >> ~/machine.txt
```

Complétez avec deux lignes de votre cru. Que font `>` et `>>`, d'après vous ? Le chapitre 5 confirmera.

Exercice 5 — APT, Snap ou Flatpak ?

Quelle méthode choisiriez-vous, et pourquoi ?

1. `jq`, un outil en ligne de commande utilisé par un script du système.
2. Visual Studio Code, que vous voulez toujours à jour.
3. Une application de bureau dont la version des dépôts d'Ubuntu a deux ans de retard.
4. Un serveur web pour un serveur de production.

Pour aller plus loin

Le bureau. `help.ubuntu.com` pour le guide du bureau d'Ubuntu, en français. `extensions.gnome.org` pour le catalogue des extensions.

Le terminal. *The Linux Command Line*, de William Shotts, gratuit sur `linuxcommand.org` : le meilleur cours d'initiation qui existe, en anglais. `explainshell.com` : collez-y une ligne de commande, le site en explique chaque morceau.

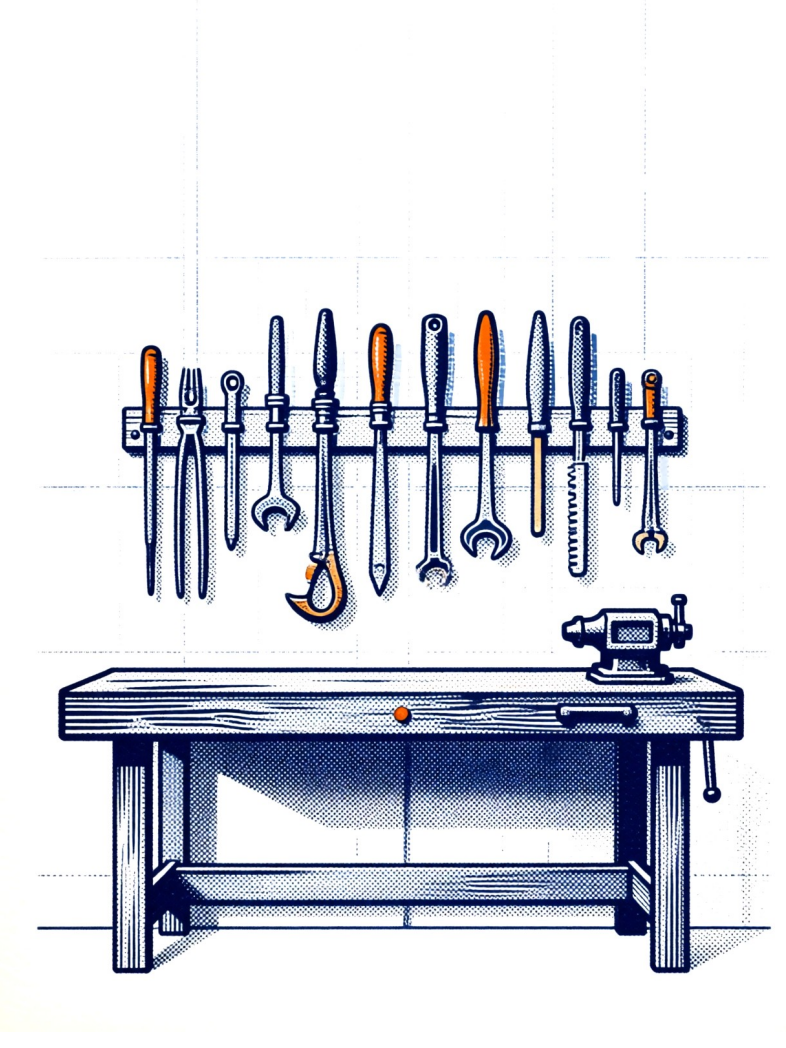
Les éditeurs. `vimtutor`, dans votre terminal, pour apprendre vim en trente minutes. Dans Emacs, Ctrl+H puis T ouvre le tutoriel intégré.

Les logiciels. `snapcraft.io/store` et `flathub.org` pour parcourir les catalogues Snap et Flatpak.

PARTIE II

L'environnement Ubuntu

Comprendre comment un système Linux est organisé et apprendre à le manipuler en ligne de commande.





CHAPITRE 4

Le système de fichiers

Une seule arborescence, où chaque dossier a un rôle précis. Une fois la carte en tête, on sait toujours où chercher.

DANS CE CHAPITRE

- Expliquer ce que signifie « tout est fichier » et ce que l'on y gagne.
- Dire à quoi sert chaque grand dossier du système : `/etc`, `/var`, `/usr`, `/home`, `/proc`, `/dev` et les autres.
- Passer d'un chemin absolu à un chemin relatif, et savoir lequel employer.
- Créer des liens symboliques et des liens physiques, et comprendre ce qui les distingue.
- Reconnaître le type d'un fichier avec `ls -l`, `file` et `stat`.
- Comprendre ce qu'est un inode, et les comportements qu'il explique.

> **AVANT DE COMMENCER**
>
> Le chapitre 3 : vous savez vous déplacer avec `cd`, lister avec `ls` et lire un fichier avec `cat`. Ce chapitre ne contient aucune commande dangereuse. Explorez sans retenue.

4.1 « Tout est fichier »

Cette formule revient dans tous les livres sur Unix. Elle résume un choix de conception qui explique la cohérence du système, et quelques-unes de ses bizarreries.

Sous Linux, un fichier n'est pas seulement un document rangé sur un disque. C'est tout objet qui porte un nom dans l'arborescence, et dans lequel on peut lire ou écrire. Cette définition englobe beaucoup de choses :

- les documents ordinaires ;
- les dossiers, qui sont des fichiers contenant une liste de noms ;
- les périphériques : le premier disque s'appelle `/dev/sda` ou `/dev/nvme0n1`, le terminal où vous tapez s'appelle `/dev/pts/0` ;
- l'état du noyau et du matériel, que l'on consulte sous `/proc` et `/sys` ;
- les canaux par lesquels les programmes se parlent : tubes et sockets.

L'intérêt est immédiat. Les mêmes commandes servent à tout. Avec `cat`, on lit un fichier texte, mais aussi l'état de la mémoire :

```
gary@vostro:~$ grep "model name" /proc/cpuinfo | head -n 1
model name      : Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz
gary@vostro:~$ head -n 3 /proc/meminfo
MemTotal:       16056900 kB
MemFree:          572724 kB
MemAvailable:    3384772 kB
gary@vostro:~$ cat /sys/class/thermal/thermal_zone0/temp
52000
```

Aucun de ces « fichiers » n'existe sur le disque. Le noyau fabrique leur contenu au moment où on les lit. Le dernier donne la température du processeur en millièmes de degré, soit ici 52 °C.

On peut aussi écrire. Ouvrez deux terminaux, tapez `tty` dans le second pour connaître son nom, puis depuis le premier :

```
echo "Bonjour depuis l'autre fenêtre" > /dev/pts/1
```

Le texte s'affiche dans l'autre terminal. Vous venez d'écrire dans un périphérique comme dans un fichier.

> **REPÈRE HISTORIQUE · Une idée de 1969**
>
> Thompson et Ritchie voulaient un système que deux ou trois personnes puissent écrire et comprendre en entier. Présenter les disques, les terminaux et les communications entre programmes sous la même forme leur a permis de tout manipuler avec quelques opérations : ouvrir, lire, écrire, fermer. C'est pour cela que `grep`, `cat` ou `tail`, écrits dans les années 1970, servent encore à diagnostiquer un serveur aujourd'hui.

Le principe a ses limites. Une connexion réseau ou une fenêtre graphique ne se manipulent pas comme des fichiers. Mais pour tout ce qui touche à la configuration, aux journaux, aux périphériques et à l'état du système, il tient. C'est l'essentiel du travail d'administration.

4.2 L'arborescence standard

Windows attribue une lettre à chaque disque : `C:`, `D:`, `E:`. Linux procède autrement. Il n'existe qu'une seule arborescence, qui part de la racine, notée `/`. Les autres disques, les clés USB et les partages réseau viennent s'y greffer à un endroit donné. On dit qu'ils y sont *montés*.

L'organisation de cette arborescence suit une norme, le *Filesystem Hierarchy Standard* (FHS), que toutes les grandes distributions respectent. Ce que vous apprenez ici vaut donc aussi pour Debian, Fedora ou Arch.

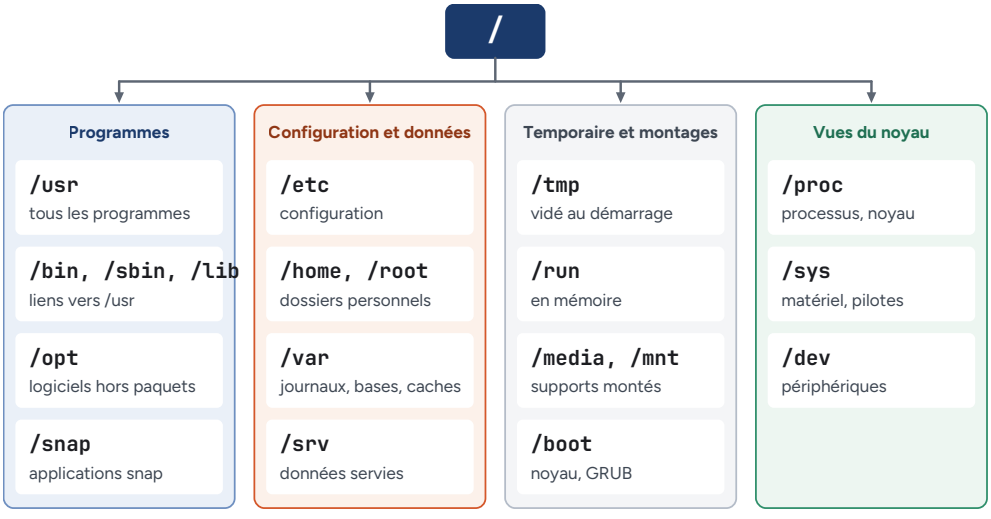


Figure 4.1 — Les grands dossiers d'un système Ubuntu, regroupés selon leur rôle.

Dossier	Ce qu'on y trouve
`/etc`	La configuration du système et des services, en fichiers texte.
`/home`	Les dossiers personnels des utilisateurs.
`/root`	Le dossier personnel de l'administrateur `root`.
`/usr`	Les programmes, les bibliothèques et leurs données. Géré par le système de paquets.
`/bin`, `/sbin`, `/lib`	Des liens vers `/usr/bin`, `/usr/sbin` et `/usr/lib`.
`/var`	Les données qui changent en permanence : journaux, caches, bases de données.
`/tmp`	Les fichiers temporaires, vidés à chaque démarrage.
`/run`	Les données d'exécution des services, en mémoire.
`/boot`	Le noyau et ce qu'il faut pour démarrer.
`/dev`	Les périphériques.
`/proc`, `/sys`	L'état du noyau, des processus et du matériel.
`/opt`	Les logiciels installés en dehors du système de paquets.
`/srv`	Les données servies par la machine, par exemple des sites web.
`/media`, `/mnt`	Les points de montage : supports amovibles, montages manuels.

Dossier	Ce qu'on y trouve
`/snap`	Les applications installées sous forme de snaps.

Les programmes : `/usr`, `/bin`, `/sbin`

Autrefois, Unix séparait les commandes indispensables au démarrage (`/bin`, `/sbin`) de toutes les autres (`/usr/bin`, `/usr/sbin`), parce que `/usr` pouvait se trouver sur un second disque. Cette séparation n'a plus de raison d'être. Ubuntu a tout regroupé sous `/usr`, comme les autres distributions, et n'a gardé les anciens noms que sous forme de liens :

```
gary@vostro:~$ ls -ld /bin /sbin /lib
lrwxrwxrwx 1 root root 7 avril 22  2024 /bin -> usr/bin
lrwxrwxrwx 1 root root 7 avril 22  2024 /lib -> usr/lib
lrwxrwxrwx 1 root root 8 avril 22  2024 /sbin -> usr/sbin
```

`/bin/ls` et `/usr/bin/ls` désignent donc le même programme.

À l'intérieur de `/usr` :

- `/usr/bin` contient presque toutes les commandes ;
- `/usr/sbin` contient celles qui servent à l'administration (`fdisk`, `useradd`) ;
- `/usr/lib` contient les bibliothèques ;
- `/usr/share` contient les données indépendantes du processeur : documentation, icônes, traductions ;
- `/usr/local` reproduit la même structure (`bin`, `lib`, `share`) pour ce que *vous* installez à la main. Le système de paquets n'y touche jamais.

La configuration : `/etc`

S'il ne fallait retenir qu'un dossier, ce serait celui-ci. `/etc` contient la configuration de tout le système, sous forme de fichiers texte que n'importe quel éditeur ouvre. Il n'y a pas de base de registre.

Quelques fichiers que vous retrouverez tout au long du livre :

Fichier ou dossier	Contenu
`/etc/os-release`	La version de la distribution
`/etc/hostname`, `/etc/hosts`	Le nom de la machine, des correspondances entre noms et adresses IP

Fichier ou dossier	Contenu
`/etc/passwd`, `/etc/group`	Les utilisateurs et les groupes
`/etc/shadow`	Les mots de passe chiffrés, lisibles par `root` seulement
`/etc/fstab`	Les systèmes de fichiers à monter au démarrage
`/etc/apt/`	Les dépôts de paquets
`/etc/ssh/sshd_config`	La configuration du serveur SSH
`/etc/netplan/`	La configuration du réseau
`/etc/systemd/`	La configuration de systemd et les services que vous ajoutez

> **ATTENTION · Faites une copie avant de modifier**
>
> Un fichier de `/etc` mal écrit peut empêcher un service de démarrer, couper le réseau ou vous interdire l'accès à la machine. Avant chaque modification, gardez l'original : `sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.orig`. Le chapitre 27 montre une méthode plus solide, avec Git.

Les utilisateurs : `/home` et `/root`

Chaque utilisateur possède un dossier à son nom dans `/home`. Le tilde `~` en est l'abréviation : pour l'utilisateur `gary`, `~` vaut `/home/gary`.

```
gary@vostro:~$ ls -l /home
total 8
drwxr-x--- 5 alice alice 4096 avril 12 14:20 alice
drwxr-x--- 18 gary gary  4096 avril 22 11:45 gary
```

Regardez les droits, `drwxr-x---` : sur Ubuntu 24.04, un utilisateur ne peut pas entrer dans le dossier d'un autre.

Votre dossier personnel contient aussi des fichiers et dossiers cachés, dont le nom commence par un point. Ils portent vos réglages :

- `~/.bashrc` et `~/.profile` : la configuration de votre shell (chapitre 8) ;
- `~/.config/` : les réglages de vos applications ;
- `~/.ssh/` : vos clés SSH ;
- `~/.cache/` : des données temporaires que l'on peut supprimer ;
- `~/.local/` : les programmes et données installés pour vous seul.

L'administrateur `root` a son dossier à part, `/root`, et non dans `/home`. Si `/home` se trouve sur un disque qui ne répond plus, `root` peut ainsi toujours ouvrir une session pour réparer.

Les données qui bougent : `/var`

Le contenu de `/usr` et de `/etc` change rarement. Celui de `/var` change tout le temps.

- `/var/log/` : les journaux du système et des services ;
- `/var/lib/` : les données des services. Une base MySQL vit dans `/var/lib/mysql/`, l'état du système de paquets dans `/var/lib/dpkg/` ;
- `/var/cache/` : des données que l'on peut reconstruire, comme les paquets téléchargés par APT ;
- `/var/spool/` : les files d'attente (impressions, courrier, tâches planifiées) ;
- `/var/www/` : l'emplacement par défaut des sites servis par Apache ou Nginx ;
- `/var/tmp/` : des fichiers temporaires conservés d'un démarrage à l'autre.

Sur un serveur, c'est `/var` qui grossit. Pour voir ce qui occupe la place :

```
gary@vostro:~$ sudo du -h --max-depth=1 /var | sort -hr | head -n 5
6,1G    /var
4,2G    /var/lib
1,3G    /var/log
512M    /var/cache
24K     /var/spool
```

> **DANGER · Le disque plein, première cause de panne**
>
> Un programme qui écrit la même erreur mille fois par seconde dans son journal peut remplir un petit serveur en une nuit. Un disque plein, ce sont des bases de données qui s'arrêtent et des connexions qui échouent. Surveillez `/var/log`. Le chapitre 13 explique comment limiter la taille des journaux.

Le temporaire : `/tmp` et `/run`

`/tmp` est ouvert en écriture à tout le monde. Sur Ubuntu 24.04, il se trouve sur le disque, avec le reste du système. Il est vidé à chaque démarrage, et les fichiers qui n'ont pas servi depuis trente jours en sont retirés. N'y laissez rien que vous vouliez garder.

```
gary@vostro:~$ ls -ld /tmp
drwxrwxrwt 41 root root 57344 oct.   4 20:32 /tmp
```

Le `t` final des droits est le *sticky bit*. Dans un dossier ainsi marqué, chacun peut créer des fichiers, mais ne peut supprimer que les siens. Le chapitre 6 y revient.

`/run` contient ce dont les services ont besoin pendant qu'ils tournent : numéros de processus, sockets. Il réside en mémoire et repart de zéro à chaque démarrage.

Les fenêtres sur le noyau : `/proc`, `/sys`, `/dev`

Ces trois dossiers n'occupent aucune place sur le disque. Le noyau les construit en mémoire.

`/proc` décrit les processus et le noyau. Chaque processus y a un dossier qui porte son numéro, et quelques fichiers donnent l'état général de la machine :

```
cat /proc/loadavg      # la charge du système
cat /proc/uptime       # le temps écoulé depuis le démarrage, en secondes
cat /proc/version      # la version du noyau
cat /proc/cmdline      # les paramètres passés au noyau au démarrage
ls /proc/1/            # tout ce que le noyau sait du processus numéro 1
```

`/sys` décrit le matériel et les pilotes :

```
gary@vostro:~$ ls /sys/class/net/
enp3s0  lo  wlp2s0
gary@vostro:~$ cat /sys/class/dmi/id/product_name
Vostro 5401
```

`/dev` contient un fichier par périphérique, et quelques fichiers spéciaux dont on se sert tous les jours :

Fichier	Ce que c'est
`/dev/sda`, `/dev/sda1`	Un disque SATA ou USB, et sa première partition
`/dev/nvme0n1`, `/dev/nvme0n1p1`	Un disque NVMe, et sa première partition
`/dev/null`	Un puits sans fond : tout ce qu'on y écrit disparaît
`/dev/zero`	Une source inépuisable d'octets nuls
`/dev/urandom`	Une source inépuisable d'octets aléatoires
`/dev/pts/0`	Un terminal

```
# Faire taire une commande : on envoie tout ce qu'elle affiche dans /dev/null
commande_bavarde > /dev/null 2>&1

# Fabriquer un fichier d'un mégaoctet de données aléatoires
head -c 1M /dev/urandom > essai.bin
```

Les autres dossiers

`/boot` contient le noyau (`vmlinuz-…`), le système minimal chargé au démarrage (`initrd.img-…`) et la configuration de GRUB. Chaque mise à jour du noyau y ajoute des fichiers. Ne les supprimez jamais à la main : `sudo apt autoremove` retire proprement les anciens noyaux.

`/opt` accueille les logiciels fournis hors des dépôts, un sous-dossier par logiciel. Google Chrome, par exemple, s'installe dans `/opt/google/chrome/`.

`/srv` est prévu pour les données que la machine met à disposition : sites web, dépôts. Il est souvent vide, car beaucoup d'administrateurs utilisent `/var/www` par habitude.

`/media` reçoit les supports amovibles. Une clé USB branchée sur Ubuntu Desktop apparaît sous `/media/gary/NOM_DE_LA_CLE/`.

`/mnt` sert aux montages que l'on fait à la main, pour un moment :

```
sudo mount -o loop ubuntu-24.04.5-live-server-amd64.iso /mnt
ls /mnt
sudo umount /mnt
```

`/snap` contient les snaps. Chacun est une image en lecture seule montée dans l'arborescence. Les nombreuses lignes `/dev/loop…` que montrent `df` et `mount` viennent de là.

> **ASTUCE · La page de manuel de l'arborescence**
>
> `man hier` décrit le rôle de chaque dossier. Pensez-y quand vous hésitez.

4.3 Chemins absolus et chemins relatifs

Il y a deux façons de désigner un fichier.

Un **chemin absolu** part de la racine et commence donc par `/`. Il désigne toujours le même fichier, où que vous soyez : `/etc/hostname`, `/home/gary/Documents/rapport.pdf`.

Un **chemin relatif** part du dossier courant, celui qu'affiche `pwd`. Il ne commence pas par `/`. Le même chemin relatif désigne donc des fichiers différents selon l'endroit où l'on se trouve :

```
gary@vostro:~$ ls Documents/rapport.pdf
Documents/rapport.pdf
gary@vostro:~$ cd /tmp
gary@vostro:/tmp$ ls Documents/rapport.pdf
ls: impossible d'accéder à 'Documents/rapport.pdf': Aucun fichier ou dossier de ce nom
```

Quatre notations abrègent les chemins :

Notation	Signification	Exemple
`.`	Le dossier courant	`./script.sh` lance le script situé ici
`..`	Le dossier parent	`cd ../..` remonte de deux niveaux
`~`	Votre dossier personnel	`cp /tmp/notes.txt ~/Documents/`
`~alice`	Le dossier personnel d'alice	`ls ~alice`

Lequel utiliser ? Au clavier, le chemin relatif va plus vite. Dans un script, un fichier de configuration ou une tâche planifiée, écrivez des chemins absolus : vous ne savez pas depuis quel dossier ils seront exécutés.

> **DANGER · Les tâches planifiées et les chemins relatifs**
>
> Un script lancé par le planificateur `cron` ne démarre pas dans le dossier où vous l'avez écrit. S'il ouvre `donnees/clients.csv`, il ne trouvera rien. C'est l'erreur la plus fréquente avec les tâches planifiées. Le chapitre 13 y revient.

4.4 Fichiers cachés et liens

Les fichiers cachés

Un fichier dont le nom commence par un point est *caché*. Il n'y a là ni attribut spécial ni protection. C'est une simple convention : `ls` et le gestionnaire de fichiers ne montrent pas ces noms, sauf si on le demande.

```
gary@vostro:~$ ls -A
.bash_history  .bashrc  .config  .profile  Bureau     Images
.bash_logout   .cache   .local   .ssh      Documents  Vidéos
```

`ls -a` affiche tout, y compris `.` et `..`, deux entrées présentes dans chaque dossier et qui désignent le dossier lui-même et son parent. `ls -A` les omet. Dans le gestionnaire de fichiers, Ctrl+H affiche ou masque les fichiers cachés.

Les liens symboliques

Un lien symbolique est un petit fichier qui contient le chemin d'un autre. Quand on l'ouvre, le système va chercher la cible. C'est l'équivalent d'un raccourci.

```
gary@vostro:~$ ln -s /var/log journaux
gary@vostro:~$ ls -l journaux
lrwxrwxrwx 1 gary gary 8 avril 23 11:40 journaux -> /var/log
gary@vostro:~$ ls journaux/
alternatives.log  auth.log  dpkg.log  kern.log  syslog
```

La syntaxe est `ln -s cible nom_du_lien`, dans cet ordre. Pour s'en souvenir : c'est le même ordre que `cp`, d'abord ce qui existe, puis ce que l'on crée.

Dans la sortie de `ls -l`, on reconnaît un lien au `l` initial et à la flèche. Supprimer le lien avec `rm` ne touche pas à la cible.

Le système est plein de liens symboliques. `/bin` en est un. Activer un service avec `systemctl enable` en crée un autre. Et quand plusieurs versions d'un logiciel cohabitent, un lien au nom générique désigne celle qui est en service.

Si la cible disparaît, le lien reste, mais il ne mène plus nulle part :

```
gary@vostro:~$ cat lien-casse
cat: lien-casse: Aucun fichier ou dossier de ce nom
```

Le message est trompeur. C'est la cible qui manque, pas le lien. La commande `find . -xtype l` retrouve les liens cassés d'une arborescence.

Les liens physiques

Un lien physique (en anglais *hard link*) est d'une autre nature. Ce n'est pas un fichier qui en désigne un autre. C'est un second nom pour le même fichier. Il n'y a ni original ni copie : les deux noms sont à égalité.

```
gary@vostro:~$ echo bonjour > a.txt
gary@vostro:~$ ln a.txt b.txt
gary@vostro:~$ ls -li a.txt b.txt
9459587 -rw-rw-r-- 2 gary gary 8 avril 23 11:52 a.txt
9459587 -rw-rw-r-- 2 gary gary 8 avril 23 11:52 b.txt
```

L'option `-i` affiche en première colonne un numéro, identique pour les deux noms. La troisième colonne, qui vaut 2, compte les noms qui mènent à ce fichier. Supprimons-en un :

```
gary@vostro:~$ rm a.txt
gary@vostro:~$ cat b.txt
bonjour
```

Le contenu est toujours là. Il ne disparaîtra que lorsque son dernier nom sera supprimé.

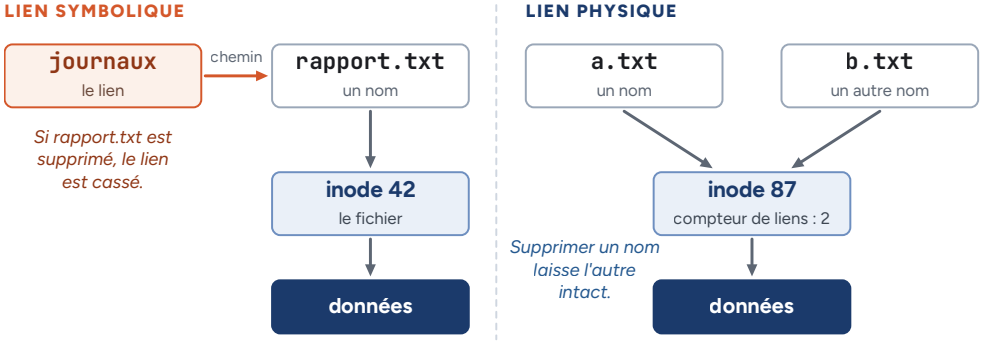


Figure 4.2 — Lien symbolique et lien physique : le premier désigne un nom, le second désigne directement les données.

	Lien symbolique	Lien physique
Commande	`ln -s cible lien`	`ln cible lien`
Peut désigner un dossier	Oui	Non
Peut franchir la limite d'un disque ou d'une partition	Oui	Non
Si l'on supprime la cible	Le lien est cassé	Les données restent accessibles
Dans `ls -l`	`l` initial et flèche	Rien de visible, sauf le compteur de liens

Dans la pratique, on utilise presque toujours des liens symboliques. Les liens physiques servent surtout aux outils de sauvegarde, qui s'en servent pour ne pas recopier les fichiers inchangés d'un jour sur l'autre (chapitre 16).

4.5 Les types de fichiers

Le premier caractère de chaque ligne de `ls -l` indique le type du fichier. Il en existe sept.

Caractère	Type	Exemple
`-`	Fichier ordinaire	`/etc/hostname`
`d`	Dossier	`/home`
`l`	Lien symbolique	`/bin`
`b`	Périphérique en mode bloc (disques)	`/dev/sda`
`c`	Périphérique en mode caractère (terminaux)	`/dev/null`
`s`	Socket, pour la communication entre programmes	`/run/systemd/notify`
`p`	Tube nommé	créé avec `mkfifo`

```
gary@vostro:~$ ls -l /dev/sda /dev/null
crw-rw-rw- 1 root root 1, 3 oct.   3 08:08 /dev/null
brw-rw---- 1 root disk 8, 0 oct.   3 08:08 /dev/sda
```

Deviner le contenu : `file`

Sous Linux, l'extension d'un nom de fichier n'est qu'une indication, que rien n'oblige à respecter. La commande `file` examine le contenu :

```
gary@vostro:~$ file /etc/hostname /usr/bin/ls /etc/os-release /dev/null
/etc/hostname:   ASCII text
/usr/bin/ls:     ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, ...
/etc/os-release: symbolic link to ../usr/lib/os-release
/dev/null:       character special (1/3)
```

Tout savoir d'un fichier : `stat`

```
gary@vostro:~$ stat /etc/hostname
  Fichier : /etc/hostname
   Taille : 7           Blocs : 8          Blocs d'E/S : 4096   fichier
Périphérique : 259/4    Inœud : 4981686     Liens : 1
Accès : (0644/-rw-r--r--)  UID : (    0/    root)   GID : (    0/    root)
Accès : 2026-04-23 10:33:28.116725615 +0100
Modif. : 2026-04-20 09:14:06.606050401 +0100
Changt : 2026-04-20 09:14:06.606050401 +0100
  Créé : 2026-04-20 09:14:06.605980146 +0100
```

On y lit la taille, les droits, le propriétaire, le numéro d'inode (ici orthographié « inœud ») et quatre dates :

- **Accès** : la dernière lecture du contenu ;
- **Modif.** : la dernière modification du contenu. C'est la date qu'affiche `ls -l` ;
- **Changt** : la dernière modification des informations qui décrivent le fichier, c'est-à-dire ses droits, son propriétaire ou son nom. Un `chmod` change cette date sans toucher à la précédente ;
- **Créé** : la date de création.

> **NOTE · La date d'accès n'est pas exacte**
>
> Écrire sur le disque à chaque lecture coûterait cher. Linux ne met donc à jour la date d'accès qu'une fois par jour au plus, ou quand le fichier a été modifié depuis sa dernière lecture. Ne comptez pas sur elle pour savoir précisément quand un fichier a été lu.

4.6 L'inode

Le mot est apparu deux fois déjà. Il désigne la structure qui se trouve au cœur des systèmes de fichiers Unix, et qui explique plusieurs comportements surprenants.

Le nom d'un côté, le fichier de l'autre

On imagine volontiers qu'un fichier est un contenu auquel on a collé un nom. Sous Linux, ce sont deux choses séparées.

L'**inode** est une fiche qui décrit le fichier : son type, sa taille, son propriétaire, ses droits, ses dates et l'emplacement de ses données sur le disque. Chaque inode porte un numéro. Il contient tout, sauf le nom.

Le **nom**, lui, est rangé dans un dossier. Un dossier n'est rien d'autre qu'une table à deux colonnes : des noms, et en face de chacun un numéro d'inode.

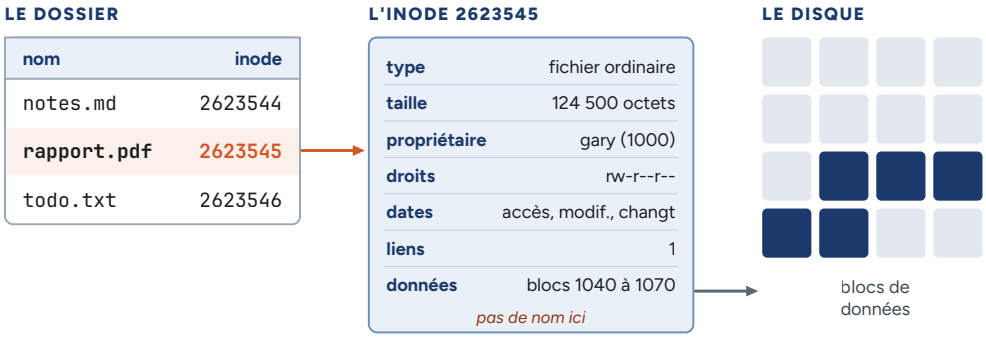


Figure 4.3 — Le nom se trouve dans le dossier ; tout le reste se trouve dans l'inode.

Un lien physique, on le comprend maintenant, est une seconde ligne dans un dossier, qui porte le même numéro d'inode.

Ce que l'inode explique

Renommer est instantané. `mv gros-film.mkv film.mkv` modifie une ligne dans un dossier. Les données ne bougent pas, que le fichier pèse 4 ko ou 40 Go.

Déplacer vers un autre disque prend du temps. Chaque système de fichiers a sa propre table d'inodes. `mv ~/film.mkv /media/gary/CLE/` doit donc recopier toutes les données, puis supprimer l'original.

Supprimer ne libère pas toujours la place. `rm` retire un nom. Les données ne sont effacées que lorsque plus aucun nom ne mène à l'inode et que plus aucun programme ne tient le fichier ouvert. C'est un classique des serveurs : on supprime un journal énorme pour faire de la place, et l'espace ne revient pas, parce que le service écrit toujours dedans. Il faut redémarrer ce service, ou mieux, vider le fichier au lieu de le supprimer :

```
sudo truncate -s 0 /var/log/application/enorme.log
```

On peut manquer d'inodes avant de manquer de place. Leur nombre est fixé à la création du système de fichiers. Des millions de fichiers minuscules peuvent l'épuiser alors qu'il reste des gigaoctets libres. Le message d'erreur est le même, *No space left on device*, et `df -h` n'y comprend rien. C'est `df -i` qui donne la réponse :

```
gary@vostro:~$ df -i /
Sys. de fichiers  Inœuds IUtil.  ILibre IUti% Monté sur
/dev/nvme0n1p2   9601024 706971 8894053    8% /
```

Une colonne `IUti%` à 100 % signifie qu'il faut supprimer des fichiers, et peu importe leur taille. Les coupables habituels sont des dossiers de cache ou de sessions jamais nettoyés.

En résumé

- Sous Linux, les périphériques, l'état du noyau et les canaux de communication se présentent comme des fichiers. Les mêmes commandes servent à tout.
- Il n'y a qu'une arborescence, qui part de `/`. Les autres disques y sont montés.
- `/etc` pour la configuration, `/var` pour ce qui change, `/usr` pour les programmes, `/home` pour les utilisateurs, `/tmp` pour le temporaire, `/proc`, `/sys` et `/dev` pour le noyau et le matériel.
- Un chemin absolu commence par `/`. Un chemin relatif part du dossier courant. Dans un script, on écrit des chemins absolus.
- Un nom qui commence par un point est caché. `ls -A` le montre.
- Un lien symbolique désigne un chemin et peut se casser. Un lien physique est un second nom pour les mêmes données.
- Le premier caractère de `ls -l` donne le type du fichier. `file` examine le contenu, `stat` affiche toutes ses caractéristiques.
- L'inode décrit le fichier, le dossier contient son nom. Cela explique qu'un renommage soit instantané, qu'un `rm` ne libère pas toujours la place et qu'on puisse manquer d'inodes.

Atelier — Arpenter l'arborescence

Objectif : parcourir le système et vérifier par vous-même ce que dit le chapitre. Comptez quarante-cinq minutes.

Voir l'arbre

1. Installez `tree` avec `sudo apt install tree`, puis lancez `tree -L 1 /`. Retrouvez chaque dossier dans le tableau de la section 4.2.

2. Lancez `tree -L 2 /etc | less`. Repérez les dossiers `apt`, `ssh`, `systemd` et `netplan`.

Interroger le système

3. Affichez le modèle de votre processeur, la mémoire totale et le temps écoulé depuis le démarrage, à partir de `/proc`.
4. Listez vos interfaces réseau à partir de `/sys`.
5. Lancez `ls -l /dev | less` et trouvez un périphérique en mode bloc et un périphérique en mode caractère.
6. Combien de noyaux sont installés ? Regardez dans `/boot`.

Fabriquer des liens

7. Dans un dossier `~/essais`, créez `original.txt` avec une ligne de texte.
8. Créez-en un lien symbolique `sym.txt` et un lien physique `phy.txt`.
9. Avec `ls -li`, comparez les numéros d'inode et les compteurs de liens.
10. Ajoutez une ligne à `original.txt` avec `echo "deuxième ligne" >> original.txt`. Lisez les trois fichiers.
11. Supprimez `original.txt`. Lequel des deux liens fonctionne encore ? Pourquoi ?

Construire une arborescence

12. Lancez `mkdir -p ~/essais/projet/{scripts,notes,archives}` puis `tree ~/essais`. Les accolades ont créé trois dossiers d'un coup. Le chapitre 5 explique comment.
13. Faites le ménage : `rm -r ~/essais`.

Exercices

Exercice 1 — Où chercher ?

Dans quel dossier iriez-vous chercher, ou ranger :

1. la configuration du serveur SSH ;
2. les journaux d'un serveur web ;
3. votre clé SSH privée ;
4. le noyau installé ;
5. une clé USB que vous venez de brancher ;
6. un programme que vous avez compilé vous-même ;

7. le cache de Firefox ;
8. la liste des utilisateurs ;
9. un logiciel commercial fourni dans une archive ;
10. les informations sur un processus en cours ?

Exercice 2 — Lire `ls -l`

Pour chaque ligne, dites de quel type de fichier il s'agit et ce que c'est probablement.

```
-rw-r--r-- 1 gary gary  1234 avril 10 14:00 rapport.pdf
drwxr-xr-x 3 gary gary  4096 avril 10 14:00 projets
lrwxrwxrwx 1 root root     7 avril 22  2024 bin -> usr/bin
brw-rw---- 1 root disk  8, 0 avril 10 08:00 sda
crw-rw-rw- 1 root root  1, 3 avril 10 08:00 null
-rw------- 1 gary gary   654 avril 22 11:00 .bash_history
srw-rw-rw- 1 root root     0 avril 23 08:00 notify
```

Exercice 3 — Du relatif à l'absolu

Vous êtes dans `/home/gary/projets/site/backend`. Donnez le chemin absolu correspondant à :

1. `./start.sh`
2. `../frontend/index.html`
3. `../../archives/`
4. `~/Documents/`
5. `config/nginx.conf`
6. `../../../../etc/hostname` (attention, comptez bien)

Exercice 4 — Vrai ou faux ?

1. Un lien symbolique peut désigner un dossier.
2. Un lien physique peut désigner un dossier.
3. Quand on supprime le fichier d'origine, un lien physique vers lui cesse de fonctionner.
4. Deux noms qui portent le même numéro d'inode désignent le même fichier.
5. `ln -s /tmp/nexiste-pas lien` échoue, car la cible n'existe pas.

Exercice 5 — Raisonner avec les inodes

1. Pourquoi `mv video.mkv film.mkv` est-il instantané pour un fichier de 4 Go ?

2. Pourquoi `mv ~/video.mkv /media/gary/CLE/` prend-il plusieurs minutes ?
3. Vous supprimez un journal de 10 Go, et `df -h` affiche toujours un disque plein. Que se passe-t-il ? Que faites-vous ?
4. Un serveur refuse de créer un fichier vide alors que `df -h` annonce 20 Go libres. Quelle commande lancez-vous ?

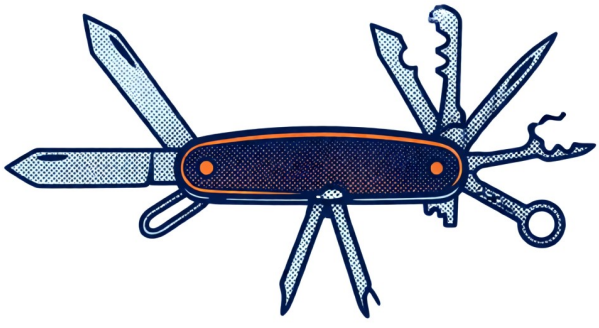
Pour aller plus loin

Sur la machine. `man hier` pour l'arborescence, `man 5 proc` pour tout ce que contient `/proc`, `man 7 inode` pour le détail de ce qu'enregistre un inode.

En ligne. `refspecs.linuxfoundation.org/fhs.shtml` pour le texte du Filesystem Hierarchy Standard. `wiki.archlinux.org/title/File_systems` pour un panorama des systèmes de fichiers, valable aussi sur Ubuntu.

Outils. `ncdu` (`sudo apt install ncdu`) parcourt un dossier et montre ce qui prend de la place, de façon interactive. Sur un disque plein, c'est le premier outil à lancer.

Lecture. *The Linux Programming Interface*, de Michael Kerrisk : la référence sur le fonctionnement interne de Linux. Ses chapitres 14 à 18 traitent des systèmes de fichiers, des inodes et des liens.



CHAPITRE 5

Les commandes essentielles du shell

Une cinquantaine de commandes, et surtout l'art de les combiner. C'est là que le terminal prend l'avantage sur la souris.

DANS CE CHAPITRE

- Créer, copier, déplacer et supprimer des fichiers et des dossiers.
- Lire un fichier de plusieurs façons, y compris quand il est compressé.
- Retrouver un fichier avec `find` et `locate`.
- Connaître l'état de la machine : charge, mémoire, disques, processus.
- Relier des commandes entre elles avec les redirections et les tubes.
- Désigner plusieurs fichiers d'un coup avec les motifs du shell.
- Créer un alias et lire une variable d'environnement.
- Créer et extraire des archives `tar` et `zip`.

> **AVANT DE COMMENCER**
>
> Les chapitres 3 et 4. Ce chapitre est long, car il sert aussi de référence. Parcourez-le une première fois en tapant les exemples, puis revenez-y quand vous aurez besoin d'une commande précise.

5.1 Lister et visualiser

Vous connaissez `ls`. Voici ses options les plus utiles, que l'on combine librement.

Option	Effet
`-l`	Une ligne par fichier, avec droits, propriétaire, taille et date
`-a`, `-A`	Montre les fichiers cachés ; `-A` omet `.` et `..`
`-h`	Tailles en K, M, G (avec `-l`)
`-t`	Trie par date, le plus récent en premier
`-S`	Trie par taille, le plus gros en premier
`-r`	Inverse l'ordre du tri
`-R`	Descend dans les sous-dossiers
`-d`	Montre le dossier lui-même, pas son contenu

```
ls -lah              # tout, en détail, avec des tailles lisibles
ls -lt | head        # les fichiers modifiés le plus récemment
ls -lSh | head       # les plus gros fichiers du dossier
ls -ld /etc          # les droits du dossier /etc lui-même
```

Pour voir une arborescence d'un coup d'œil, `tree` est irremplaçable :

```
# Installer tree
sudo apt install tree

# Arbre complet du dossier courant
tree
.
├── README.md
├── src
│   ├── index.js
│   ├── utils
│   │   ├── format.js
│   │   └── parse.js
```

```
│   └── config.json
└── package.json

# Limiter la profondeur à 2 niveaux
tree -L 2

# Afficher aussi les fichiers cachés
tree -a

# Afficher seulement les dossiers
tree -d

# Avec les tailles, lisibles
tree -sh

# Ignorer un dossier
tree -I 'node_modules|.git'
```

5.2 Créer, copier, déplacer, supprimer

Créer : `mkdir` et `touch`

```
# Créer un dossier
mkdir projet

# Créer plusieurs dossiers d'un coup
mkdir docs src tests

# Créer une arborescence (avec -p = parents)
mkdir -p projet/src/utils/helpers
# Crée tous les dossiers intermédiaires, sans erreur s'ils existent

# Avec les accolades, encore plus puissant (voir section 5.7)
mkdir -p projet/{src,tests,docs,config}
# Crée projet/src, projet/tests, projet/docs, projet/config

# Avec des permissions précises dès la création
mkdir -m 700 ~/.secrets    # dossier visible seulement par le propriétaire
```

L'option `-p` est à retenir. Elle crée les dossiers intermédiaires et ne se plaint pas s'ils existent déjà.

`touch` crée un fichier vide. S'il existe, il met simplement sa date à jour.

```
# Créer un fichier vide
touch notes.txt

# Créer plusieurs fichiers
```

```
touch a.txt b.txt c.txt

# Avec des accolades
touch {janvier,février,mars}.md

# Mettre à jour la date d'un fichier existant
touch README.md   # atime et mtime à maintenant

# Forcer une date précise
touch -d "2025-01-01 00:00" ancien.txt

# Mettre la date d'un autre fichier
touch -r reference.txt nouveau.txt
```

Copier : `cp`

```
# Copier un fichier
cp source.txt destination.txt

# Copier un fichier dans un dossier (garde le même nom)
cp rapport.pdf ~/Documents/

# Copier en renommant dans un dossier
cp rapport.pdf ~/Documents/rapport-final.pdf

# Copier plusieurs fichiers vers un dossier
cp fichier1.txt fichier2.txt fichier3.txt ~/backup/

# Copier un dossier (option -r = récursif, obligatoire)
cp -r projet/ projet-backup/

# Préserver permissions, propriétaire, dates (-a = archive)
cp -a projet/ projet-backup/

# Afficher chaque fichier copié
cp -rv src/ dest/

# Interactif : demander avant d'écraser
cp -i source.txt destination.txt

# Ne copier que si source plus récente
cp -u source.txt destination.txt
```

Deux options à connaître. `-r` est obligatoire pour copier un dossier. `-a` fait la même chose en conservant les droits, les propriétaires, les dates et les liens : c'est elle qu'il faut pour une copie fidèle.

Déplacer et renommer : `mv`

Une seule commande pour les deux, puisque renommer un fichier revient à le déplacer sous un autre nom (chapitre 4).

```
# Renommer un fichier
mv ancien.txt nouveau.txt

# Déplacer dans un dossier
mv rapport.pdf ~/Documents/

# Déplacer en renommant
mv rapport.pdf ~/Documents/rapport-final.pdf

# Déplacer plusieurs fichiers
mv a.txt b.txt c.txt ~/archives/

# Interactif : demander avant d'écraser
mv -i source.txt destination.txt

# Ne remplacer que si source plus récente
mv -u source.txt destination.txt

# Pas d'option -r nécessaire pour les dossiers (contrairement à cp)
mv vieux-dossier/ archives/
```

Supprimer : `rm` et `rmdir`

Il n'y a pas de corbeille en ligne de commande. Ce que `rm` supprime est perdu.

```
# Supprimer un fichier
rm fichier.txt

# Supprimer plusieurs fichiers
rm a.txt b.txt c.txt

# Supprimer un dossier vide (ou utiliser rmdir)
rm -d dossier-vide

# Supprimer un dossier et tout son contenu (récursif)
rm -r mon-dossier/

# Forcer : ne pas demander confirmation, ignorer les erreurs si absent
rm -f fichier-peut-etre-la.txt

# La combinaison redoutable (à utiliser avec prudence)
rm -rf ancien-projet/

# Interactif : demander avant chaque suppression
rm -i *.txt
```

```
# Afficher ce qui est supprimé
rm -rv ancien/
```

`rmdir` ne supprime que les dossiers vides. Il échoue sinon, ce qui en fait un garde-fou commode.

DANGER · rm -rf ne pardonne pas

`rm -rf dossier` supprime tout, sans rien demander. Les accidents viennent presque toujours d'un espace en trop. Avec `rm -rf ~ /essais`, le dossier personnel entier disparaît, alors qu'on visait `~/essais`. Trois habitudes à prendre :

- lancez d'abord `ls` avec le même argument, vérifiez la liste, puis rappelez la ligne et remplacez `ls` par `rm` ;
- regardez l'invite avant de valider : êtes-vous dans le bon dossier, sur la bonne machine ?
- ne tapez jamais `rm -rf` suivi d'une variable dont vous n'êtes pas sûr.

ASTUCE · Une corbeille pour le terminal

Le paquet `trash-cli` fournit la commande `trash-put`, qui envoie les fichiers dans la corbeille du bureau au lieu de les détruire. `trash-list` montre son contenu, `trash-restore` récupère un fichier.

5.3 Lire des fichiers

`cat`, `less`, `head`, `tail`

```
# Afficher un fichier
cat /etc/os-release

# Concaténer plusieurs fichiers
cat fichier1.txt fichier2.txt > fichier-combine.txt

# Avec numérotation des lignes
cat -n script.sh

# Afficher les caractères invisibles (fin de ligne, tabulations)
cat -A fichier.txt
```

Pour un fichier long, on utilise `less`. Ses touches sont les mêmes que dans les pages de manuel.

Touche	Effet dans `less`
Espace, b	Page suivante, page précédente
g, G	Début, fin du fichier
`/mot`, `?mot`	Chercher vers le bas, vers le haut
n, N	Occurrence suivante, précédente
F	Suivre le fichier à mesure qu'il grandit ; Ctrl+C pour reprendre la main
q	Quitter

```
# 10 premières lignes (par défaut)
head /var/log/syslog

# N premières lignes
head -n 5 /etc/passwd
head -5 /etc/passwd   # équivalent

# 10 dernières lignes
tail /var/log/syslog

# N dernières lignes
tail -n 20 /var/log/syslog

# Suivre un fichier en temps réel (-f = follow)
tail -f /var/log/syslog
# Ctrl+C pour quitter

# Suivre en attendant le fichier s'il n'existe pas encore
tail -F /var/log/mon-app.log

# Afficher à partir de la ligne 100
tail -n +100 fichier.log

# Combiner : lignes 50 à 60 d'un fichier
head -n 60 fichier.txt | tail -n 11
```

`tail -f` est la commande que l'on garde ouverte dans un coin pendant qu'on teste un service. Chaque nouvelle ligne du journal s'affiche dès qu'elle est écrite.

Les fichiers compressés

Les anciens journaux sont compressés pour gagner de la place. On les lit sans les décompresser :

```
# Lire un .gz
zcat /var/log/syslog.2.gz
zless /var/log/auth.log.3.gz    # navigable

# Pour .bz2
bzcat fichier.txt.bz2

# Pour .xz
xzcat fichier.txt.xz

# Rechercher dans un .gz sans décompresser
zgrep "ERROR" /var/log/syslog.*.gz
```

Compter : `wc`

```
gary@vostro:~$ wc /etc/passwd
  50   91 2934 /etc/passwd
gary@vostro:~$ wc -l /etc/passwd
50 /etc/passwd
```

Les trois nombres sont les lignes, les mots et les octets. `wc -l` ne garde que les lignes. On le place très souvent au bout d'un tube, pour compter des résultats.

5.4 Trouver des fichiers et des commandes

`find`

`find` parcourt une arborescence et retient les fichiers qui satisfont des critères. Sa forme générale est `find où critères action`.

```
# Trouver par nom
find /home/gary -name "*.pdf"

# Insensible à la casse
find . -iname "readme*"

# Par type
find /etc -type f    # fichiers uniquement
find /etc -type d    # dossiers uniquement
find / -type l       # liens symboliques

# Par taille
find ~ -size +100M           # > 100 Mo
find ~ -size -100k           # < 100 Ko
find ~ -type f -empty        # fichiers vides
find ~ -size +1G -size -5G   # entre 1 et 5 Go
```

```
# Par date de modification
find ~/Documents -mtime -7   # modifié dans les 7 derniers jours
find ~/Documents -mtime +30  # non modifié depuis 30 jours
find ~ -mmin -60             # modifié dans la dernière heure

# Par propriétaire
find /var -user gary
find /tmp -group nogroup

# Par permissions
find /etc -perm 644
find /home -perm -u+x       # au moins x pour le user

# Combiner des critères (-a = AND implicite, -o = OR)
find ~ -name "*.txt" -size +1M
find ~ \( -name "*.jpg" -o -name "*.png" \)

# Exclure un dossier
find / -name "*.conf" -not -path "*/snap/*"
```

ATTENTION · Le piège des tailles arrondies

`find` arrondit les tailles à l'unité supérieure avant de comparer. `-size -1k` ne trouve donc pas les fichiers de moins d'un kilooctet, mais seulement les fichiers vides. Pour les petits fichiers, comptez en octets, avec le suffixe `c` : `-size -1024c`.

Une fois les fichiers trouvés, on peut agir dessus :

```
# Juste afficher le chemin (par défaut)
find ~ -name "*.log"

# Afficher avec ls -l (détails)
find /var/log -name "*.log" -ls

# Exécuter une commande sur chaque résultat
find ~/Images -name "*.png" -exec chmod 644 {} \;
# {} est remplacé par chaque fichier trouvé
# \; termine la commande (obligatoire)

# Avec confirmation à chaque fichier
find . -name "*.tmp" -ok rm {} \;

# Plus efficace : + au lieu de \; — une seule invocation
find ~ -name "*.log" -exec du -h {} +

# Supprimer directement (raccourci pour -exec rm)
find /tmp -name "cache-*" -mtime +30 -delete
```

Dans `-exec`, les accolades `{}` représentent le fichier trouvé. La commande se termine par `\;` pour être lancée une fois par fichier, ou par `+` pour être lancée une seule fois avec tous les fichiers. La seconde forme est bien plus rapide.

Avant un `-delete`, lancez toujours la même commande sans lui. Vous verrez ce qui serait supprimé.

`locate`

`find` relit le disque à chaque fois. `locate` interroge un index, reconstruit chaque jour, et répond instantanément. En contrepartie, il ignore les fichiers créés depuis la dernière mise à jour.

```
sudo apt install plocate
locate sshd_config
sudo updatedb             # reconstruire l'index tout de suite
```

Où se trouve une commande ?

```
gary@vostro:~$ which python3
/usr/bin/python3
gary@vostro:~$ type ls
ls est un alias vers « ls --color=auto »
gary@vostro:~$ type cd
cd est une primitive du shell
```

`which` donne le chemin du programme qui sera lancé. `type` en dit davantage, car il connaît aussi les alias et les commandes internes du shell. Quand une commande ne se comporte pas comme prévu, commencez par `type`.

5.5 Connaître l'état de la machine

La charge : `uptime`

```
gary@vostro:~$ uptime
 20:36:54 up 1 day, 12:28,  1 user,  load average: 0,70, 0,55, 0,65
```

La machine tourne depuis un jour et demi, un utilisateur est connecté. Les trois derniers nombres sont la *charge moyenne* sur la dernière minute, les cinq et les quinze dernières minutes. La charge compte les processus qui utilisent le processeur ou qui attendent leur tour.

On la compare au nombre de cœurs, que donne la commande `nproc`. Sur une machine à quatre cœurs, une charge de 2 signifie qu'elle travaille à mi-régime. Une charge de 8 signifie que des processus font la queue. Les trois valeurs lues ensemble donnent la tendance : si la première dépasse nettement la troisième, la charge monte.

La mémoire : `free`

```
gary@vostro:~$ free -h
               total        utilisé      libre     partagé tamp/cache   disponible
Mem:            15Gi       5,2Gi       1,8Gi       245Mi       8,4Gi       9,2Gi
Échange:       4,0Gi          0B       4,0Gi
```

> **À RETENIR · Regardez « disponible », pas « libre »**
>
> Linux utilise la mémoire inoccupée pour garder en cache les fichiers lus récemment. C'est la colonne `tamp/cache`. Ce cache est rendu dès qu'un programme a besoin de place. Une colonne `libre` presque à zéro est donc normale, et même souhaitable. La mémoire réellement utilisable est celle de la colonne `disponible`.

La ligne `Échange` concerne le *swap*, un espace sur le disque qui prend le relais quand la mémoire manque. Un swap qui se remplit est le signe d'une machine à court de mémoire.

Les disques : `df`, `du`, `lsblk`

`df` donne l'espace libre de chaque système de fichiers :

```
gary@vostro:~$ df -h -x tmpfs -x squashfs
Sys. de fichiers Taille Utilisé Dispo Uti% Monté sur
/dev/nvme0n1p2     458G    142G  293G  33% /
/dev/nvme0n1p1     1,1G    6,2M  1,1G   1% /boot/efi
```

L'option `-x` écarte les systèmes de fichiers en mémoire et les snaps, qui encombrent l'affichage.

`du` mesure ce qu'occupe un dossier :

```
du -sh ~/Documents                          # le total, en une ligne
du -h --max-depth=1 ~ | sort -hr | head     # les plus gros sous-dossiers
```

`lsblk` montre les disques, leurs partitions et l'endroit où elles sont montées :

```
gary@vostro:~$ lsblk -e7
NAME                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
nvme0n1             259:0    0 476,9G  0 disk
├─nvme0n1p1         259:1    0     1G  0 part  /boot/efi
├─nvme0n1p2         259:2    0     2G  0 part  /boot
└─nvme0n1p3         259:3    0 473,9G  0 part
  └─dm_crypt-0      252:0    0 473,9G  0 crypt
    └─ubuntu--vg-ubuntu--lv 252:1 0 473,9G 0 lvm /
```

On reconnaît ici une installation chiffrée : une partition EFI, une partition `/boot`, puis une partition chiffrée qui contient un volume LVM. L'option `-e7` masque les périphériques des snaps. `lsblk -f` ajoute le type de système de fichiers et l'identifiant de chaque partition.

Le matériel

```
lscpu                # le processeur
lsmem                # la mémoire
lspci                # les cartes : graphique, réseau, son
lsusb                # les périphériques USB
sudo lshw -short     # tout le matériel, en résumé
```

Les processus

```
ps aux               # tous les processus
ps aux | grep nginx  # ceux dont la ligne contient « nginx »
top                  # vue en direct ; q pour quitter
htop                 # la même chose, en plus lisible
```

Dans `top`, `P` trie par utilisation du processeur et `M` par mémoire. Le chapitre 12 est consacré aux processus.

5.6 Redirections et tubes

Voici ce qui sépare l'utilisateur qui connaît des commandes de celui qui sait s'en servir. Les outils Unix sont petits et spécialisés. Leur puissance vient de ce qu'on peut les brancher les uns sur les autres.

Trois canaux

Chaque programme dispose de trois canaux, numérotés de 0 à 2.

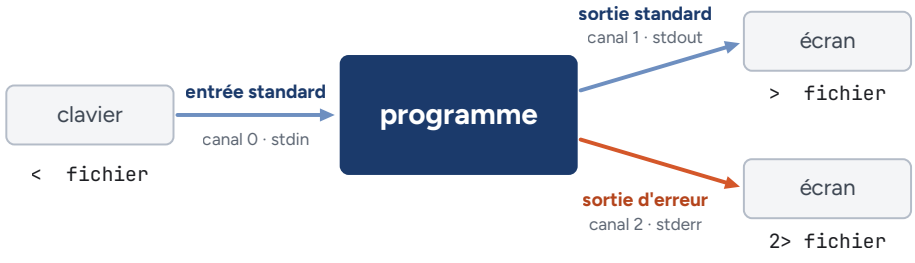


Figure 5.1 — Les trois canaux d'un programme. Chacun peut être redirigé séparément.

- **L'entrée standard** (0, *stdin*) : ce que le programme lit. Par défaut, le clavier.
- **La sortie standard** (1, *stdout*) : ses résultats. Par défaut, l'écran.
- **La sortie d'erreur** (2, *stderr*) : ses messages d'erreur. Par défaut, l'écran aussi.

Séparer les résultats des erreurs permet d'enregistrer les uns sans les autres.

Rediriger la sortie

```
# Rediriger stdout vers un fichier (écrase)
ls -lah > contenu.txt

# Rediriger en ajoutant (append)
date >> journal.log
echo "Nouvelle entrée" >> journal.log

# Rediriger stderr
ls /dossier-inexistant 2> erreurs.log

# Rediriger stdout et stderr dans des fichiers différents
./script.sh > sortie.log 2> erreurs.log

# Rediriger les deux au même endroit
./script.sh > tout.log 2>&1
# ou, raccourci moderne
./script.sh &> tout.log

# Rediriger vers /dev/null (jeter)
commande-verbeuse > /dev/null 2>&1
commande-verbeuse &> /dev/null
```

Écriture	Effet
`> fichier`	Envoie la sortie dans le fichier, qui est écrasé

Écriture	Effet
`>> fichier`	Ajoute la sortie à la fin du fichier
`2> fichier`	Envoie les erreurs dans le fichier
`> fichier 2>&1`	Envoie la sortie et les erreurs dans le même fichier
`&> fichier`	La même chose, en plus court
`< fichier`	Prend l'entrée dans le fichier

> **ATTENTION · Le signe > écrase sans prévenir**
>
> `ls > liste.txt` remplace le contenu de `liste.txt`, s'il existait. Dans le doute, utilisez `>>`.

Dans `2>&1`, lisez « le canal 2 va là où va le canal 1 ». L'ordre compte : il faut d'abord dire où va le canal 1. On écrit donc `commande > fichier 2>&1`, et non l'inverse.

Fournir une entrée

```
# Compter les lignes d'un fichier
wc -l < /etc/passwd
# identique à : wc -l /etc/passwd (mais la commande ne voit pas le nom du fichier)

# Envoyer le contenu d'un fichier comme entrée d'une commande
sort < fichier.txt

# Here-document (<<EOF) : entrée multi-lignes inline
cat <<EOF > config.txt
nom=gary
ville=douala
role=admin
EOF

# Here-string (<<<) : entrée d'une ligne
grep "gary" <<< "bonjour gary"
```

Le *here-document* (`<<EOF` … `EOF`) est précieux dans les scripts. Il permet d'écrire un fichier de plusieurs lignes sans ouvrir d'éditeur.

Les tubes

Le tube, noté `|`, relie la sortie d'une commande à l'entrée de la suivante.

```
# Lister les utilisateurs, trier alphabétiquement
cut -d: -f1 /etc/passwd | sort

# Combien de processus firefox ?
ps aux | grep firefox | wc -l

# Les 10 plus gros fichiers de /var
sudo du -h /var 2>/dev/null | sort -hr | head -10

# Avec xargs : passer la sortie en arguments à une commande
find . -name "*.log" | xargs du -sh
# mieux :
find . -name "*.log" -print0 | xargs -0 du -sh
# (-print0 et -0 gèrent les espaces dans les noms)
```

Prenons un exemple complet. Quels sont les shells utilisés sur cette machine, et par combien de comptes ? Dans `/etc/passwd`, le shell est le septième champ de chaque ligne.

```
gary@vostro:~$ cut -d: -f7 /etc/passwd | sort | uniq -c | sort -rn
     39 /usr/sbin/nologin
      7 /bin/false
      3 /bin/bash
      1 /bin/sync
```

Quatre petits outils, chacun avec un rôle simple. `cut` extrait le septième champ, `sort` range les lignes identiques côte à côte, `uniq -c` les compte, et le second `sort` classe par nombre décroissant. On construit une telle ligne un morceau à la fois, en regardant le résultat à chaque étape. Le chapitre 9 présente ces outils en détail.

`tee` et le piège de `sudo`

`tee` écrit ce qu'il reçoit dans un fichier, tout en le laissant passer.

```
# Voir la commande à l'écran ET l'enregistrer
ls -l | tee listing.txt

# Ajouter (au lieu d'écraser) avec -a
date | tee -a journal.log

# Cas classique : écrire dans un fichier système avec sudo
# CELA NE MARCHE PAS (sudo ne s'applique pas à la redirection) :
sudo echo "option=valeur" > /etc/mon-fichier
```

```
# ÇA MARCHE :
echo "option=valeur" | sudo tee /etc/mon-fichier
echo "option=valeur" | sudo tee -a /etc/mon-fichier  # ajouter
```

DANGER · sudo ne s'applique pas à la redirection

Dans `sudo echo texte > /etc/fichier`, seul `echo` s'exécute avec les droits de `root`. La redirection, elle, est faite par votre shell, avec vos droits, et elle échoue avec « Permission non accordée ». La solution habituelle est `echo texte | sudo tee /etc/fichier`.

5.7 Les motifs de noms de fichiers

Avant de lancer une commande, le shell remplace certains caractères par la liste des fichiers correspondants. Quand vous tapez `ls *.txt`, la commande `ls` ne voit jamais l'étoile. Elle reçoit directement `notes.txt rapport.txt`.

Motif	Correspond à	Exemple
`*`	N'importe quelle suite de caractères	`*.pdf`
`?`	Un caractère, exactement	`photo-0?.jpg`
`[abc]`	Un caractère parmi ceux de la liste	`rapport-[12].odt`
`[a-z]`, `[0-9]`	Un caractère dans l'intervalle	`chap[0-9].md`
`[!abc]`	Un caractère qui n'est pas dans la liste	`[!.]*`
`**`	Tous les sous-dossiers, si `globstar` est activé	`**/*.py`

L'étoile ignore les fichiers cachés : `*.txt` ne trouve pas `.notes.txt`.

Les accolades fonctionnent autrement. Elles ne cherchent pas de fichiers existants, elles fabriquent des mots :

```
gary@vostro:~$ echo chap{1..3}.md
chap1.md chap2.md chap3.md
gary@vostro:~$ echo rapport.{pdf,odt}
rapport.pdf rapport.odt
```

```
# Lister tous les PDF du home
ls ~/*.pdf

# Supprimer tous les .tmp dans le dossier courant
```

```
rm *.tmp

# Copier tous les .jpg et .png vers un dossier
cp *.{jpg,png} ~/Images/

# Créer une série de fichiers
touch chap{1..10}.md
# crée chap1.md, chap2.md, ..., chap10.md

# Fichiers nommés jour-01 à jour-31
touch jour-{01..31}.md

# Renommer par lot avec un script simple
for f in *.jpeg; do mv "$f" "${f%.jpeg}.jpg"; done

# Activer le glob récursif **
shopt -s globstar
ls **/*.py              # tous les .py à tous les niveaux

# Pour inclure les fichiers cachés dans *
shopt -s dotglob
ls *                    # inclut .bashrc, .profile, etc.
```

ASTUCE · Voir ce que le shell va faire

En cas de doute sur un motif, mettez `echo` devant la commande : `echo rm *.tmp` affiche la ligne que le shell exécuterait, sans rien supprimer.

NOTE · Ce ne sont pas des expressions régulières

Les motifs du shell servent à désigner des noms de fichiers. Les expressions régulières, utilisées par `grep` et `sed`, servent à décrire du texte, et l'étoile n'y a pas le même sens. Le chapitre 9 fait le point.

5.8 L'historique et l'édition de la ligne

```
# Voir tout l'historique
history

# Les 20 dernières
history 20

# Rejouer la dernière commande
!!

# Rejouer la N-ième commande de l'historique
```

```
!42

# Rejouer la dernière qui commence par 'apt'
!apt

# Rejouer la dernière qui contient 'nginx'
!?nginx

# Juste afficher ce qui serait exécuté (:p)
!42:p

# La dernière commande, avec sudo devant
sudo !!

# Le dernier argument de la commande précédente
!$
# Exemple :
mkdir ~/projets/site
cd !$            # équivalent à cd ~/projets/site

# Tous les arguments de la commande précédente
!*

# Effacer l'historique
history -c
```

De toutes ces écritures, deux servent chaque jour : `sudo !!`, pour relancer avec `sudo` la commande qui vient d'être refusée, et `!$`, qui reprend le dernier argument de la commande précédente.

Pour retrouver une commande ancienne, Ctrl+R reste le moyen le plus rapide. Tapez quelques lettres, et la dernière commande qui les contient apparaît. Ctrl+R de nouveau remonte à la précédente, Entrée exécute, → permet de la modifier avant.

Raccourci	Effet
Ctrl+A, Ctrl+E	Début, fin de la ligne
Alt+B, Alt+F	Reculer, avancer d'un mot
Ctrl+U, Ctrl+K	Effacer avant, après le curseur
Ctrl+W	Effacer le mot précédent
Ctrl+Y	Recoller ce qui vient d'être effacé
Ctrl+Z	Suspendre le programme en cours ; `fg` le reprend

5.9 Alias et variables d'environnement

Ces deux notions sont approfondies au chapitre 8. En voici l'essentiel.

Les alias

Un alias donne un nom court à une commande longue.

```
gary@vostro:~$ alias
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
gary@vostro:~$ alias maj='sudo apt update && sudo apt upgrade'
```

Un alias créé ainsi disparaît à la fermeture du terminal. Pour le conserver, ajoutez la ligne à la fin du fichier `~/.bashrc`, puis rechargez-le avec `source ~/.bashrc`.

Les variables d'environnement

Une variable d'environnement est une valeur nommée, que le shell transmet aux programmes qu'il lance.

```
gary@vostro:~$ echo $HOME
/home/gary
gary@vostro:~$ echo $LANG
fr_FR.UTF-8
gary@vostro:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
usr/local/games:/snap/bin
```

`PATH` est la plus importante. Elle contient la liste des dossiers dans lesquels le shell cherche les commandes, dans l'ordre. Un programme placé ailleurs n'est pas trouvé, à moins de donner son chemin : c'est pourquoi on lance un script du dossier courant avec `./script.sh`.

```
env                          # toutes les variables
export EDITOR=nano           # définir une variable pour cette session
LANG=C ls -l                 # la définir pour une seule commande
```

> **ASTUCE · ~/bin et ~/.local/bin**
>
> Sur Ubuntu, si vous créez un dossier `~/bin` ou `~/.local/bin`, il est ajouté à `PATH` à l'ouverture de session suivante. C'est l'endroit où ranger vos scripts personnels.

5.10 Les archives

Dans le monde Unix, on distingue deux opérations. *Archiver*, c'est réunir plusieurs fichiers en un seul. *Compresser*, c'est réduire la taille d'un fichier. `tar` archive, `gzip`, `xz` et `zstd` compressent, et `tar` sait faire appel à eux.

Créer une archive

```
# Créer une archive tar simple (non compressée)
tar -cf archive.tar dossier1 fichier1 dossier2

# Archive compressée gzip (.tar.gz)
tar -czf projet.tar.gz projet/

# Archive compressée bzip2 (.tar.bz2)
tar -cjf projet.tar.bz2 projet/

# Archive compressée xz (.tar.xz)
tar -cJf projet.tar.xz projet/

# Options communes :
#   -c (create)     : créer
#   -v (verbose)    : afficher les fichiers traités
#   -f (file)       : spécifie le nom du fichier
#   -z (gzip)
#   -j (bzip2)
#   -J (xz)

# En affichant les fichiers traités
tar -czvf backup.tar.gz ~/Documents ~/Images

# Exclure un sous-dossier
tar -czf backup.tar.gz --exclude='node_modules' --exclude='.git' projet/
```

Les lettres se lisent ainsi : `c` pour créer, `z` pour compresser avec gzip, `v` pour afficher les fichiers, `f` pour indiquer le nom de l'archive. Le `f` vient toujours en dernier, juste avant ce nom.

Lister et extraire

```
# Extraire une archive (tar détecte le format)
tar -xf archive.tar.gz

# En affichant les fichiers extraits
tar -xvzf archive.tar.gz

# Extraire dans un dossier précis
```

```
tar -xzf archive.tar.gz -C /tmp/destination/

# Les options :
#   -x (extract)    : extraire
#   -t (list)       : lister sans extraire
#   -C              : changer de dossier avant d'extraire

# Lister le contenu d'une archive SANS extraire
tar -tzvf archive.tar.gz
tar -tzvf archive.tar.gz | head

# Extraire un seul fichier de l'archive
tar -xzf archive.tar.gz chemin/vers/fichier.txt
```

Regardez toujours le contenu d'une archive avec `-t` avant de l'extraire. Certaines déposent leurs fichiers directement dans le dossier courant, au lieu de créer un sous-dossier.

Compresser un fichier seul

```
# Compresser (crée gros.log.gz et supprime gros.log)
gzip gros.log

# Décompresser
gunzip gros.log.gz
# ou
gzip -d gros.log.gz

# Garder l'original
gzip -k gros.log

# Compresser en forçant un niveau de compression (1 = rapide, 9 = max)
gzip -9 gros.log

# bzip2 — meilleur taux de compression, plus lent
bzip2 gros.log         # → gros.log.bz2
bunzip2 gros.log.bz2

# xz — encore meilleur ratio
xz fichier.tar         # → fichier.tar.xz
unxz fichier.tar.xz

# zstd — moderne, excellent compromis
zstd fichier           # → fichier.zst
unzstd fichier.zst
```

Le format zip

Pour échanger avec des utilisateurs de Windows ou de macOS, le format zip reste le plus sûr.

```
# Installer zip et unzip si absents
sudo apt install zip unzip

# Créer un zip (récursif -r pour inclure les dossiers)
zip -r projet.zip projet/

# Avec exclusion
zip -r backup.zip dossier/ -x "*.log" "node_modules/*"

# Lister le contenu sans extraire
unzip -l archive.zip

# Extraire
unzip archive.zip

# Extraire dans un dossier précis
unzip archive.zip -d /tmp/dest/

# Protéger par mot de passe
zip -r -e sensible.zip dossier/
# (demandera le mot de passe)
```

Format	Taux de compression	Vitesse	Où on le rencontre
`.tar.gz`	Bon	Rapide	Partout ; le choix par défaut
`.tar.bz2`	Meilleur	Lent	Archives anciennes
`.tar.xz`	Excellent	Très lent à la compression	Sources de logiciels, noyau
`.tar.zst`	Très bon	Très rapide	Systèmes récents, sauvegardes
`.zip`	Moyen	Rapide	Échanges avec Windows et macOS

En résumé

- `mkdir -p`, `cp -a`, `mv`, `rm -r` : les quatre gestes de base. `rm` ne connaît pas la corbeille.
- `less` pour lire, `tail -f` pour suivre un journal, `zcat` et `zgrep` pour les fichiers compressés.
- `find` cherche selon le nom, la taille, la date, le propriétaire, et peut agir sur ce qu'il trouve.
- `uptime`, `free -h`, `df -h`, `du -sh`, `lsblk` : cinq commandes pour savoir comment va la machine.
- `>` écrase, `>>` ajoute, `2>` capte les erreurs, `|` relie deux commandes.
- `sudo` ne porte pas sur une redirection. On passe par `sudo tee`.
- `*`, `?` et `[…]` désignent des fichiers existants. `{…}` fabrique des mots.
- `tar -czf` crée une archive compressée, `tar -tf` la liste, `tar -xf` l'extrait.

Atelier — Remettre de l'ordre

Objectif : ranger, nettoyer puis archiver un dossier en désordre, en combinant les commandes du chapitre. Comptez quarante minutes.

Créer le désordre

Créez un dossier `~/atelier5`, placez-vous dedans et lancez :

```
# Dans ~/atelier-chap5
touch rapport.{pdf,docx,odt}
touch photo-{01..15}.jpg
touch notes-{01..08}.txt
touch script-{deploy,backup,cleanup}.sh
touch {janvier,fevrier,mars,avril}.md
touch temp-{01..20}.tmp
echo "Contenu" > README.md
echo "Log 2024-01-15" > access.log
echo "Log 2024-02-10" > error.log
mkdir -p vieux-trucs/{sous-1,sous-2}/{a,b}
touch vieux-trucs/{old1,old2,old3}.bak
touch vieux-trucs/sous-1/a/fichier-{01..05}.dat
```

Observer

1. Combien y a-t-il de fichiers en tout, sous-dossiers compris ? (`find` et `wc`)
2. Affichez l'arborescence avec `tree` .

Ranger

3. Créez les dossiers `docs` , `images` , `scripts` et `journaux` .
4. Avec des motifs, déplacez chaque fichier dans le bon dossier : les `.pdf` , `.docx` , `.odt` , `.md` et `.txt` dans `docs` , les `.jpg` dans `images` , les `.sh` dans `scripts` , les `.log` dans `journaux` .
5. Vérifiez avec `tree -L 2` .

Nettoyer

6. Affichez la liste des fichiers `.tmp` avec `ls` , puis supprimez-les en rappelant la ligne.
7. Supprimez le dossier `vieux-trucs` et tout ce qu'il contient.

Archiver

8. Remontez d'un niveau et créez `atelier5.tar.gz` .
9. Listez le contenu de l'archive sans l'extraire.
10. Extrayez-la dans `/tmp` et comparez avec l'original : `diff -r ~/atelier5 /tmp/atelier5` ne doit rien afficher.

En une ligne

11. Combien de fichiers `.jpg` contient `images` ?
12. Quel est le fichier modifié le plus récemment dans tout le dossier ?

Exercices

Exercice 1 — Une ligne, pas plus

1. Lister votre dossier personnel, fichiers cachés compris, avec des tailles lisibles, du plus récent au plus ancien.
2. Créer d'un coup `~/projets/{alpha,beta}/{src,tests,docs}` .
3. Trouver dans `/var/log` les fichiers `.log` de plus de 10 Mo.
4. Compter les lignes de `/var/log/syslog` qui contiennent `error` , sans tenir compte de la casse.

5. Archiver `~/Documents` au format `.tar.gz` en excluant les dossiers `node_modules`.

Exercice 2 — Que font ces lignes ?

```
# (a)
cut -d: -f1 /etc/passwd | sort | uniq | wc -l

# (b)
find ~ -type f -name "*.txt" -mtime -1 | head -5

# (c)
du -h /home/* 2>/dev/null | sort -hr | head -3

# (d)
history | awk '{print $2}' | sort | uniq -c | sort -rn | head -10
```

Exercice 3 — Vrai ou faux ?

1. `mv a.txt b.txt` écrase `b.txt` sans demander confirmation.
2. `>` ajoute à la fin d'un fichier existant.
3. `*.txt` trouve le fichier `.notes.txt`.
4. `sudo echo ok > /etc/essai` écrit « ok » dans `/etc/essai`.
5. `tar -xf archive.tar.gz` reconnaît seul que l'archive est compressée avec gzip.
6. `find . -size -1k` trouve les fichiers de moins d'un kilooctet.

Exercice 4 — Un alias durable

Créez un alias `maj` qui met le système à jour. Rendez-le permanent, rechargez votre configuration et vérifiez-le avec `type maj`.

Exercice 5 — Le disque est plein

Un serveur signale que son disque est plein. Vous y êtes connecté en SSH. Donnez les commandes qui permettent de :

1. voir le taux d'occupation de chaque système de fichiers ;
2. trouver les dix plus gros dossiers de `/var` ;
3. trouver les fichiers de plus de 500 Mo sur tout le système ;
4. vérifier qu'il ne s'agit pas d'un manque d'inodes.

Pour aller plus loin

Documentation. `gnu.org/software/coreutils/manual/` décrit en détail `ls`, `cp`, `mv`, `rm`, `sort` et tous les utilitaires de base. `explainshell.com` décompose n'importe quelle ligne de commande.

Des outils plus récents. Ils ne remplacent pas les commandes classiques, qu'il faut connaître parce qu'elles sont partout, mais ils rendent le travail interactif plus agréable. Tous sont dans les dépôts d'Ubuntu : `eza` (un `ls` en couleurs), `bat` (un `cat` avec coloration, lancé par `batcat`), `fd-find` (un `find` plus simple, lancé par `fdfind`), `ripgrep` (un `grep` très rapide, lancé par `rg`), `duf` (un `df` lisible), `ncdu` (un `du` interactif), `fzf` (une recherche floue dans l'historique et les fichiers).

S'entraîner. `overthewire.org/wargames/bandit` : un jeu en ligne où chaque niveau se résout avec les commandes de ce chapitre. C'est le meilleur exercice qui soit.

Lecture. *Linux Pocket Guide*, de Daniel Barrett (O'Reilly) : un aide-mémoire dense et bien fait.



CHAPITRE 6

Utilisateurs, groupes et permissions

Qui a le droit de faire quoi, sur quel fichier. Un modèle vieux de cinquante ans, simple, et sur lequel repose toute la sécurité du système.

DANS CE CHAPITRE

- Comprendre comment Linux identifie les utilisateurs et les groupes.
- Lire `/etc/passwd`, `/etc/shadow` et `/etc/group`.
- Créer, modifier, verrouiller et supprimer des comptes et des groupes.
- Lire les droits affichés par `ls -l` et les modifier avec `chmod`, `chown` et `chgrp`.
- Utiliser les droits spéciaux : SUID, SGID et sticky bit.
- Régler les droits par défaut avec `umask` et les affiner avec les ACL.
- Déléguer des droits d'administration avec `sudo`.

> **AVANT DE COMMENCER**
>
> Les chapitres 4 et 5. Les exemples sont exécutés sur Ubuntu Server, installé en anglais : les messages du système y sont donc en anglais. Faites les manipulations dans une machine virtuelle. On y crée et supprime des comptes, ce que l'on préfère éviter sur sa machine de tous les jours.

6.1 Utilisateurs et groupes

Des numéros, pas des noms

Le noyau ne connaît pas les noms d'utilisateurs. Il ne manipule que des numéros : un **UID** (*user identifier*) pour chaque utilisateur, un **GID** (*group identifier*) pour chaque groupe. Le nom `gary` n'est qu'une étiquette associée au numéro 1000, pour le confort des humains.

```
gary@serveur:~$ id
uid=1000(gary) gid=1000(gary) groups=1000(gary),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),100(users)
```

La commande `id` affiche votre identité telle que le système la voit : votre numéro, votre groupe principal, et tous les groupes dont vous faites partie.

Les numéros suivent une convention :

UID	Qui	Exemples
0	L'administrateur	`root`
1 à 999	Les comptes système, sous lesquels tournent les services	`www-data`, `postgres`, `sshd`
1000 et au-delà	Les personnes	`gary` (1000), `alice` (1001)
65534	Un compte sans aucun droit	`nobody`

Les comptes système ne servent pas à ouvrir une session. Ils existent pour que chaque service tourne sous une identité aux droits limités. Si le serveur web est compromis, l'attaquant se retrouve `www-data`, et non `root`.

Groupe principal et groupes secondaires

Chaque utilisateur a un **groupe principal** et, s'il le faut, des **groupes secondaires**.

Sur Ubuntu, le groupe principal porte le même nom que l'utilisateur et ne contient que lui. Les fichiers que vous créez lui appartiennent.

Les groupes secondaires donnent accès à des ressources partagées. Le premier compte créé à l'installation appartient à plusieurs d'entre eux :

Groupe	Ce qu'il permet
`sudo`	Utiliser la commande `sudo`, donc administrer la machine
`adm`	Lire les journaux de `/var/log`
`plugdev`, `cdrom`	Utiliser les supports amovibles
`users`	Groupe commun à tous les comptes de personnes

6.2 Les trois fichiers des comptes

Tous les comptes de la machine sont décrits dans trois fichiers texte.

`/etc/passwd`

Une ligne par compte, sept champs séparés par des deux-points. Tout le monde peut le lire.

```
gary@serveur:~$ getent passwd alice
alice:x:1001:1001:Alice Nkomo,,,:/home/alice:/bin/bash
```

Champ	Valeur	Signification
1	`alice`	Nom du compte
2	`x`	Indique que le mot de passe se trouve dans `/etc/shadow`
3	`1001`	UID
4	`1001`	GID du groupe principal
5	`Alice Nkomo,,,`	Nom complet et coordonnées facultatives
6	`/home/alice`	Dossier personnel
7	`/bin/bash`	Shell lancé à la connexion

Pour un compte système, le septième champ vaut souvent `/usr/sbin/nologin`. Ce programme refuse poliment toute ouverture de session.

> **REPÈRE HISTORIQUE · Pourquoi ce « x » ?**
>
> Dans les premiers Unix, `/etc/passwd` contenait les mots de passe chiffrés. Or ce fichier doit rester lisible par tous, puisque c'est lui qui permet à `ls` d'afficher des noms plutôt que des numéros. N'importe quel utilisateur pouvait donc copier les mots de passe chiffrés et tenter de les casser à loisir. À la fin des années 1980, ils ont été déplacés dans un fichier réservé à `root`, `/etc/shadow`. Le `x` est resté, pour marquer l'emplacement.

`/etc/shadow`

```
gary@serveur:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 679 Oct  4 19:41 /etc/shadow
gary@serveur:~$ sudo grep alice /etc/shadow
alice:$y$j9T$HJibgqTPj7NmFIoqhObpF1$aGja…:20730:0:99999:7:::
```

Le deuxième champ est l'*empreinte* du mot de passe : le résultat d'un calcul à sens unique, d'où l'on ne peut pas remonter au mot de passe. Son début indique l'algorithme employé. `$y$` désigne yescrypt, utilisé par Ubuntu 24.04. `$6$` désigne SHA-512, que l'on rencontre sur les systèmes plus anciens.

Si ce champ commence par `!` ou vaut `*`, aucun mot de passe ne permet d'ouvrir une session avec ce compte. C'est le cas de `root` sur Ubuntu.

Les champs suivants règlent la durée de vie du mot de passe : date du dernier changement, comptée en jours depuis le 1er janvier 1970, puis délai minimal et maximal entre deux changements, et délai d'avertissement.

`/etc/group`

```
gary@serveur:~$ getent group sudo
sudo:x:27:gary
```

Quatre champs : le nom du groupe, un `x` hérité de l'histoire, le GID, et la liste des membres. Le groupe principal d'un utilisateur n'y figure pas. Il est indiqué dans `/etc/passwd`.

> **ATTENTION · On ne modifie pas ces fichiers à la main**
>
> Une ligne mal formée dans `/etc/passwd` peut interdire toute connexion, y compris celle de `root`. Passez par les commandes présentées dans la section suivante. Elles vérifient ce qu'elles écrivent.

6.3 Gérer les comptes

Créer un compte

Ubuntu propose deux commandes. `adduser` pose des questions et s'occupe de tout : groupe, dossier personnel, mot de passe. `useradd` est l'outil de base, commun à toutes les distributions. Il ne fait que ce qu'on lui demande par des options. Au clavier, on préfère `adduser`.

```
gary@serveur:~$ sudo adduser alice
info: Adding user `alice' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `alice' (1001) ...
info: Adding new user `alice' (1001) with group `alice (1001)' ...
info: Creating home directory `/home/alice' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for alice
Enter the new value, or press ENTER for the default
        Full Name []: Alice Nkomo
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
info: Adding new user `alice' to supplemental / extra groups `users' ...
info: Adding user `alice' to group `users' ...
```

Le dossier personnel est créé à partir de `/etc/skel`, qui contient les fichiers remis à tout nouveau compte, comme `.bashrc` et `.profile`.

Dans un script, où personne n'est là pour répondre aux questions, on utilise `useradd` :

```
sudo useradd -m -s /bin/bash -c "Bob Tamba" bob     # -m crée le dossier personnel
sudo passwd bob                                     # le mot de passe se définit à part
```

Mots de passe

```
passwd                    # changer son propre mot de passe
sudo passwd alice         # définir celui d'alice
sudo passwd -e alice      # l'obliger à en changer à sa prochaine connexion
sudo passwd -l alice      # verrouiller le compte
sudo passwd -u alice      # le déverrouiller
```

```
gary@serveur:~$ sudo passwd -S alice
alice P 2026-10-04 0 99999 7 -1
```

La deuxième colonne donne l'état du mot de passe : `P` s'il est utilisable, `L` s'il est verrouillé, `NP` s'il n'y en a pas.

> **NOTE · Verrouiller le mot de passe ne ferme pas toutes les portes**
>
> `passwd -l` empêche la connexion par mot de passe. Une clé SSH déjà installée continue de fonctionner. Pour fermer complètement un compte, faites-le aussi expirer : `sudo usermod --expiredate 1 alice`.

Modifier un compte

```
sudo usermod -aG sudo alice             # ajouter alice au groupe sudo
sudo usermod -s /usr/bin/zsh alice      # changer son shell
sudo usermod -c "Alice Nkomo-Bell" alice
sudo usermod -L alice                   # verrouiller ; -U pour déverrouiller
```

> **DANGER · usermod -G sans le « a »**
>
> L'option `-G` *remplace* la liste des groupes secondaires. Avec `-a` (*append*), elle y *ajoute* un groupe. La différence coûte cher :
>
> ```
> gary@serveur:~$ id -nG alice
> alice sudo users equipe
> gary@serveur:~$ sudo usermod -G equipe alice
> gary@serveur:~$ id -nG alice
> alice equipe
> ```
>
> Alice vient de perdre le groupe `sudo`. Faites cela sur votre propre compte, et vous ne pouvez plus administrer la machine. Écrivez toujours `-aG`.

Supprimer un compte

```
sudo deluser alice                    # supprime le compte, garde ses fichiers
sudo deluser --remove-home alice      # supprime aussi son dossier personnel
```

Avant de supprimer le compte d'une personne qui quitte l'équipe, verrouillez-le quelques semaines. On découvre souvent après coup qu'un script ou une tâche planifiée tournait sous son nom.

Se renseigner

```
# L'ensemble des infos identité
id alice

# Juste les groupes
groups alice

# La ligne /etc/passwd
getent passwd alice

# Statut mot de passe / expiration
sudo chage -l alice
Last password change                                    : Oct 04, 2026
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change : 0
Maximum number of days between password change : 99999
...

# Qui est actuellement connecté
who
w

# Dernières connexions
last
last alice          # filtrer
last -n 10          # les 10 dernières

# Tentatives de connexion échouées
sudo lastb
```

6.4 Gérer les groupes

```
sudo groupadd equipe                # créer un groupe
sudo gpasswd -a alice equipe        # y ajouter alice
sudo gpasswd -d alice equipe        # l'en retirer
getent group equipe                 # voir ses membres
sudo groupdel equipe                # le supprimer
```

`sudo gpasswd -a alice equipe` et `sudo usermod -aG equipe alice` font la même chose.

> **À RETENIR · Un nouveau groupe ne prend effet qu'à la connexion suivante**
>
> Les groupes d'un utilisateur sont lus à l'ouverture de sa session. Si vous vous ajoutez à un groupe, vos terminaux ouverts n'en savent rien. Fermez la session et rouvrez-la. C'est l'explication habituelle du « je suis dans le groupe `docker` , et pourtant la commande est refusée ».

6.5 Les permissions

Trois droits pour trois catégories

Chaque fichier a un utilisateur propriétaire et un groupe propriétaire. Ses droits sont définis pour trois catégories de personnes :

- **u**, le propriétaire (*user*) ;
- **g**, les membres du groupe (*group*) ;
- **o**, tous les autres (*others*).

Et pour chaque catégorie, trois droits : **r** pour lire (*read*), **w** pour écrire (*write*), **x** pour exécuter (*execute*).



Figure 6.1 — Lire une ligne de ls -l.

```
gary@serveur:~$ ls -l
-rw-r--r-- 1 gary gary  1234 Apr 23 14:00 rapport.pdf
-rwxr-xr-x 1 gary gary   450 Apr 22 09:00 sauvegarde.sh
-rw------- 1 gary gary   654 Apr 22 11:00 secret.txt
drwxr-xr-x 3 gary gary  4096 Apr 10 14:00 projet
```

`rapport.pdf` peut être lu par tout le monde et modifié par son propriétaire. `sauvegarde.sh` est en plus exécutable. `secret.txt` n'est accessible qu'à `gary`.

Fichiers et dossiers : le même mot, deux sens

Droit	Sur un fichier	Sur un dossier
`r`	Lire le contenu	Lister les noms qu'il contient
`w`	Modifier le contenu	Créer, supprimer, renommer des fichiers dedans
`x`	Le lancer comme un programme	Y entrer, et atteindre ce qu'il contient

Deux conséquences surprennent toujours.

Sans `x` sur un dossier, on n'atteint rien de ce qu'il contient, même si les fichiers eux-mêmes sont ouverts à tous. Pour lire `/srv/site/index.html`, il faut le droit `x` sur `/`, sur `/srv` et sur `/srv/site`.

Le droit de supprimer un fichier dépend du dossier, pas du fichier. Supprimer, c'est retirer un nom d'un dossier (chapitre 4). Celui qui a le droit `w` sur un dossier peut y supprimer un fichier qu'il n'a même pas le droit de lire.

Comment le système décide

Quand un programme veut accéder à un fichier, le noyau applique une seule des trois séries de droits, la première qui correspond :

1. Si l'utilisateur est le propriétaire, ce sont les droits du propriétaire qui s'appliquent.
2. Sinon, s'il est membre du groupe, ce sont les droits du groupe.
3. Sinon, ce sont les droits des autres.

Les droits ne se cumulent pas. Avec `-r--rw-rw-`, le propriétaire ne peut pas écrire dans son propre fichier, alors que tous les autres le peuvent. Il lui reste, il est vrai, le droit de changer les permissions.

`root` échappe à ces vérifications : il peut lire et modifier tous les fichiers.

6.6 La notation octale et la notation symbolique

En chiffres

On attribue une valeur à chaque droit : 4 pour `r`, 2 pour `w`, 1 pour `x`. On additionne pour chaque catégorie, ce qui donne trois chiffres.

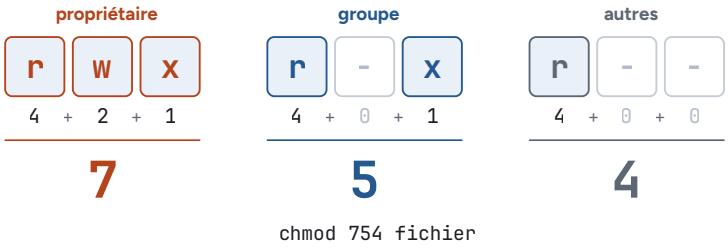


Figure 6.2 — De la notation en lettres à la notation en chiffres.

Valeur	Droits	Usage courant
644	`rw-r--r--`	Fichier ordinaire, lisible par tous
640	`rw-r-----`	Fichier réservé au propriétaire et à son groupe
600	`rw-------`	Fichier privé : clé SSH, mot de passe
755	`rwxr-xr-x`	Programme, script, dossier ouvert à tous
750	`rwxr-x---`	Dossier réservé au propriétaire et à son groupe
700	`rwx------`	Dossier privé
775, 664	`rwxrwxr-x`, `rw-rw-r--`	Dossier et fichier partagés au sein d'un groupe

En lettres

La notation symbolique modifie un droit sans toucher aux autres. Elle se compose de trois parties : à qui (`u`, `g`, `o`, ou `a` pour tout le monde), quoi faire (`+` ajouter, `-` retirer, `=` fixer), et quel droit.

```
# Ajouter x pour le user
chmod u+x script.sh

# Retirer w pour group et other
chmod go-w fichier.txt

# Définir exactement r-- pour other
chmod o=r document.pdf

# Ajouter x à tous
chmod a+x script.sh

# Combinaisons séparées par virgule
chmod u=rwx,g=rx,o= secret.sh    # o= sans rien après = aucun droit
```

La notation en chiffres fixe tous les droits d'un coup. La notation en lettres en retouche un. On utilise les deux.

> **DANGER · chmod 777 n'est jamais la solution**
>
> Quand « ça ne marche pas », la tentation est de donner tous les droits à tout le monde. Avec `777`, n'importe quel compte de la machine peut modifier le fichier, y compris un service compromis. Cherchez plutôt qui a besoin d'accéder au fichier, puis donnez-lui ce droit par le propriétaire ou par le groupe.

6.7 chmod, chown, chgrp

Modifier les droits : `chmod`

```
# Octal
chmod 755 script.sh
chmod 644 document.txt
chmod 600 ~/.ssh/id_ed25519     # clé SSH privée, obligatoire

# Symbolique
chmod u+x script.sh
chmod go-rwx sensible.txt        # enlève tous droits à group et other
chmod a=r readonly.txt

# Récursif sur un dossier
chmod -R 755 projet/

# Avec find pour distinguer fichiers et dossiers
find projet/ -type d -exec chmod 755 {} \;
find projet/ -type f -exec chmod 644 {} \;

# Préserver le x sur les dossiers mais pas sur les fichiers (X majuscule)
chmod -R u+rwX,go+rX projet/
# X = x seulement si c'est un dossier OU si quelqu'un a déjà x
```

La dernière forme mérite une explication. Appliquer `755` à toute une arborescence rendrait tous les fichiers exécutables, ce qu'on ne veut pas. Le `X` majuscule n'ajoute le droit d'exécution qu'aux dossiers, et aux fichiers qui l'ont déjà.

Changer de propriétaire : `chown` et `chgrp`

```
# Changer le propriétaire (root seul)
sudo chown alice fichier.txt

# Changer propriétaire et groupe
```

```
sudo chown alice:developers fichier.txt

# Juste le groupe (avec chown)
sudo chown :developers fichier.txt

# Récursif
sudo chown -R www-data:www-data /var/www/mon-site

# Copier l'owner d'un autre fichier
sudo chown --reference=modele.txt cible.txt

# Cas classique : reprendre la main sur un fichier laissé par root
sudo chown $USER:$USER /tmp/fichier-abandonne
```

Seul `root` peut donner un fichier à un autre utilisateur. Les fichiers d'un site web appartiennent en général à `www-data`, le compte du serveur web.

`chgrp groupe fichier` change le groupe seul. Il équivaut à `chown :groupe fichier`.

6.8 Les droits spéciaux

Trois droits supplémentaires modifient le comportement des programmes et des dossiers.

SUID : emprunter l'identité du propriétaire

Pour changer votre mot de passe, il faut écrire dans `/etc/shadow`, qui n'est accessible qu'à `root`. Pourtant, vous y arrivez sans `sudo`. L'explication est dans les droits de la commande `passwd` :

```
gary@serveur:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 64152 May 30  2024 /usr/bin/passwd
```

Le `s` à la place du `x` du propriétaire est le droit **SUID**. Un programme qui le porte s'exécute avec l'identité de son propriétaire, ici `root`, quel que soit l'utilisateur qui le lance. `passwd` est écrit pour ne modifier que votre propre ligne.

On le pose avec `chmod u+s`, ou par un 4 devant les trois chiffres : `chmod 4755`.

ATTENTION · Un programme SUID est une porte vers root

S'il contient une faille, n'importe quel utilisateur peut s'en servir pour devenir administrateur. Les distributions en limitent donc le nombre, et tout audit de sécurité commence par les recenser :

```
sudo find / -xdev -perm -4000 -type f
```

Sur une installation neuve, la liste contient une dizaine de commandes : `passwd`, `sudo`, `su`, `mount`… Tout ce qui s'y ajoute doit pouvoir s'expliquer. Notez aussi que Linux ignore le SUID sur les scripts.

SGID : hériter du groupe du dossier

Sur un dossier, le droit **SGID** est très utile. Les fichiers créés dedans reçoivent le groupe *du dossier*, et non le groupe principal de celui qui les crée. C'est la base d'un dossier partagé.

```
gary@serveur:~$ sudo mkdir /srv/projet
gary@serveur:~$ sudo chown root:equipe /srv/projet
gary@serveur:~$ sudo chmod 2770 /srv/projet
gary@serveur:~$ ls -ld /srv/projet
drwxrws--- 2 root equipe 4096 Oct  4 19:41 /srv/projet
```

Le 2 placé devant `770` pose le SGID, visible au `s` dans les droits du groupe. Alice, membre du groupe `equipe`, y crée un fichier :

```
alice@serveur:/srv/projet$ touch notes.txt
alice@serveur:/srv/projet$ ls -l notes.txt
-rw-rw-r-- 1 alice equipe 0 Oct  4 19:41 notes.txt
```

Le fichier appartient au groupe `equipe`, et non au groupe `alice`. Les autres membres peuvent donc le modifier.

Sticky bit : chacun ses fichiers

Sur un dossier où tout le monde peut écrire, le **sticky bit** interdit de supprimer les fichiers des autres. C'est ce qui rend `/tmp` utilisable :

```
gary@serveur:~$ ls -ld /tmp
drwxrwxrwt 11 root root 4096 Oct  4 19:41 /tmp
```

On le reconnaît au `t` final. On le pose avec `chmod +t`, ou par un `1` devant les trois chiffres.

Droit	Chiffre	Affichage	Effet
SUID	4	s à la place du x du propriétaire	Le programme s'exécute avec l'identité de son propriétaire
SGID	2	s à la place du x du groupe	Sur un dossier : les nouveaux fichiers héritent de son groupe
Sticky bit	1	t à la place du x des autres	Sur un dossier : on ne supprime que ses propres fichiers

Un `S` ou un `T` majuscule signale que le droit spécial est posé sans le droit `x` correspondant. C'est presque toujours une erreur.

6.9 Les droits par défaut : umask

Quand vous créez un fichier, il reçoit des droits que vous n'avez pas choisis. Ils viennent du **umask**, un masque qui indique les droits à *retirer*.

Le système part de `666` pour un fichier et de `777` pour un dossier, puis retire ce que désigne le masque.

```
alice@serveur:~$ umask
0002
alice@serveur:~$ touch fichier.txt; mkdir dossier
alice@serveur:~$ ls -ld fichier.txt dossier
drwxrwxr-x 2 alice alice 4096 Oct  4 19:41 dossier
-rw-rw-r-- 1 alice alice    0 Oct  4 19:41 fichier.txt
```

Avec un masque de `002`, on retire le droit d'écriture aux autres : `664` pour les fichiers, `775` pour les dossiers.

umask	Fichiers	Dossiers	Usage
`002`	`664`	`775`	Valeur d'Ubuntu pour les comptes de personnes
`022`	`644`	`755`	Valeur d'Ubuntu pour `root` et les services
`027`	`640`	`750`	Rien pour les autres
`077`	`600`	`700`	Tout est privé

> **NOTE · Pourquoi 002 et non 022 ?**
> Sur Ubuntu, chaque utilisateur a un groupe principal à son nom, dont il est le seul membre. Laisser ce groupe écrire ne présente donc aucun risque. Et cela rend les dossiers partagés fonctionnels sans réglage : dans un dossier SGID, les fichiers créés sont modifiables par tout le groupe.

La commande `umask 027` change le masque du shell courant. Pour le rendre permanent, ajoutez-la à la fin de `~/.profile`.

6.10 Les ACL : au-delà de trois catégories

Le modèle propriétaire, groupe, autres ne sait pas exprimer une demande pourtant banale : « ce dossier appartient à l'équipe, et Max, qui n'en fait pas partie, doit pouvoir le lire ». Les **ACL** (*access control lists*) ajoutent des droits pour un utilisateur ou un groupe précis.

Les commandes viennent du paquet `acl`, à installer s'il est absent.

```
gary@serveur:~$ sudo setfacl -R -m u:max:rX /srv/projet
gary@serveur:~$ getfacl /srv/projet
# file: srv/projet
# owner: root
# group: equipe
# flags: -s-
user::rwx
user:max:r-x
group::rwx
mask::rwx
other::---
gary@serveur:~$ ls -ld /srv/projet
drwxrws---+ 2 root equipe 4096 Oct  4 19:41 /srv/projet
```

La ligne `user:max:r-x` est le droit ajouté. Le `+` à la fin des droits, dans `ls -l`, signale qu'un fichier porte des ACL. La ligne `mask` indique le maximum que peuvent obtenir les utilisateurs et les groupes nommés.

```
# Donner rwx à alice sur un fichier (en plus des permissions classiques)
setfacl -m u:alice:rwx document.txt

# Vérifier
getfacl document.txt
# file: document.txt
# owner: gary
# group: gary
```

```
user::rw-
user:alice:rwx
group::r--
mask::rwx
other::r--

# ls -l affiche un + en fin de permissions
ls -l document.txt
-rw-rwxr--+ 1 gary gary 123 avril 23 document.txt
#          ^ le + indique que des ACL étendues existent

# Autres opérations courantes :

# Donner r-- au groupe data-team
setfacl -m g:data-team:r-- fichier.txt

# Retirer les droits d'alice
setfacl -x u:alice fichier.txt

# Tout supprimer (revenir au rwx classique)
setfacl -b fichier.txt

# Récursif sur un dossier
setfacl -R -m u:alice:rwX dossier/

# ACL par défaut : appliquées aux fichiers créés ensuite dans le dossier
setfacl -d -m u:alice:rwx dossier-partage/
# Tout fichier créé dans dossier-partage/ sera accessible par alice
```

Les ACL *par défaut* (option `-d`) se posent sur un dossier. Elles sont copiées sur tout ce qui y sera créé ensuite.

> **ASTUCE · Groupes d'abord, ACL ensuite**
>
> Un besoin qui s'exprime avec un groupe doit être réglé avec un groupe. Les ACL ne se voient pas au premier coup d'œil, et on finit par les oublier. Gardez-les pour les exceptions.

> **ATTENTION · Les ACL se perdent à la copie**
>
> `cp` ne les conserve qu'avec `-a`. `rsync` demande l'option `-A`, que `-a` n'inclut pas. `tar` demande `--acls`. Après une restauration, vérifiez-les avec `getfacl`.

6.11 sudo

Le modèle d'Ubuntu

Sur Ubuntu, le compte `root` existe, mais son mot de passe est verrouillé. On ne peut donc pas ouvrir de session sous ce nom. À la place, les membres du groupe `sudo` lancent les commandes d'administration en les faisant précéder de `sudo`.

Ce fonctionnement a de vrais avantages. Aucun mot de passe d'administration ne circule entre collègues. Chaque commande est enregistrée dans le journal avec le nom de celui qui l'a lancée. Et l'on travaille le reste du temps avec des droits ordinaires, ce qui limite la portée d'une fausse manœuvre.

Au quotidien

```
sudo apt update              # une commande avec les droits de root
sudo -i                      # un shell root, à refermer avec exit
sudo -u www-data whoami      # une commande sous l'identité d'un autre compte
sudo -l                      # ce que sudo vous autorise
sudo -k                      # faire oublier le mot de passe tout de suite
```

`sudo` demande *votre* mot de passe, et s'en souvient quinze minutes.

Déléguer avec précision

Les règles se trouvent dans `/etc/sudoers`. Sur une installation neuve, le fichier contient ceci :

```
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/sbin:/bin:/snap/bin"
Defaults        use_pty
root    ALL=(ALL:ALL) ALL
%admin ALL=(ALL) ALL
%sudo   ALL=(ALL:ALL) ALL
@includedir /etc/sudoers.d
```

La ligne `%sudo ALL=(ALL:ALL) ALL` se lit ainsi : les membres du groupe `sudo` (le `%` désigne un groupe), sur toutes les machines, peuvent lancer sous n'importe quelle identité n'importe quelle commande.

On peut être bien plus restrictif. Pour qu'Alice puisse redémarrer le serveur web, et rien d'autre, créez un fichier dans `/etc/sudoers.d/` :

```
gary@serveur:~$ sudo visudo -f /etc/sudoers.d/web
```

```
alice ALL=(root) /usr/bin/systemctl restart nginx, /usr/bin/systemctl reload nginx
```

Alice peut maintenant taper `sudo systemctl restart nginx`. Toute autre commande lui sera refusée.

> **DANGER · Toujours passer par visudo**
>
> Une erreur de syntaxe dans les règles de `sudo` le rend inutilisable. Et sans `sudo`, sur Ubuntu, on ne peut plus rien réparer sans redémarrer en mode de récupération. `visudo` vérifie le fichier avant de l'enregistrer et refuse une version incorrecte.
>
> Deux précisions sur `/etc/sudoers.d/` : les fichiers dont le nom contient un point sont ignorés, et le mot-clé `NOPASSWD:`, qui dispense du mot de passe, ne doit porter que sur des commandes précises.

6.12 Changer d'identité

```
# Exécuter UNE commande en tant qu'alice (sudo demande votre mot de passe,
pas celui d'alice)
sudo -u alice whoami
alice

# Lancer un shell sous l'identité d'alice, en gardant votre environnement
sudo -u alice bash
# Toujours dans /home/gary, pas dans /home/alice

# Lancer un shell de connexion (comme su - alice, mais via sudo)
sudo -i -u alice
sudo -iu alice          # raccourci
alice@vostro:~$

# Devenir root avec shell de connexion (recommandé sous Ubuntu)
sudo -i

# Devenir root en gardant votre environnement (moins courant)
sudo -s
```

Commande	Mot de passe demandé	Ce qu'elle fait
`sudo commande`	Le vôtre	Lance la commande sous l'identité de `root`

Commande	Mot de passe demandé	Ce qu'elle fait
`sudo -i`	Le vôtre	Ouvre un shell `root` complet
`sudo -u alice commande`	Le vôtre	Lance la commande sous l'identité d'alice
`sudo -iu alice`	Le vôtre	Ouvre une session complète d'alice
`su - alice`	Celui d'alice	Ouvre une session complète d'alice
`su -`	Celui de `root`	Impossible sur Ubuntu : `root` n'a pas de mot de passe

Le tiret de `su - alice` et le `-i` de `sudo -i` ont le même rôle. Ils ouvrent une vraie session, avec l'environnement et le dossier personnel du compte visé. Sans eux, vous changez d'identité tout en gardant votre environnement, ce qui produit des erreurs difficiles à comprendre.

En résumé

- Le système identifie les utilisateurs par un UID et les groupes par un GID. Les comptes de personnes commencent à 1000.
- `/etc/passwd` décrit les comptes, `/etc/shadow` contient les empreintes des mots de passe, `/etc/group` les groupes.
- `adduser` crée un compte, `usermod -aG` l'ajoute à un groupe. Sans le `a`, `-G` remplace tous ses groupes.
- Trois droits (`r`, `w`, `x`) pour trois catégories (propriétaire, groupe, autres). Une seule catégorie s'applique, la première qui correspond.
- Sur un dossier, `x` permet d'y entrer et `w` d'y créer ou supprimer des fichiers.
- `chmod 644`, `chmod u+x` : deux notations pour la même chose. `chmod 777` est toujours une erreur.
- SUID fait tourner un programme sous l'identité de son propriétaire. SGID, sur un dossier, transmet son groupe. Le sticky bit protège les fichiers de chacun.
- Le umask d'un utilisateur d'Ubuntu vaut `002`, celui de `root` `022`.
- Les ACL ajoutent des droits pour un utilisateur ou un groupe précis.
- `sudo` remplace la connexion sous `root`. Ses règles se modifient avec `visudo`.

Atelier — Un dossier partagé pour une équipe

Objectif : trois personnes doivent travailler ensemble dans `/srv/projet`. Personne d'autre ne doit y entrer. Un intervenant extérieur doit pouvoir lire, sans rien modifier. Comptez quarante minutes.

Créer les comptes et le groupe

1. Créez les comptes `alice`, `bob` et `claire` avec `adduser`.
2. Créez le groupe `equipe` et ajoutez-y les trois comptes. Vérifiez avec `getent group equipe`.

Créer le dossier

3. Créez `/srv/projet`. Donnez-le à `root` et au groupe `equipe`.
4. Appliquez les droits `2770`. Que signifie chacun des quatre chiffres ?
5. Vérifiez avec `ls -ld` : vous devez lire `drwxrws---`.

Vérifier que le partage fonctionne

6. Ouvrez une session d'alice avec `sudo -iu alice`. Créez `/srv/projet/notes.txt`. À quel groupe appartient-il ?
7. Sous l'identité de bob, ajoutez une ligne au fichier, puis supprimez-le. Les deux opérations doivent réussir.
8. Avec votre propre compte, essayez `cd /srv/projet`. Vous devez obtenir *Permission denied*.

Ajouter un lecteur avec une ACL

9. Créez le compte `max`. Donnez-lui le droit de lire le dossier : `sudo setfacl -R -m u:max:rX /srv/projet`.
10. Faites en sorte que ce droit s'applique aussi aux fichiers à venir : `sudo setfacl -d -m u:max:rX /srv/projet`.
11. Sous l'identité de max, vérifiez que vous pouvez lire un fichier et que `touch essai.txt` échoue.

Faire le ménage

12. Supprimez les quatre comptes avec `deluser --remove-home`, le groupe, puis le dossier.

Exercices

Exercice 1 — Lire les droits

Pour chaque ligne, donnez le type du fichier, la valeur des droits en chiffres, et dites qui peut faire quoi.

```
-rwxr-xr-- 1 gary equipe   1234 Apr 23 14:00 deploiement.sh
drwxrws--- 2 root equipe   4096 Apr 10 09:00 projet
-rw------- 1 gary gary      411 Apr 22 08:15 id_ed25519
drwxrwxrwt 11 root root    4096 Apr 23 16:00 tmp
-rwsr-xr-x 1 root root    64152 May 30  2024 passwd
```

Exercice 2 — Trouver la commande

1. Rendre un script lisible et exécutable par tous, modifiable par son seul propriétaire.
2. Rendre une clé privée accessible à son seul propriétaire.
3. Donner à `www-data` le dossier `/var/www/site` et tout son contenu.
4. Ajouter `bob` au groupe `docker` sans lui retirer ses autres groupes.
5. Donner à `alice` le droit de lire un fichier, sans toucher à son groupe.

Exercice 3 — Qu'est-ce qui ne va pas ?

1. `sudo usermod -G docker gary`, alors que `gary` appartenait à `sudo` et `adm`.
2. `chmod -R 777 /var/www`, « pour régler un problème de droits ».
3. Bob vient d'être ajouté au groupe `equipe`, mais il ne peut toujours pas entrer dans `/srv/projet`.
4. Un fichier porte les droits `-r--rw-rw-`. Son propriétaire n'arrive pas à le modifier, alors que tous les autres y parviennent.
5. Alice a tous les droits sur `/srv/donnees/rapport.txt`, mais elle obtient *Permission denied* en voulant le lire. Le dossier `/srv/donnees` porte les droits `drwx------ root root`.

Exercice 4 — Calculer

Quels droits reçoivent un fichier et un dossier créés avec un umask de `022` ? de `002` ? de `027` ? de `077` ?

Exercice 5 — Un audit rapide

Quelle commande répond à chaque question ?

1. Quels sont les comptes de personnes (UID supérieur ou égal à 1000) ?
2. Quels programmes portent le droit SUID ?
3. Qui appartient au groupe `sudo` ?
4. Quels fichiers de `/home` sont modifiables par tout le monde ?
5. Qui s'est connecté récemment à la machine ?

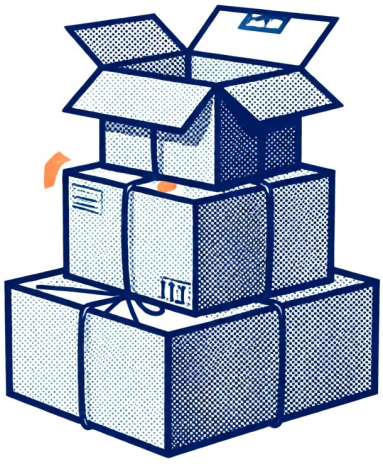
Pour aller plus loin

Sur la machine. `man 5 passwd`, `man 5 shadow`, `man chmod`, `man 5 sudoers` (long, mais complet), `man acl`.

Aller plus loin dans le modèle. Les *capabilities* découpent les pouvoirs de `root` en une quarantaine de droits distincts, que l'on peut accorder un par un à un programme (`man 7 capabilities`). AppArmor, présenté au chapitre 22, ajoute une couche de contrôle que même `root` ne peut pas contourner.

Sécurité. Le chapitre 19 montre comment exiger un second facteur à la connexion SSH. Le chapitre 20 reprend la gestion des comptes sous l'angle du durcissement : politique de mots de passe, restrictions de `sudo`, audit.

Lecture. *UNIX and Linux System Administration Handbook*, d'Evi Nemeth et de ses coauteurs. Le chapitre sur le contrôle d'accès replace tout ce modèle dans son histoire.



CHAPITRE 7

La gestion des paquets

D'où viennent les logiciels, comment les installer, les tenir à jour et les retirer proprement.

DANS CE CHAPITRE

- Expliquer ce qu'apporte un gestionnaire de paquets et distinguer APT de dpkg.
- Savoir dans quel dépôt se trouve un paquet, et ce que cela implique.
- Lire et modifier la liste des dépôts, au format deb822.
- Installer, mettre à jour, supprimer et inspecter des paquets avec APT.
- Ajouter un PPA ou un dépôt tiers avec sa clé de signature.
- Installer un fichier `.deb` téléchargé.
- Utiliser Snap, Flatpak et AppImage, et choisir entre eux.
- Automatiser les mises à jour de sécurité.

> **AVANT DE COMMENCER**
>
> Les chapitres 3 à 6 : `sudo` , les permissions, les redirections. Les exemples conviennent aux éditions Desktop et Server. Les sorties de commandes sont en anglais.

7.1 Pourquoi un gestionnaire de paquets

Sous Windows, on a longtemps installé un logiciel en téléchargeant un programme d'installation sur le site de son éditeur. Chaque logiciel apportait ses propres bibliothèques, se mettait à jour à sa façon, et laissait des traces à la désinstallation.

Les distributions Linux ont choisi une autre voie dès le début des années 1990 : confier tous les logiciels à un seul outil, le gestionnaire de paquets. Il apporte cinq choses.

- **Les dépendances sont gérées.** Si le logiciel A a besoin de la bibliothèque B, B est installée avec lui. Si dix logiciels en ont besoin, elle n'est installée qu'une fois.
- **Les paquets sont signés.** Une signature numérique garantit que le paquet vient bien de la distribution et n'a pas été modifié en chemin.
- **La source est unique.** On n'a pas à visiter cinq sites pour installer cinq logiciels.
- **La désinstallation est propre.** Le système sait quel fichier appartient à quel paquet.
- **Tout se met à jour d'un coup**, le système comme les applications.

Sur Ubuntu, cet outil a deux étages.

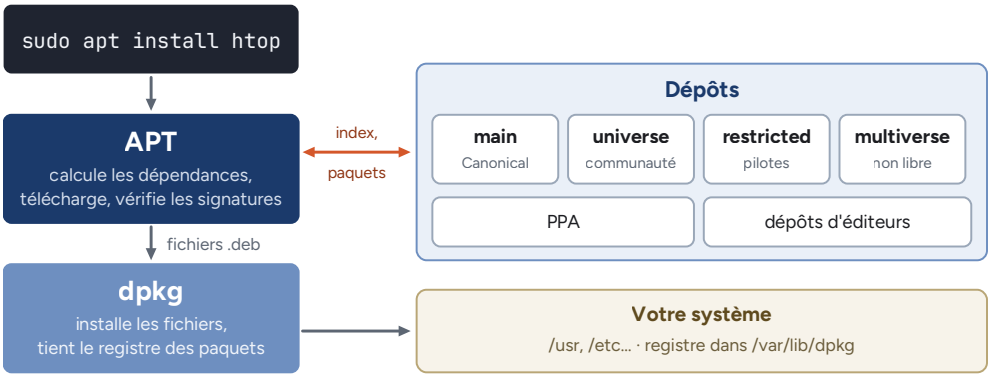


Figure 7.1 — La chaîne d'installation d'un paquet.

Un **paquet** est un fichier `.deb`. C'est une archive qui contient les fichiers du logiciel, sa description (nom, version, dépendances) et de petits scripts à exécuter à l'installation et à la suppression.

dpkg installe un fichier `.deb` et tient le registre de ce qui est installé. Il ne sait pas télécharger, et il ne résout pas les dépendances. Si quelque chose manque, il le signale et s'arrête là.

APT fait le reste. Il connaît les dépôts, calcule ce qu'il faut installer, télécharge les paquets, vérifie leur signature, puis les confie à dpkg.

Au quotidien, on parle à APT. On s'adresse à dpkg pour installer un fichier `.deb` isolé, ou pour interroger le registre.

> **NOTE · apt, apt-get, apt-cache**
>
> Les commandes `apt-get` et `apt-cache` sont les outils d'origine. La commande `apt`, apparue en 2014, les réunit sous une forme plus agréable, avec couleurs et barre de progression. Elle est faite pour le clavier. Dans un script, gardez `apt-get`, dont la sortie ne change pas d'une version à l'autre.

Interroger le registre avec dpkg

```
# Voir les infos d'un paquet
dpkg -I /var/cache/apt/archives/htop_3.3.0-4build1_amd64.deb

# Lister les fichiers d'un paquet installé
dpkg -L htop
/usr/bin/htop
/usr/share/applications/htop.desktop
/usr/share/man/man1/htop.1.gz
...

# Quel paquet appartient à ce fichier ?
dpkg -S /usr/bin/vim
vim: /usr/bin/vim

# État d'un paquet
dpkg -s htop
Package: htop
Status: install ok installed
Version: 3.3.0-4build1
...

# Lister tous les paquets installés
dpkg -l
```

```
# Chercher dans la liste des installés
dpkg -l | grep python
```

`dpkg -S` est très utile dans l'autre sens : face à un fichier inconnu, il dit quel paquet l'a installé.

7.2 Les dépôts d'Ubuntu

Un dépôt est un serveur qui met à disposition des paquets et leur index. Ceux d'Ubuntu sont organisés selon deux axes.

Quatre composants

Composant	Logiciels	Maintenu par	Exemples
`main`	Libres	Canonical, pendant toute la vie de la version	`bash`, `nginx`, `postgresql`, `htop`
`universe`	Libres	La communauté	`vlc`, `gimp`, `nodejs`, `mariadb-server`
`restricted`	Propriétaires	Canonical, dans la mesure du possible	`nvidia-driver-550`
`multiverse`	Propriétaires ou soumis à restrictions	Personne	`steam-installer`, `unrar`

La distinction compte pour la sécurité. Les paquets de `main` reçoivent des correctifs pendant cinq ans. Ceux de `universe` en reçoivent quand un bénévole s'en charge. L'abonnement Ubuntu Pro, gratuit jusqu'à cinq machines, étend la couverture de Canonical à `universe`.

Pour savoir d'où vient un paquet :

```
gary@serveur:~$ apt-cache policy vlc
vlc:
  Installed: (none)
  Candidate: 3.0.20-3build6
  Version table:
     3.0.20-3build6 500
        500 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages
```

Cinq suites

Dans les dépôts, Ubuntu 24.04 porte son nom de code, `noble`. Chaque composant existe en plusieurs *suites* :

Suite	Contenu
`noble`	Les paquets tels qu'ils étaient le jour de la sortie. Figée.
`noble-security`	Les correctifs de sécurité.
`noble-updates`	Les correctifs de bogues, et de nouveau les correctifs de sécurité.
`noble-backports`	Des versions plus récentes de quelques logiciels. Activée, mais jamais installée sans demande explicite.
`noble-proposed`	Les mises à jour en cours de validation. À ne pas activer.

7.3 La liste des dépôts

La configuration d'APT se trouve dans `/etc/apt/`. Ubuntu 24.04 décrit ses dépôts dans le fichier `/etc/apt/sources.list.d/ubuntu.sources`, au format *deb822* :

```
Types: deb
URIs: http://archive.ubuntu.com/ubuntu/
Suites: noble noble-updates noble-backports
Components: main universe restricted multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg

Types: deb
URIs: http://security.ubuntu.com/ubuntu/
Suites: noble-security
Components: main universe restricted multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

Chaque bloc décrit une source : le type de paquets, l'adresse du serveur, les suites, les composants, et la clé qui sert à vérifier les signatures.

Vous croiserez encore l'ancien format, où chaque dépôt tient sur une ligne, dans des fichiers `.list` :

```
deb http://archive.ubuntu.com/ubuntu/ noble main restricted universe multiverse
```

Il reste accepté. Sur Ubuntu 24.04, l'ancien fichier `/etc/apt/sources.list` ne contient plus qu'un commentaire qui renvoie au nouveau.

> **À RETENIR · Un fichier par dépôt**
>
> Ne touchez pas à `ubuntu.sources`. Pour ajouter un dépôt, créez un fichier à part dans `/etc/apt/sources.list.d/`. Le jour où vous n'en voulez plus, il suffit de le supprimer.

7.4 APT au quotidien

Mettre à jour

```
sudo apt update          # télécharge les index : ce qui existe, en quelle version
sudo apt upgrade         # installe les nouvelles versions des paquets installés
```

`apt update` n'installe rien. Il rafraîchit la liste de ce qui est disponible. C'est `apt upgrade` qui met le système à jour. On lance toujours les deux, dans cet ordre.

```
# upgrade : met à jour les paquets sans jamais retirer ni ajouter
sudo apt upgrade

# full-upgrade : autorise aussi la suppression de paquets quand la mise à jour l'exige
sudo apt full-upgrade

# Sans aucune question (scripts)
sudo DEBIAN_FRONTEND=noninteractive apt-get -y full-upgrade

# Voir ce qui serait mis à jour, sans le faire
apt list --upgradable

# Simulation (-s)
sudo apt -s full-upgrade
```

> **NOTE · Des paquets « kept back » ?**
>
> Il arrive qu'`apt upgrade` annonce que certains paquets sont *conservés*. Le plus souvent, c'est normal. Ubuntu diffuse ses mises à jour par vagues : chaque machine reçoit une mise à jour donnée un peu plus tôt ou un peu plus tard, pour qu'un défaut soit repéré avant d'atteindre tout le monde. Le paquet arrivera dans les jours qui suivent.
>
> L'autre cas est celui d'une mise à jour qui exige de retirer un paquet, ce qu'`upgrade` s'interdit. `sudo apt full-upgrade` l'accepte. Lisez alors la liste de ce qui sera supprimé avant de valider.

Installer

```
# Installation standard
sudo apt install htop

# Plusieurs paquets d'un coup
sudo apt install htop btop tmux ripgrep fzf

# Installer une version précise
sudo apt install htop=3.3.0-4build1

# Installer sans confirmation (-y)
sudo apt install -y htop

# Installer sans les paquets recommandés (allège l'installation)
sudo apt install --no-install-recommends htop

# Réinstaller un paquet (utile si des fichiers ont été supprimés)
sudo apt install --reinstall htop

# Installer à partir d'un fichier .deb local (voir section 7.7)
sudo apt install ./paquet-local.deb
```

Avant de confirmer, APT résume ce qu'il va faire : paquets installés, mis à jour, supprimés, et place nécessaire. Prenez l'habitude de lire ce résumé, surtout la ligne des suppressions.

Supprimer

```
# Supprimer un paquet (mais conserver sa configuration)
sudo apt remove htop

# Supprimer ET effacer sa configuration
sudo apt purge htop

# Supprimer les dépendances devenues inutiles
sudo apt autoremove

# Combinaison courante après désinstallation d'un gros paquet
sudo apt autoremove --purge
```

`remove` laisse en place les fichiers de configuration du paquet, dans `/etc`. Si vous le réinstallez, vous retrouvez vos réglages. `purge` les supprime aussi. Dans les deux cas, les données ne sont pas touchées : après la suppression de PostgreSQL, les bases restent dans `/var/lib/postgresql`.

Chercher et inspecter

```
gary@serveur:~$ apt show htop
Package: htop
Version: 3.3.0-4build1
Priority: optional
Section: utils
Origin: Ubuntu
Installed-Size: 434 kB
Depends: libc6 (>= 2.38), libncursesw6 (>= 6), libnl-3-200 (>= 3.2.7), libtinfo6 (>= 6)
Suggests: lm-sensors, lsof, strace
Homepage: https://htop.dev/
Download-Size: 171 kB
APT-Sources: http://archive.ubuntu.com/ubuntu noble/main amd64 Packages
Description: interactive processes viewer
```

```
apt search "moniteur"            # chercher dans les noms et les descriptions
apt list --installed             # tout ce qui est installé
apt list --upgradable            # ce qui peut être mis à jour
apt-mark showmanual              # ce que vous avez demandé vous-même
```

La dernière commande est précieuse le jour où l'on veut reconstruire une machine. Elle donne la liste des paquets installés volontairement, sans leurs dépendances.

Retenir une version, faire le ménage

```
sudo apt-mark hold postgresql-16     # ne plus mettre ce paquet à jour
sudo apt-mark unhold postgresql-16   # lever l'interdiction
sudo apt autoremove                  # retirer les dépendances devenues inutiles
sudo apt clean                       # vider le cache des paquets téléchargés
```

`apt autoremove` supprime aussi les anciens noyaux, qui s'accumulent sinon dans `/boot`.

Commande	Ce qu'elle fait
`sudo apt update`	Télécharger la liste des paquets disponibles
`sudo apt upgrade`	Mettre à jour sans ajouter/retirer de paquets
`sudo apt full-upgrade`	Mettre à jour en autorisant ajout/suppression
`sudo apt install paquet`	Installer un paquet
`sudo apt remove paquet`	Supprimer sans toucher à la config
`sudo apt purge paquet`	Supprimer avec config
`sudo apt autoremove`	Supprimer les dépendances devenues inutiles

Commande	Ce qu'elle fait
`apt search motif`	Chercher dans les noms et descriptions
`apt show paquet`	Voir les détails d'un paquet
`apt list --installed`	Lister les paquets installés
`apt list --upgradable`	Lister les mises à jour disponibles
`sudo apt clean`	Vider le cache des .deb téléchargés

7.5 PPA et dépôts tiers

Les dépôts d'Ubuntu ne contiennent pas tout, et les versions y sont figées. Pour un logiciel absent ou une version plus récente, on ajoute un dépôt.

> **ATTENTION · Ajouter un dépôt, c'est lui donner les droits de root**
>
> Les paquets d'un dépôt s'installent avec les droits de l'administrateur, et leurs scripts font ce qu'ils veulent. N'ajoutez que des dépôts dont vous connaissez l'origine : celui de l'éditeur du logiciel, ou un PPA tenu par une équipe identifiée.

Les PPA

Un PPA (*Personal Package Archive*) est un dépôt hébergé sur Launchpad, la plateforme de développement d'Ubuntu. Les développeurs de Git en maintiennent un, par exemple, pour publier leurs dernières versions.

```
# La commande add-apt-repository est le plus simple
sudo add-apt-repository ppa:git-core/ppa
# Téléchargement automatique de la clé GPG,
# ajout d'une source, invitation à faire apt update

sudo apt update
sudo apt install git
# → installe la dernière version de git

# Voir les PPA installés
ls /etc/apt/sources.list.d/
git-core-ubuntu-ppa-noble.sources

# Supprimer un PPA
sudo add-apt-repository --remove ppa:git-core/ppa
```

```
# Ou manuellement
sudo rm /etc/apt/sources.list.d/git-core-ubuntu-ppa-noble.sources
sudo apt update
```

La commande crée un fichier dans `/etc/apt/sources.list.d/`. Il contient à la fois l'adresse du dépôt et sa clé de signature.

Le dépôt d'un éditeur

Beaucoup d'éditeurs tiennent leur propre dépôt : Docker, Grafana, PostgreSQL, Microsoft. La marche à suivre est toujours la même, en trois temps. Prenons Grafana, un outil de tableaux de bord absent des dépôts d'Ubuntu.

```
# 1. Télécharger la clé de signature de l'éditeur
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://apt.grafana.com/gpg.key | sudo gpg --dearmor -o /etc/apt/
keyrings/grafana.gpg

# 2. Décrire le dépôt
sudo tee /etc/apt/sources.list.d/grafana.sources > /dev/null <<EOF
Types: deb
URIs: https://apt.grafana.com
Suites: stable
Components: main
Signed-By: /etc/apt/keyrings/grafana.gpg
EOF

# 3. Rafraîchir les index
sudo apt update
```

Le paquet `grafana` peut alors être installé avec `apt install`. Il sera mis à jour avec le reste du système.

La ligne `Signed-By` a son importance. Elle indique que cette clé ne vaut que pour ce dépôt. Un dépôt tiers ne peut donc pas se faire passer pour celui d'Ubuntu.

> **DANGER · apt-key appartient au passé**
>
> D'anciens tutoriels font ajouter les clés avec `apt-key add`. Cette commande plaçait la clé dans un trousseau commun, où elle était acceptée pour *tous* les dépôts. Elle est dépréciée. Quand vous en rencontrez une, remplacez-la par la méthode ci-dessus : une clé dans `/etc/apt/keyrings/`, désignée par `Signed-By`.

7.6 Installer un fichier .deb

Certains éditeurs fournissent simplement un fichier `.deb` à télécharger. On l'installe avec APT, en donnant son chemin :

```
sudo apt install ./logiciel_1.2.3_amd64.deb
```

Le `./` est indispensable. Sans lui, APT croit qu'on lui donne un nom de paquet et le cherche dans les dépôts. Avec lui, il installe le fichier et va chercher ses dépendances.

L'ancienne méthode passe par dpkg et demande une seconde commande, pour installer les dépendances qu'il n'a pas su résoudre :

```
sudo dpkg -i logiciel_1.2.3_amd64.deb
sudo apt install -f
```

Retenez `apt install -f`. Il répare un système où des paquets sont restés à moitié installés.

```
# Supprimer un paquet installé par .deb
sudo dpkg -r nom-du-paquet       # supprime, garde la configuration
sudo dpkg -P nom-du-paquet       # supprime tout

# Rejouer les questions de configuration d'un paquet
sudo dpkg-reconfigure tzdata     # ex: reconfigurer le fuseau horaire
sudo dpkg-reconfigure keyboard-configuration

# Voir ce qu'il reste à configurer
sudo dpkg --configure -a

# Chercher les paquets dans un état incohérent
sudo dpkg --audit
```

> **NOTE · Certains .deb ajoutent leur dépôt**
>
> Les paquets `.deb` de Google Chrome ou de Visual Studio Code installent aussi le dépôt de leur éditeur. C'est pratique, car le logiciel se met ensuite à jour tout seul. Sachez-le, et regardez de temps en temps ce que contient `/etc/apt/sources.list.d/`.

7.7 Snap

Le chapitre 3 a présenté Snap en quelques lignes. Regardons-le de plus près.

Un snap est une image en lecture seule, qui contient l'application et ses bibliothèques. Elle est montée dans `/snap`, et l'application s'exécute dans un espace *confiné*. AppArmor limite ce qu'elle peut voir et faire, et elle doit obtenir des autorisations, appelées *interfaces*, pour accéder au réseau, à la caméra ou à vos fichiers.

Certains logiciels ne peuvent pas fonctionner ainsi : un éditeur de code doit lire tous vos fichiers et lancer d'autres programmes. Leurs snaps s'installent avec l'option `--classic`, qui lève le confinement.

```
# Chercher un snap
snap find code

# Voir les détails
snap info code

# Installer (stable par défaut)
sudo snap install code --classic

# Installer depuis un canal précis (stable / candidate / beta / edge)
sudo snap install code --edge

# Lister les snaps installés
snap list

# Mettre à jour tout de suite (sinon, la mise à jour est automatique)
sudo snap refresh                # tout
sudo snap refresh firefox        # un snap précis

# Voir les mises à jour en attente
snap refresh --list

# Revenir à la version précédente
sudo snap revert firefox

# Désinstaller
sudo snap remove discord

# Désinstaller sans garder de copie des données
sudo snap remove --purge discord

# Voir les autorisations d'un snap
snap connections firefox

# Accorder une autorisation
sudo snap connect firefox:removable-media
```

Les canaux

Chaque snap est publié sur quatre canaux : `stable`, `candidate`, `beta` et `edge`, du plus sûr au plus récent. On en change avec `sudo snap refresh firefox --channel=beta`.

Maîtriser les mises à jour

Les snaps se mettent à jour seuls, plusieurs fois par jour si nécessaire. Sur un poste de travail, c'est un confort. Sur un serveur, on préfère choisir le moment.

```
sudo snap refresh --hold=72h nextcloud     # suspendre trois jours pour ce snap
sudo snap refresh --hold                   # suspendre pour tous, sans limite
sudo snap refresh --unhold                 # reprendre
sudo snap set system refresh.timer=sat,03:00-05:00    # ne mettre à jour que le
samedi, de 3 h à 5 h
```

Faut-il utiliser les snaps ?

Le débat est vif, et chaque camp a de bons arguments.

Pour. L'application est confinée. Les éditeurs publient eux-mêmes leurs versions, sans attendre la distribution. Un retour à la version précédente tient en une commande.

Contre. Les paquets sont gros. Le premier lancement est plus lent. Le confinement gêne parfois : le snap de Firefox ne voit pas le dossier `/tmp` du système. Et le magasin dépend de Canonical seule.

Une ligne de conduite raisonnable : APT pour tout ce qui fait le système et pour les services d'un serveur ; un snap quand c'est le canal officiel de l'éditeur ou la seule façon d'obtenir une version récente.

7.8 Flatpak

Flatpak répond au même besoin que Snap, avec d'autres choix. Son catalogue principal, Flathub, est géré par une communauté. Il ne s'occupe que des applications de bureau.

```
# Installer Flatpak
sudo apt install flatpak

# Ajouter Flathub comme source
flatpak remote-add --if-not-exists flathub \
```

```
  https://dl.flathub.org/repo/flathub.flatpakrepo

# Rechercher une application
flatpak search spotify

# Installer depuis Flathub
flatpak install flathub com.spotify.Client

# Lancer
flatpak run com.spotify.Client

# Lister les applications Flatpak installées
flatpak list

# Mettre à jour
flatpak update

# Désinstaller
flatpak uninstall com.spotify.Client

# Faire le ménage des runtimes inutilisés
flatpak uninstall --unused
```

Les applications portent un nom en trois parties, comme `org.videolan.VLC`. Elles reposent sur des *runtimes*, des socles communs partagés entre plusieurs applications. La première installation en télécharge un, ce qui explique sa taille.

	Snap	Flatpak
Catalogue	Snap Store, géré par Canonical	Flathub, géré par la communauté
Installé sur Ubuntu	Oui	Non, à ajouter
Confinement	AppArmor	bubblewrap
Mises à jour	Automatiques, suspendables	À la demande
Services, outils en ligne de commande	Oui	Non

7.9 AppImage

Une AppImage est un fichier unique, qui contient l'application et ses bibliothèques. Il n'y a rien à installer. On le rend exécutable et on le lance.

```
sudo apt install libfuse2t64            # nécessaire une fois, sur Ubuntu 24.04
chmod +x Logiciel-1.2.3-x86_64.AppImage
./Logiciel-1.2.3-x86_64.AppImage
```

C'est simple, et cela ne demande aucun droit d'administration. Les contreparties sont sérieuses : pas de mise à jour automatique, pas de confinement, pas d'entrée dans le menu des applications.

> **ATTENTION · Deux obstacles sur Ubuntu 24.04**
>
> Les AppImages ont besoin de la bibliothèque FUSE dans sa version 2, qui n'est plus installée par défaut. C'est le rôle du paquet `libfuse2t64`. Par ailleurs, les applications construites avec Electron peuvent refuser de démarrer : leur mécanisme d'isolation interne se heurte à la restriction des espaces de noms décrite au chapitre 1. Le chapitre 22 explique comment leur donner l'autorisation nécessaire.

7.10 Compiler depuis les sources

Compiler un logiciel soi-même a longtemps fait partie du quotidien sous Linux. C'est devenu rare. Cela reste utile quand le logiciel n'est empaqueté nulle part, quand il faut une option absente de la version distribuée, ou quand on contribue à son développement.

```
# 1. Installer les outils de compilation
sudo apt install build-essential
# build-essential = gcc, g++, make, libc-dev — les bases

# Si le logiciel a des dépendances de bibliothèques
sudo apt install libssl-dev libncurses-dev
# Convention : libfoo-dev contient ce qu'il faut pour compiler avec la bibliothèque libfoo

# 2. Télécharger et extraire les sources
wget https://example.com/monprog-1.0.tar.gz
tar -xzf monprog-1.0.tar.gz
cd monprog-1.0/

# 3. Configurer (vérifie les dépendances, génère le Makefile)
./configure
# Ou avec des options :
./configure --prefix=/usr/local --enable-xyz

# 4. Compiler
make -j$(nproc)
# -j$(nproc) = paralléliser sur tous les cœurs disponibles

# 5. Installer
sudo make install
# Les binaires vont dans /usr/local/bin/ par défaut (ne pollue pas /usr/bin)
```

```
# 6. Plus tard, pour désinstaller (si le Makefile le supporte)
sudo make uninstall
```

Beaucoup de projets récents utilisent d'autres outils, comme CMake ou Meson. Leur fichier `README` ou `INSTALL` indique la marche à suivre.

> **À RETENIR · Ce qui est compilé à la main échappe au système**
>
> Un logiciel installé par `make install` n'est connu ni d'APT ni de dpkg. Il ne recevra aucune mise à jour de sécurité. Il s'installe dans `/usr/local`, justement pour ne pas se mélanger avec les fichiers du système. Notez quelque part ce que vous avez installé ainsi, et conservez le dossier des sources : c'est lui qui permettra `make uninstall`.

7.11 Les mises à jour automatiques

Une faille corrigée n'est plus un danger, à condition que le correctif soit installé. Sur Ubuntu, le paquet `unattended-upgrades` s'en charge. Il est installé et actif par défaut sur toutes les éditions.

Deux fichiers le configurent. Le premier, `/etc/apt/apt.conf.d/20auto-upgrades`, l'active :

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
```

Chaque jour, les index sont rafraîchis et les mises à jour installées.

Le second, `/etc/apt/apt.conf.d/50unattended-upgrades`, dit lesquelles :

```
Unattended-Upgrade::Allowed-Origins {
        "${distro_id}:${distro_codename}";
        "${distro_id}:${distro_codename}-security";
        "${distro_id}ESMApps:${distro_codename}-apps-security";
        "${distro_id}ESM:${distro_codename}-infra-security";
//      "${distro_id}:${distro_codename}-updates";
};
```

Par défaut, seuls les correctifs de sécurité sont installés. La ligne `-updates`, commentée par deux barres obliques, ajouterait les correctifs de bogues.

Plus bas dans le même fichier, trois réglages méritent votre attention sur un serveur :

```
Unattended-Upgrade::Automatic-Reboot "true";
Unattended-Upgrade::Automatic-Reboot-Time "03:00";
Unattended-Upgrade::Remove-Unused-Kernel-Packages "true";
```

ATTENTION · Sans redémarrage, le nouveau noyau ne sert à rien

Un noyau mis à jour n'est utilisé qu'après un redémarrage. Tant que la machine ne redémarre pas, elle tourne sur l'ancien, avec ses failles. Quand un redémarrage est attendu, le fichier `/var/run/reboot-required` existe. Deux politiques sont possibles : laisser la machine redémarrer seule à heure fixe, comme ci-dessus, ou le faire à la main lors d'une plage de maintenance. Ne rien décider revient à ne jamais redémarrer.

```
sudo unattended-upgrade --dry-run --debug              # voir ce qui serait fait
cat /var/log/unattended-upgrades/unattended-upgrades.log   # voir ce qui a été fait
```

Ubuntu Pro et Livepatch

L'abonnement Ubuntu Pro, gratuit jusqu'à cinq machines, apporte deux services utiles sur un serveur : les correctifs de sécurité pour les paquets de `universe`, et **Livepatch**, qui applique les correctifs les plus graves au noyau en cours d'exécution, sans redémarrage.

```
# 1. Créer un compte Ubuntu One (gratuit)
# Sur https://ubuntu.com/pro

# 2. Récupérer votre token Pro et l'attacher à la machine
sudo pro attach VOTRE-TOKEN

# 3. Activer le service livepatch
sudo pro enable livepatch

# Vérifier
sudo pro status
canonical-livepatch status

# Voir les patchs appliqués
sudo canonical-livepatch status --verbose
```

Livepatch ne dispense pas de redémarrer un jour. Il permet de choisir le moment.

En résumé

- APT télécharge et résout les dépendances, dpkg installe et tient le registre.
- `main` est maintenu par Canonical, `universe` par la communauté. `restricted` et `multiverse` contiennent des logiciels non libres.
- Les dépôts d'Ubuntu 24.04 sont décrits dans `/etc/apt/sources.list.d/ubuntu.sources`, au format deb822.
- `apt update` rafraîchit les index, `apt upgrade` installe les mises à jour.
- `remove` garde la configuration, `purge` la supprime, `autoremove` retire les dépendances inutiles.
- Un dépôt tiers s'ajoute avec un fichier à part et une clé désignée par `Signed-By`. `apt-key` est à proscrire.
- `sudo apt install ./fichier.deb` installe un paquet téléchargé, avec ses dépendances.
- Snap et Flatpak fournissent des applications confinées et récentes. Une AppImage est un simple fichier exécutable.
- Les correctifs de sécurité s'installent seuls. Il faut encore décider quand la machine redémarre.

Atelier — Un logiciel, trois façons de l'installer

Objectif : installer le lecteur multimédia VLC par APT, par Snap et par Flatpak, comparer, puis ajouter un dépôt tiers. Comptez quarante minutes, sur Ubuntu Desktop.

Par APT

1. Cherchez le paquet avec `apt search vlc`, puis affichez sa fiche avec `apt show vlc`. Dans quel composant se trouve-t-il ? Quelle est sa version ?
2. Installez-le. Combien de paquets APT installe-t-il en tout ? Pourquoi ?
3. Lancez VLC, notez la version affichée dans *Aide*, *À propos*.
4. Désinstallez-le avec `apt purge`, puis lancez `apt autoremove`. Que se passe-t-il ?

Par Snap

5. Affichez `snap info vlc`. Quelles versions proposent les canaux `stable` et `edge` ?

6. Installez le snap, lancez-le et comparez la version à celle d'APT.
7. Affichez ses autorisations avec `snap connections vlc`.
8. Désinstallez-le.

Par Flatpak

9. Installez Flatpak et ajoutez Flathub, comme en section 7.8.
10. Installez `org.videolan.VLC`. Qu'est-ce qui est téléchargé en plus de l'application ?
11. Lancez-le, puis désinstallez-le, ainsi que les runtimes inutilisés.

Un dépôt tiers

12. Ajoutez le dépôt de Grafana, comme en section 7.5. `sudo apt update` ne doit afficher aucun avertissement.
13. Vérifiez avec `apt-cache policy grafana` que le paquet est proposé. Ne l'installez pas.
14. Retirez le dépôt : supprimez le fichier `.sources` et la clé, puis relancez `apt update`.

Bilan. Remplissez un tableau à trois colonnes : version obtenue, place occupée, temps du premier lancement.

Exercices

Exercice 1 — Diagnostiquer

1. `sudo apt upgrade` annonce que trois paquets sont conservés. Citez deux explications possibles.
2. Après `sudo apt remove mysql-server`, il reste 2 Go dans `/var/lib/mysql`. Pourquoi ? Que faire pour tout supprimer ?
3. `dpkg -i` s'arrête sur « dependency problems ». Quelle commande lancez-vous ensuite ?
4. Un tutoriel commence par `curl … | sudo apt-key add -`. Comment l'adaptez-vous ?
5. `sudo apt install logiciel.deb` répond « Unable to locate package ». Qu'avez-vous oublié ?

Exercice 2 — Trouver la commande

1. Lister les paquets installés dont le nom contient `python`.
2. Trouver quel paquet a installé le fichier `/usr/bin/git`.
3. Lister les fichiers installés par le paquet `openssh-server`.
4. Empêcher la mise à jour du paquet `nginx`.
5. Afficher la liste des paquets que vous avez installés vous-même.

Exercice 3 — Quelle méthode ?

Pour chaque logiciel, dites quelle méthode vous choisiriez et pourquoi.

1. PostgreSQL, sur un serveur de production.
2. Le client de messagerie d'un éditeur, qui ne publie qu'un snap.
3. Un outil graphique de retouche dont vous voulez la toute dernière version.
4. Nginx, sur un serveur qui héberge plusieurs sites.
5. Un petit utilitaire distribué uniquement sous forme d'AppImage.

Exercice 4 — Lire un fichier de dépôt

```
Types: deb
URIs: https://download.docker.com/linux/ubuntu
Suites: noble
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
```

1. Quel est le rôle de chaque ligne ?
2. Où ce fichier doit-il être placé ?
3. Que se passe-t-il si le fichier `docker.asc` n'existe pas ?

Exercice 5 — Un script de mise à jour

Écrivez un script qui rafraîchit les index, installe les mises à jour, retire les paquets inutiles, met à jour les snaps, puis indique si un redémarrage est nécessaire. Le chapitre 10 vous donnera de quoi le rendre robuste.

Pour aller plus loin

Sur la machine. `man apt`, `man 5 sources.list` pour le format deb822, `man apt_preferences` pour donner des priorités aux dépôts.

En ligne. `snapcraft.io/docs` et `docs.flatpak.org` pour la documentation de Snap et de Flatpak. `documentation.ubuntu.com/pro/` pour Ubuntu Pro.

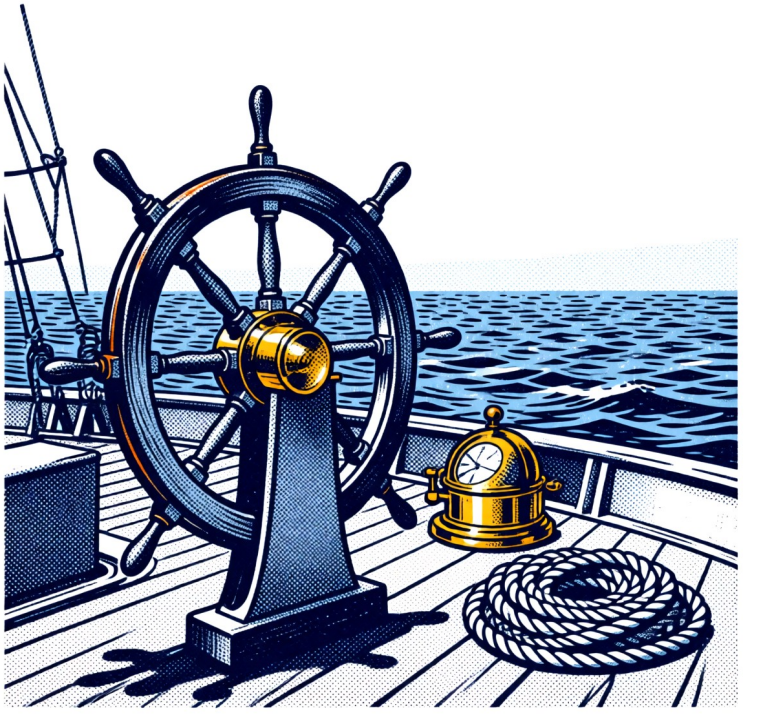
Deux outils. `apt-file` cherche quel paquet *contient* un fichier, même s'il n'est pas installé : `apt-file search bin/dig`. `needrestart`, présent sur Ubuntu Server, signale après une mise à jour les services à redémarrer.

Lecture. Le *Debian Administrator's Handbook*, de Raphaël Hertzog et Roland Mas, disponible gratuitement en français sous le titre *Le cahier de l'administrateur Debian* sur `debian-handbook.info`. Son chapitre sur APT vaut aussi pour Ubuntu.

PARTIE III

Maîtrise du shell Bash

Passer du statut d'utilisateur du shell à celui d'auteur de scripts qui rendent vraiment service.





CHAPITRE 8

Le shell en profondeur

Entre le moment où vous appuyez sur Entrée et celui où la commande démarre, Bash transforme votre ligne. Savoir comment évite la plupart des surprises.

DANS CE CHAPITRE

- Dire ce qu'est un shell et situer Bash parmi les autres.
- Savoir quels fichiers de configuration Bash lit, et dans quel cas.
- Distinguer variable du shell et variable d'environnement, et utiliser les variables spéciales.
- Comprendre comment Bash trouve une commande grâce à `PATH`.
- Suivre les transformations que subit une ligne de commande avant d'être exécutée.
- Utiliser les guillemets à bon escient.
- Régler l'historique, la complétion et l'invite.
- Garder des sessions ouvertes sur un serveur avec `tmux`.

> **AVANT DE COMMENCER**
>
> Les chapitres 5 et 6. Ce chapitre explique ce que vous faites depuis le chapitre 3 sans y penser. Tapez les exemples : le comportement des guillemets se comprend en l'observant.

8.1 Qu'est-ce qu'un shell ?

Un shell est un programme qui répète sans fin la même boucle : afficher une invite, lire une ligne, la transformer, lancer la commande demandée, attendre qu'elle se termine, puis recommencer. Son nom, qui signifie « coquille », vient de ce qu'il enveloppe le noyau : c'est par lui que l'utilisateur atteint le système.

Il en existe plusieurs.

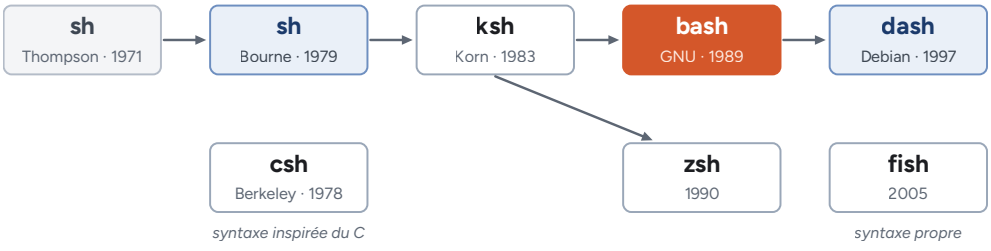


Figure 8.1 — Les principaux shells et leur filiation.

Shell	Sur Ubuntu
`bash`	Le shell des sessions interactives. Celui de ce livre.
`dash`	Un shell minimal et rapide. `/bin/sh` pointe sur lui : il exécute les scripts du système.
`zsh`	Une alternative très appréciée pour l'usage interactif. À installer.
`fish`	Un shell convivial, dont la syntaxe diffère de celle de Bash. À installer.

> **REPÈRE HISTORIQUE · Bourne Again Shell**
>
> En 1979, la septième édition d'Unix apporte un nouveau shell, écrit par Stephen Bourne aux laboratoires Bell. Le *Bourne shell*, nommé `sh`, est resté la référence pendant vingt ans. En 1989, Brian Fox écrit pour le projet GNU un remplaçant libre et le baptise *Bourne Again Shell*, jeu de mots sur *born again*, « né de nouveau ». Bash reprend la syntaxe de `sh` et y ajoute l'historique, la complétion, les tableaux et le calcul.

```
gary@vostro:~$ echo $SHELL
/bin/bash
gary@vostro:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 mars  31  2024 /bin/sh -> dash
```

ATTENTION · sh n'est pas bash

Sur Ubuntu, un script qui commence par `#!/bin/sh` est exécuté par `dash`, qui ne connaît pas les extensions de Bash : tableaux, doubles crochets `[[ ]]`, et quelques autres. Un script écrit pour Bash doit commencer par `#!/bin/bash`. Le chapitre 10 y revient.

8.2 Les fichiers de configuration

Bash lit des fichiers au démarrage, et ce ne sont pas les mêmes selon la façon dont il est lancé. C'est une source de confusion inépuisable. Tout repose sur deux notions.

Un **shell de connexion** est le premier shell d'une session. On l'obtient en se connectant par SSH, sur une console, ou avec `su -` et `sudo -i`.

Un **shell interactif ordinaire** est celui qu'ouvre chaque fenêtre ou onglet de terminal du bureau, ou la commande `bash` tapée dans un shell.

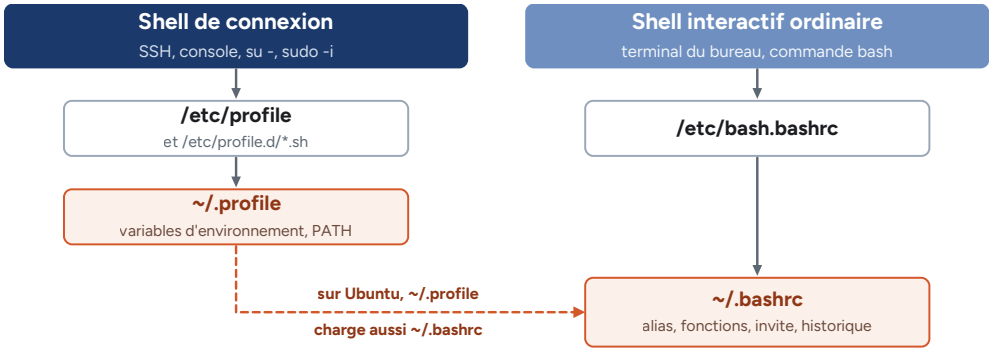


Figure 8.2 — Les fichiers lus par Bash selon la façon dont il démarre.

Fichier	Lu par	On y met
`/etc/profile`, `/etc/profile.d/*.sh`	Les shells de connexion, pour tous les utilisateurs	Les réglages communs à toute la machine

Fichier	Lu par	On y met
`~/.profile`	Votre shell de connexion	Les variables d'environnement, `PATH`
`/etc/bash.bashrc`	Les shells interactifs, pour tous les utilisateurs	Rarement modifié
`~/.bashrc`	Vos shells interactifs	Alias, fonctions, invite, historique
`~/.bash_logout`	Votre shell de connexion, à sa fermeture	Presque jamais utilisé

Ubuntu simplifie les choses par une précaution. Le fichier `~/.profile` fourni à chaque compte contient ces lignes :

```
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi
```

Un shell de connexion charge donc aussi `~/.bashrc`. Que vous arriviez par SSH ou par un terminal du bureau, vos alias et votre invite sont là.

> **À RETENIR · Où écrire quoi**
>
> Les **variables d'environnement** vont dans `~/.profile`. Il est lu une fois, à l'ouverture de la session, y compris la session graphique : toutes vos applications en héritent, même celles lancées depuis le menu.
>
> Les **alias, les fonctions et l'invite** vont dans `~/.bashrc`. Ils ne concernent que le shell interactif, et les alias ne se transmettent pas d'un shell à l'autre.
>
> Attention : si vous créez un fichier `~/.bash_profile`, Bash le lit *à la place* de `~/.profile`, qui est alors ignoré.

Après avoir modifié `~/.bashrc`, rechargez-le dans le shell courant avec `source ~/.bashrc`. Pour `~/.profile`, fermez la session et rouvrez-la.

Un script, lui, ne lit aucun de ces fichiers. Il ne connaît ni vos alias ni vos fonctions. Il hérite seulement des variables d'environnement.

8.3 Les variables

Variables du shell et variables d'environnement

Une variable se crée par une simple affectation, sans espace autour du signe égal. Elle n'existe que dans le shell où elle a été créée.

```
gary@vostro:~$ ville="Douala"
gary@vostro:~$ echo $ville
Douala
gary@vostro:~$ bash -c 'echo "Ville : $ville"'
Ville :
```

Le second shell, lancé par `bash -c`, ne la connaît pas. Pour qu'une variable soit transmise aux programmes lancés depuis le shell, il faut l'*exporter*. Elle devient alors une variable d'environnement.

```
gary@vostro:~$ export ville
gary@vostro:~$ bash -c 'echo "Ville : $ville"'
Ville : Douala
```

La transmission ne va que dans un sens, du parent vers l'enfant. Un programme ne peut pas modifier les variables du shell qui l'a lancé. C'est pourquoi un script qui fait `cd` ou `export` ne change rien dans votre terminal.

```
env                 # les variables d'environnement
set | less          # toutes les variables et fonctions du shell
unset ville         # supprimer une variable
```

Variable	Rôle
`HOME`	Votre dossier personnel
`USER`	Votre nom d'utilisateur
`SHELL`	Votre shell de connexion
`PATH`	Les dossiers où chercher les commandes
`PWD`, `OLDPWD`	Le dossier courant, le dossier précédent
`LANG`, `LC_*`	La langue et les formats de date, de nombre, de tri
`EDITOR`	L'éditeur lancé par `crontab -e`, `git commit`, `visudo`
`TERM`	Le type de terminal

Variable	Rôle
`TZ`	Le fuseau horaire, si l'on veut en changer pour un programme

Les variables spéciales

Bash tient à jour des variables que l'on ne peut que lire.

Variable	Contenu
`$?`	Code de retour de la dernière commande : 0 si elle a réussi
`$$`	Numéro de processus du shell
`$!`	Numéro du dernier processus lancé en arrière-plan
`$0`	Nom du script, ou du shell
`$1`, `$2`…	Arguments reçus par le script ; au-delà de 9, on écrit `${10}`
`$#`	Nombre d'arguments
`"$@"`	Tous les arguments, chacun gardé entier
`$_`	Dernier argument de la commande précédente
`$RANDOM`	Un nombre au hasard entre 0 et 32767

La plus utilisée est `$?`. Toute commande renvoie en se terminant un *code de retour* : 0 si tout s'est bien passé, un autre nombre sinon.

```
gary@vostro:~$ ls /nexiste
ls: impossible d'accéder à '/nexiste': Aucun fichier ou dossier de ce nom
gary@vostro:~$ echo $?
2
gary@vostro:~$ ls / > /dev/null
gary@vostro:~$ echo $?
0
```

Ce code permet d'enchaîner des commandes sous condition :

```
sudo apt update && sudo apt upgrade      # la seconde seulement si la première réussit
ping -c 1 serveur || echo "Injoignable"  # la seconde seulement si la première échoue
```

8.4 PATH : comment Bash trouve une commande

Quand vous tapez `ls`, Bash cherche un programme de ce nom dans les dossiers de la variable `PATH`, de gauche à droite. Il lance le premier qu'il trouve.

```
gary@vostro:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
usr/local/games:/snap/bin
gary@vostro:~$ which -a ls
/usr/bin/ls
/bin/ls
```

`which -a` montre toutes les correspondances, dans l'ordre. La première gagne. Ici, il s'agit du même fichier, puisque `/bin` est un lien vers `/usr/bin`.

L'ordre a des conséquences. `/usr/local/bin` vient avant `/usr/bin` : un programme que vous y installez prend le pas sur celui du système qui porte le même nom. C'est voulu.

```
export PATH="$HOME/outils:$PATH"     # ajouter un dossier en tête : il sera prioritaire
export PATH="$PATH:$HOME/outils"     # l'ajouter en queue
```

DANGER · N'oubliez pas $PATH dans la nouvelle valeur

`export PATH="$HOME/outils"` remplace tout le contenu de la variable. Plus aucune commande n'est trouvée, pas même `ls`. Si cela vous arrive, fermez le terminal et ouvrez-en un autre. Si l'erreur est écrite dans `~/.profile`, corrigez le fichier en appelant l'éditeur par son chemin complet : `/usr/bin/nano ~/.profile`.

Sur Ubuntu, vous n'avez presque jamais à modifier `PATH`. Comme on l'a vu au chapitre 5, les dossiers `~/bin` et `~/.local/bin` y sont ajoutés automatiquement s'ils existent.

Bash garde en mémoire l'emplacement des commandes déjà utilisées. Si vous déplacez un programme et que Bash s'obstine à le chercher à l'ancien endroit, `hash -r` vide cette mémoire.

Le dossier courant ne figure pas dans `PATH`. Pour lancer un script qui s'y trouve, il faut donc écrire `./script.sh`. C'est une mesure de sécurité : sans elle, un fichier nommé `ls` déposé dans `/tmp` serait exécuté à la place de la vraie commande.

8.5 De la ligne tapée à la commande exécutée

Bash ne transmet pas votre ligne telle quelle. Il la transforme d'abord, par étapes, et la commande ne reçoit que le résultat. Presque tous les comportements étranges du shell s'expliquent par ces transformations, qu'on appelle des *expansions*.

```
cp ~/notes/*.txt /tmp/sauve-$(date +%F)/chap{1,2}
```

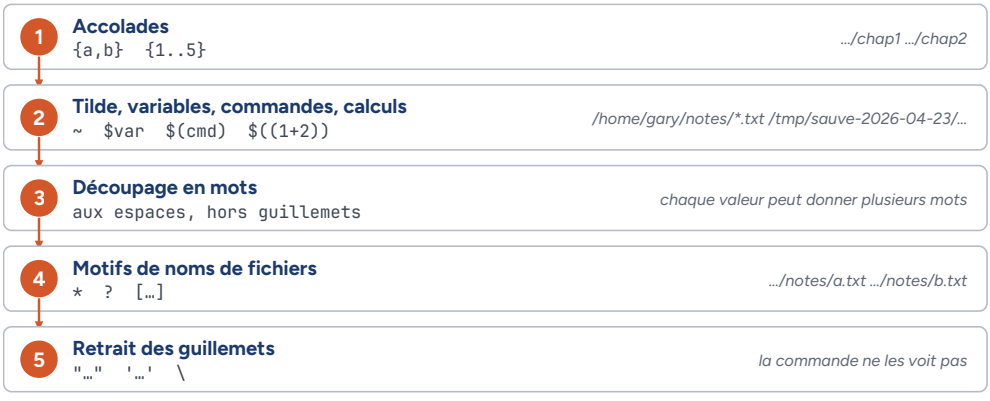


Figure 8.3 — Les transformations d'une ligne de commande, dans l'ordre où Bash les applique.

Les accolades

Les accolades fabriquent des mots. Elles ne regardent pas ce qui existe sur le disque.

```
gary@vostro:~$ echo {1..10..2}
1 3 5 7 9
gary@vostro:~$ echo chap-{01..03}.{md,txt}
chap-01.md chap-01.txt chap-02.md chap-02.txt chap-03.md chap-03.txt
```

Une application que vous utiliserez souvent : `cp nginx.conf{,.orig}` se développe en `cp nginx.conf nginx.conf.orig`.

Le tilde

`~` devient votre dossier personnel, `~alice` celui d'alice.

Les variables

`$nom` est remplacé par la valeur de la variable. La forme `${nom}` fait la même chose, et elle est nécessaire quand du texte suit immédiatement : `${fichier}_copie`.

Entre les accolades, Bash offre bien plus : un petit langage pour retoucher la valeur au passage.

```
NAME="Gary"

# Forme simple
echo $NAME                # Gary
echo ${NAME}              # Gary (accolades recommandées si ambiguïté)

# Longueur de la chaîne
echo ${#NAME}             # 4

# Sous-chaîne (offset, longueur)
echo ${NAME:1:2}          # ar
echo ${NAME:1}            # ary (jusqu'à la fin)

# Valeur par défaut si vide / non définie
echo ${EMPTY:-default}    # default  (si EMPTY n'est pas défini)
echo ${EMPTY:=default}    # default ET affecte EMPTY

# Substitution
FILE="document.txt"
echo ${FILE/.txt/.md}     # document.md — remplace la première occurrence
echo ${FILE//t/T}         # documenT.TxT — remplace toutes

# Suppression de préfixe/suffixe
PATH_FILE="/home/gary/doc.txt"
echo ${PATH_FILE#/home/}  # gary/doc.txt  — enlève le préfixe le plus court
echo ${PATH_FILE##*/}     # doc.txt       — enlève le préfixe le plus long
echo ${PATH_FILE%.txt}    # /home/gary/doc — enlève le suffixe le plus court
echo ${PATH_FILE%%/*}     #                — enlève le plus long depuis /

# Majuscules / minuscules
echo ${NAME^^}            # GARY
echo ${NAME,,}            # gary

# Valeur avec erreur si variable vide
echo ${REQUIRED:?variable REQUIRED non définie}
```

Quatre de ces écritures reviennent sans cesse dans les scripts :

Écriture	Effet	Exemple avec `f=/home/gary/doc.txt`
`${var:-défaut}`	La valeur, ou `défaut` si la variable est vide	
`${var##*/}`	Retire tout jusqu'à la dernière barre oblique	`doc.txt`
`${var%/*}`	Retire la dernière barre oblique et ce qui suit	`/home/gary`
`${var%.txt}`	Retire le suffixe `.txt`	`/home/gary/doc`

Pour s'en souvenir, regardez le clavier : sur un clavier américain, `#` est à gauche de `$` et retire au début, `%` est à droite et retire à la fin. Doublé, le symbole retire le plus possible.

La substitution de commande

`$(commande)` est remplacé par ce que la commande affiche.

```
gary@vostro:~$ echo "Nous sommes le $(date +%F)"
Nous sommes le 2026-04-23
gary@vostro:~$ mkdir "sauvegarde-$(date +%F)"
```

L'ancienne écriture, entre accents graves, fait la même chose. Elle se lit mal et ne s'imbrique pas. Évitez-la.

Le calcul

`$(( ))` calcule sur des nombres entiers.

```
gary@vostro:~$ echo $((7 / 2)) $((7 % 2)) $((2 ** 10))
3 1 1024
```

La division est entière : 7 divisé par 2 donne 3. Pour des nombres à virgule, on passe par un autre outil : `echo "scale=2; 7/2" | bc` affiche 3.50.

Le découpage en mots

Une fois les variables et les commandes remplacées par leur valeur, Bash découpe le résultat en mots, à chaque espace. C'est l'étape qui cause le plus d'erreurs.

```
gary@vostro:~$ fichier="fichier 1.txt"
gary@vostro:~$ ls $fichier
ls: impossible d'accéder à 'fichier': Aucun fichier ou dossier de ce nom
ls: impossible d'accéder à '1.txt': Aucun fichier ou dossier de ce nom
gary@vostro:~$ ls "$fichier"
fichier 1.txt
```

Sans guillemets, la valeur a été coupée en deux, et `ls` a reçu deux arguments. Entre guillemets doubles, elle reste entière.

> **À RETENIR · Toujours des guillemets autour des variables**
>
> Écrivez `"$fichier"`, `"$1"`, `"$(commande)"`. Toujours. Un nom de fichier peut contenir un espace, et ceux qui viennent d'un téléchargement ou d'une clé USB en contiennent souvent. L'outil `shellcheck`, présenté au chapitre 10, repère les guillemets oubliés.

Les motifs de noms de fichiers

En dernier lieu, Bash remplace `*`, `?` et `[…]` par les noms de fichiers correspondants, comme on l'a vu au chapitre 5. Si rien ne correspond, le motif est laissé tel quel, ce qui produit un message curieux :

```
gary@vostro:~/vide$ ls *.txt
ls: impossible d'accéder à '*.txt': Aucun fichier ou dossier de ce nom
```

`ls` a reçu littéralement `*.txt`, et aucun fichier ne porte ce nom.

8.6 Les guillemets

Les guillemets servent à bloquer tout ou partie de ces transformations.

Écriture	Ce qui est encore transformé	Exemple	Résultat
`"…"`	Les variables, `$(…)` et `$((…))`	`echo "Bonjour $USER"`	`Bonjour gary`
`'…'`	Rien	`echo 'Bonjour $USER'`	`Bonjour $USER`
`\`	Rien, pour le caractère qui suit	`echo Prix : 42 \$`	`Prix : 42 $`

Entre guillemets doubles, le découpage en mots et les motifs sont désactivés, mais les variables sont remplacées. C'est le cas le plus courant.

Entre apostrophes, tout est pris à la lettre. On s'en sert pour les expressions régulières et pour le code passé à d'autres outils : `grep '^root:' /etc/passwd`, `awk '{print $1}'`.

La barre oblique inversée protège un seul caractère. Placée en fin de ligne, elle permet de poursuivre une commande sur la ligne suivante :

```
sudo apt install --no-install-recommends \
    nginx \
    certbot
```

> **NOTE · Une apostrophe entre apostrophes**
>
> On ne peut pas placer d'apostrophe dans une chaîne entre apostrophes. La parade consiste à fermer la chaîne, écrire une apostrophe protégée, puis rouvrir : `echo 'L'\''exemple'` affiche `L'exemple`. Plus simplement, utilisez des guillemets doubles : `echo "L'exemple"`.

8.7 L'historique

Les commandes de base ont été vues au chapitre 5. L'historique se règle par quelques variables, dans `~/.bashrc`. Voici les valeurs d'Ubuntu, et ce que je conseille de mettre à la place :

Variable	Valeur d'Ubuntu	Suggestion	Effet
`HISTSIZE`	1000	10000	Nombre de commandes gardées en mémoire
`HISTFILESIZE`	2000	20000	Nombre de commandes gardées dans `~/.bash_history`
`HISTCONTROL`	`ignoreboth`	`ignoreboth`	Ignore les doublons consécutifs et les lignes qui commencent par une espace
`HISTTIMEFORMAT`	non définie	`"%F %T "`	Enregistre et affiche la date de chaque commande

Avec `HISTTIMEFORMAT`, l'historique devient un journal de bord :

```
gary@serveur:~$ history | tail -n 3
  412  2026-04-23 11:15:32  sudo systemctl restart nginx
  413  2026-04-23 11:15:45  sudo tail /var/log/nginx/error.log
  414  2026-04-23 11:16:02  history | tail -n 3
```

Sur un serveur, savoir à quelle heure une commande a été lancée aide beaucoup à reconstituer un incident.

ASTUCE · Une commande qui ne laisse pas de trace

Avec le réglage d'Ubuntu, une ligne qui commence par une espace n'est pas enregistrée. C'est utile quand une commande contient un mot de passe ou un jeton.

Par défaut, chaque terminal garde ses commandes en mémoire et ne les écrit dans le fichier qu'à sa fermeture. Pour qu'elles soient écrites au fur et à mesure, ajoutez dans `~/.bashrc` :

```
PROMPT_COMMAND="history -a; $PROMPT_COMMAND"
```

8.8 La complétion

Si la touche Tab sait compléter les noms des services après `systemctl restart` ou ceux des machines après `ssh`, c'est grâce au paquet `bash-completion`. Il contient des règles pour près d'un millier de commandes.

Vous pouvez ajouter les vôtres. Supposons un script `deploie`, qui attend le nom d'un environnement :

```
complete -W "dev recette production" deploie
```

Cette ligne, placée dans `~/.bashrc`, suffit : `deploie re` suivi de Tab donne `deploie recette`.

Beaucoup d'outils récents savent produire leurs propres règles. La commande `kubectl completion bash`, par exemple, affiche celles de `kubectl`, que l'on enregistre dans `/etc/bash_completion.d/`.

8.9 L'invite

L'invite est définie par la variable `PS1`. Elle contient du texte et des codes que Bash remplace à chaque affichage.

Code	Affiche
`\u`	Le nom d'utilisateur
`\h`	Le nom de la machine
`\w`, `\W`	Le dossier courant : chemin complet, ou dernier élément seulement
`\t`	L'heure
`\n`	Un retour à la ligne
`\$`	$ pour un utilisateur ordinaire, # pour `root`
`\[ … \]`	Encadrent ce qui n'occupe pas de place à l'écran : les codes de couleur

Les couleurs s'obtiennent par des séquences comme `\e[32m` (vert), `\e[34m` (bleu) et `\e[0m` (retour à la normale). Elles doivent être placées entre `\[` et `\]`. Sans cela, Bash se trompe sur la longueur de l'invite et l'affichage se dérègle sur les longues lignes.

```
PS1='\[\e[32m\]\u@\h\[\e[0m\]:\[\e[34m\]\w\[\e[0m\]\$ '
```

Écrivez cette valeur entre apostrophes. Entre guillemets doubles, `\$` deviendrait un simple `$`, et l'invite ne signalerait plus `root` par un `#`.

Deux améliorations utiles :

```
# Le dossier sur une ligne, la saisie sur la suivante : confortable avec de longs chemins
PS1='\[\e[32m\]\u@\h\[\e[0m\]:\[\e[34m\]\w\[\e[0m\]\n\$ '

# La branche Git du dossier courant, grâce à une fonction fournie avec Git
PS1='\u@\h:\w$(__git_ps1 " (%s)")\$ '
```

> **ASTUCE · Une invite rouge pour root sur les serveurs**
>
> Sur un serveur de production, donnez à `root` une invite d'une couleur qui se remarque, dans `/root/.bashrc`. Elle rappelle à chaque ligne que toute erreur aura des conséquences.

8.10 tmux

Quand on travaille sur un serveur par SSH, tout ce qui tourne dans le terminal dépend de la connexion. Que le Wi-Fi décroche ou que le portable se mette en veille, et la commande en cours est interrompue.

`tmux` règle ce problème. Il crée sur le serveur une session qui vit indépendamment de la connexion. On s'en détache, on s'y rattache plus tard, et l'on retrouve tout dans l'état où on l'avait laissé. Il permet aussi d'avoir plusieurs terminaux dans une même fenêtre.

```
sudo apt install tmux

# Lancer une nouvelle session
tmux

# Lancer une session nommée
tmux new -s dev

# Lister les sessions
tmux ls

# Attacher à une session existante
tmux attach -t dev
tmux a -t dev            # raccourci

# Se détacher (laisse la session tourner)
# Dans tmux : Ctrl+b puis d

# Tuer une session
tmux kill-session -t dev
```

Toutes les commandes de tmux se donnent en deux temps : d'abord Ctrl+B, que l'on relâche, puis une touche.

Raccourci	Effet
Ctrl+B puis D	Se détacher : la session continue de tourner
Ctrl+B puis C	Créer une fenêtre
Ctrl+B puis N ou P	Fenêtre suivante, précédente
Ctrl+B puis 0 à 9	Aller à la fenêtre de ce numéro
Ctrl+B puis ,	Renommer la fenêtre
Ctrl+B puis %	Partager l'écran en deux panneaux côte à côte

Raccourci	Effet
Ctrl+B puis "	Partager l'écran en deux panneaux superposés
Ctrl+B puis une flèche	Passer d'un panneau à l'autre
Ctrl+B puis Z	Agrandir le panneau courant, puis le rétablir
Ctrl+B puis X	Fermer le panneau
Ctrl+B puis [	Remonter dans l'affichage ; q pour sortir
Ctrl+B puis ?	Afficher tous les raccourcis

Un exemple type. Vous devez lancer une restauration de base de données qui prendra deux heures :

```
gary@vostro:~$ ssh gary@serveur
gary@serveur:~$ tmux new -s restauration
gary@serveur:~$ ./restaurer-base.sh
```

Tapez Ctrl+B puis D, fermez la connexion, éteignez votre portable. Le lendemain :

```
gary@vostro:~$ ssh gary@serveur
gary@serveur:~$ tmux attach -t restauration
```

La restauration a continué sans vous, et tout ce qu'elle a affiché est là.

> **À RETENIR · Sur un serveur, travaillez dans tmux**
>
> Prenez l'habitude de lancer `tmux` dès la connexion, avant toute opération longue : mise à jour, sauvegarde, migration. Le jour où la connexion tombera en plein milieu, vous serez content de l'avoir fait.

Quelques réglages rendent tmux plus agréable. Ils se placent dans `~/.tmux.conf` :

```
# ~/.tmux.conf

# Remapper le préfixe sur Ctrl+a (plus ergonomique)
unbind C-b
set -g prefix C-a
bind C-a send-prefix

# Des touches plus parlantes pour partager l'écran
bind | split-window -h
bind - split-window -v
```

```
# Rechargement rapide de la config
bind r source-file ~/.tmux.conf \; display "Rechargé !"

# Numéroter les fenêtres à partir de 1 (plus facile à atteindre)
set -g base-index 1
setw -g pane-base-index 1

# Mode souris
set -g mouse on

# Augmenter l'historique
set -g history-limit 10000

# Couleurs
set -g default-terminal "tmux-256color"
```

`screen` est un outil plus ancien qui rend le même service. Vous le trouverez sur de vieux serveurs : `screen -S nom` crée une session, Ctrl+A puis D s'en détache, `screen -r nom` s'y rattache.

En résumé

- Le shell lit une ligne, la transforme, lance la commande et attend. Sur Ubuntu, `bash` sert aux sessions interactives et `dash` exécute les scripts en `#!/bin/sh`.
- `~/.profile` est lu à l'ouverture de la session : on y met les variables d'environnement. `~/.bashrc` est lu par chaque shell interactif : on y met alias, fonctions et invite.
- Une variable n'est transmise aux programmes que si elle est exportée, et jamais de l'enfant vers le parent.
- `$?` contient le code de retour de la dernière commande : 0 signifie succès.
- Bash cherche les commandes dans les dossiers de `PATH`, de gauche à droite.
- Avant d'exécuter une ligne, Bash développe les accolades, puis le tilde, les variables, les commandes et les calculs, puis découpe en mots, puis développe les motifs.
- Les guillemets doubles protègent des découpages, les apostrophes de tout. On met toujours les variables entre guillemets doubles.
- `tmux` garde une session ouverte sur le serveur, même quand la connexion tombe.

Atelier — Aménager son shell

Objectif : construire un `~/.bashrc` à votre main, puis prendre tmux en main. Comptez quarante-cinq minutes.

Avant tout

1. Faites une copie : `cp ~/.bashrc{,.orig}`. Que fait cette écriture ?

L'historique

2. À la fin de `~/.bashrc`, portez `HISTSIZE` à 10000 et `HISTFILESIZE` à 20000, et définissez `HISTTIMEFORMAT`.
3. Rechargez le fichier et vérifiez avec `history | tail`.

Alias et fonctions

4. Ajoutez ces lignes, puis rechargez :

```
alias ll='ls -lah'
alias ..='cd ..'
alias maj='sudo apt update && sudo apt upgrade'

# Créer un dossier et y entrer
mkcd() { mkdir -p "$1" && cd "$1"; }

# Faire une copie datée d'un fichier
copie() { cp -a "$1" "$1.$(date +%F-%H%M)"; }
```

5. Essayez `mkcd /tmp/essai/a/b`, puis créez un fichier et lancez `copie` dessus.
6. Pourquoi `mkcd` doit-elle être une fonction, et non un script ?

L'invite

7. Définissez une invite en couleurs sur deux lignes, comme en section 8.9.
8. Installez Git si besoin, placez-vous dans un dépôt et ajoutez la branche à l'invite.

tmux

9. Installez tmux et créez une session nommée `atelier`.
10. Partagez l'écran en deux panneaux. Lancez `htop` dans l'un et `tail -f /var/log/syslog` dans l'autre.
11. Détachez-vous, vérifiez avec `tmux ls`, fermez le terminal, ouvrez-en un autre et rattachez-vous.

Exercices

Exercice 1 — Quels fichiers sont lus ?

Dans chaque cas, dites quels fichiers de configuration Bash lit.

1. Vous vous connectez à un serveur par SSH.
2. Vous ouvrez un onglet dans le terminal du bureau.
3. Vous lancez `sudo -i`.
4. Vous exécutez `./script.sh`.

Exercice 2 — Que reçoit la commande ?

Le dossier courant est `/home/gary/projets`. Il contient `a.txt`, `b.md` et `c.log`. Que reçoit `echo` dans chaque cas ?

```
echo {1..3}-$USER.log
echo ~/$(basename "$PWD")/*
echo *.{txt,md}
echo "*.txt"
echo '$USER est dans $(pwd)'
```

Exercice 3 — Les guillemets

```
fichier="mon rapport.txt"
```

1. Combien d'arguments `ls` reçoit-il avec `ls $fichier` ? Et avec `ls "$fichier"` ? Et avec `ls '$fichier'` ?
2. Donnez trois façons d'afficher exactement `Le prix est de 42 $`.

Exercice 4 — PATH

1. Créez un script `~/bin/bonjour` qui affiche un message, et rendez-le exécutable. Est-il trouvé tout de suite ? Sinon, pourquoi ?
2. Créez un script `~/bin/ls` qui affiche « intercepté ». Que se passe-t-il quand vous tapez `ls` ? Comment lancer malgré tout la vraie commande ?
3. Supprimez ce script. Pourquoi `ls` répond-il encore « intercepté », ou une erreur ? Que faire ?

Exercice 5 — Variables

```
a=1
bash -c 'echo "a vaut $a"'
export a
bash -c 'a=2; echo "a vaut $a"'
echo "a vaut $a"
```

Qu'affichent ces trois commandes `echo` ? Expliquez la dernière.

Pour aller plus loin

Références. `man bash` est long et dense, mais c'est la source. Sa section EXPANSION décrit dans l'ordre tout ce que la section 8.5 résume. `mywiki.wooledge.org/BashFAQ` répond avec rigueur aux questions qui reviennent le plus souvent, et `mywiki.wooledge.org/BashPitfalls` recense les erreurs classiques.

Outils. `fzf` remplace la recherche de Ctrl+R par une recherche floue bien plus confortable. `starship.rs` fournit une invite toute faite et très complète. Le chapitre 33 les installe dans un poste de travail.

Lectures. *Learning the bash Shell*, de Cameron Newham (O'Reilly), pour une progression en douceur. *bash Cookbook*, de Carl Albing et JP Vossen (O'Reilly), pour des solutions à des problèmes précis.



CHAPITRE 9

Traiter du texte en ligne de commande

Chercher, découper, remplacer, compter. Avec grep, sed, awk et quelques autres, un journal de plusieurs gigaoctets se laisse interroger en une ligne.

DANS CE CHAPITRE

- Chercher dans des fichiers avec `grep`.
- Écrire des expressions régulières simples et savoir quel outil comprend quelle écriture.
- Transformer du texte avec `sed`.
- Exploiter des données en colonnes avec `awk`.
- Découper, trier, dédoublonner et comparer avec `cut`, `sort`, `uniq`, `join`, `diff`.
- Convertir un encodage ou des fins de ligne.
- Lire du JSON avec `jq` et du YAML avec `yq`.

> **AVANT DE COMMENCER**
>
> Les chapitres 5 et 8 : les tubes, les redirections et les guillemets. Tous les exemples du chapitre ont été exécutés. Refaites-les : ces outils s'apprennent par l'essai.

Sous Linux, presque tout est du texte : la configuration, les journaux, la sortie des commandes. Les outils de ce chapitre ont été écrits dans les années 1970 pour l'exploiter, et personne n'a fait mieux depuis pour fouiller un fichier trop gros pour un éditeur.

9.1 grep : chercher

`grep` affiche les lignes qui contiennent un motif.

```
grep "error" /var/log/syslog          # dans un fichier
grep "error" /var/log/*.log           # dans plusieurs
sudo dmesg | grep -i usb              # dans la sortie d'une commande
grep -r "TODO" ~/projets/             # dans toute une arborescence
```

Option	Effet
`-i`	Ne distingue pas majuscules et minuscules
`-v`	Affiche les lignes qui ne contiennent *pas* le motif
`-r`	Parcourt les sous-dossiers
`-n`	Affiche le numéro de chaque ligne
`-l`	N'affiche que le nom des fichiers concernés
`-c`	Compte les lignes au lieu de les afficher
`-o`	N'affiche que la partie de la ligne qui correspond
`-w`	Ne retient que les mots entiers
`-A 3`, `-B 3`, `-C 3`	Affiche aussi 3 lignes après, avant, autour
`-E`	Active les expressions régulières étendues
`-F`	Prend le motif à la lettre, sans expression régulière

```
# Insensible à la casse, lignes numérotées
grep -in "warning" /var/log/syslog
```

```
# Compter les erreurs 500 dans un log nginx
grep -c " 500 " /var/log/nginx/access.log

# Les lignes qui ne contiennent pas "debug", sans tenir compte de la casse
grep -vi "debug" /var/log/app.log

# Chercher "error" avec 3 lignes de contexte autour
grep -C 3 "error" /var/log/syslog

# Chercher récursivement dans du code source,
# seulement les .py, excluant venv/
grep -rn --include="*.py" --exclude-dir="venv" "TODO" ~/projet/

# Lister seulement les fichiers qui contiennent le motif
grep -rl "secretkey" ~/projet/

# Plusieurs motifs simultanés
grep -E "error|warning|critical" /var/log/syslog
# équivalent à :
grep -e "error" -e "warning" -e "critical" /var/log/syslog

# Extraire les adresses IP, sans doublon
grep -oP '\b(\d{1,3}\.){3}\d{1,3}\b' /var/log/nginx/access.log | sort -u
```

Deux usages reviennent tous les jours. `grep -v` nettoie une sortie trop bavarde. Et `grep -C 3` montre ce qui s'est passé juste avant et juste après une erreur, ce qui en donne souvent la cause.

> **REPÈRE HISTORIQUE · D'où vient ce nom ?**
>
> Dans `ed`, l'éditeur de texte des premiers Unix, la commande `g/re/p` affichait toutes les lignes contenant une expression régulière : *global*, *regular expression*, *print*. Ken Thompson en a fait un programme à part.

9.2 Les expressions régulières

Une expression régulière décrit une forme de texte : « une ligne qui commence par un chiffre », « quatre nombres séparés par des points ». C'est un petit langage, que partagent `grep`, `sed`, `awk` et la plupart des langages de programmation.

Les symboles à connaître

Symbole	Signification	Exemple	Correspond à
`.`	Un caractère quelconque	`p.rt`	`port`, `part`

Symbole	Signification	Exemple	Correspond à
`^`	Le début de la ligne	`^root`	les lignes qui commencent par `root`
`$`	La fin de la ligne	`bash$`	les lignes qui finissent par `bash`
`*`	Zéro, une ou plusieurs fois ce qui précède	`ab*c`	`ac`, `abc`, `abbc`
`+`	Une ou plusieurs fois	`[0-9]+`	`7`, `42`, `2026`
`?`	Zéro ou une fois	`https?`	`http`, `https`
`{2,4}`	De deux à quatre fois	`[0-9]{2,4}`	`12`, `123`, `1234`
`[abc]`	Un caractère parmi ceux-là	`[Ee]rreur`	`Erreur`, `erreur`
`[^abc]`	Un caractère qui n'est aucun de ceux-là	`[^0-9]`	tout sauf un chiffre
`a|b`	L'un ou l'autre	`error|warning`	`error`, `warning`
`( )`	Regroupe	`(ab)+`	`ab`, `abab`
`\.`	Un point, au sens propre	`192\.168`	`192.168`

L'étoile n'a pas le même sens que dans les motifs du shell. Ici, elle répète ce qui la précède. « N'importe quelle suite de caractères » s'écrit donc `.*`.

Trois dialectes

Tous les outils ne comprennent pas exactement les mêmes écritures.

Dialecte	Outils	Particularité
De base	`grep`, `sed`	`+`, `?`, `{}`, `|` et `()` doivent être précédés d'une barre oblique inversée
Étendu	`grep -E`, `sed -E`, `awk`	Ces symboles s'écrivent tels quels
Perl	`grep -P`	Ajoute des raccourcis comme `\d` pour un chiffre

Le conseil est simple : utilisez toujours le dialecte étendu, avec `grep -E` et `sed -E`. C'est celui du tableau ci-dessus, et celui de ce livre.

> **ATTENTION · \d ne fonctionne pas avec grep -E**
>
> Le raccourci `\d` vient de Perl. `grep -E '\d+'` ne trouve rien, sans le moindre message d'erreur. Écrivez `[0-9]+`, ou passez à `grep -P`. En revanche, `\w` (un caractère de mot) et `\b` (une limite de mot) sont acceptés par `grep -E`.

```
# Lignes qui commencent par un chiffre
grep -E "^[0-9]" fichier.txt

# Lignes qui contiennent exactement 5 chiffres consécutifs
grep -E "[0-9]{5}" fichier.txt

# Adresses électroniques (forme simplifiée)
grep -E "[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" fichier.txt

# Adresses IPv4 (forme simplifiée : accepte aussi 999.999.999.999)
grep -E "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" log.txt

# URLs HTTP
grep -oE "https?://[^ ]+" fichier.txt

# Dates au format YYYY-MM-DD
grep -E "[0-9]{4}-[0-9]{2}-[0-9]{2}" fichier.txt

# Lignes vides
grep -E "^$" fichier.txt

# Lignes qui NE sont PAS des commentaires (# en début)
grep -vE "^#" config.conf

# Enlever les lignes vides ET les commentaires
grep -vE "^(#|$)" config.conf
```

La dernière ligne est à retenir : `grep -vE '^(#|$)' fichier.conf` affiche un fichier de configuration sans ses commentaires ni ses lignes vides. Il ne reste que ce qui est réellement actif.

> **ASTUCE · Mettre au point une expression**
>
> Le site `regex101.com` montre en direct ce qu'une expression capture dans un texte, et explique chacun de ses éléments. Choisissez-y le dialecte qui correspond à votre outil.

9.3 sed : transformer

`sed` lit du texte ligne par ligne, applique des commandes et écrit le résultat. On l'emploie neuf fois sur dix pour remplacer.

Remplacer

```
gary@vostro:~$ echo "bonjour bonjour" | sed 's/bonjour/salut/'
salut bonjour
gary@vostro:~$ echo "bonjour bonjour" | sed 's/bonjour/salut/g'
salut salut
```

La commande `s/ancien/nouveau/` remplace la première occurrence de chaque ligne. Le `g` final les remplace toutes.

Quand le texte contient des barres obliques, on change de séparateur. N'importe quel caractère convient :

```
sed 's|/var/www/html|/srv/site|g' nginx.conf
```

Avec `-E`, les parenthèses capturent des morceaux, que l'on réutilise par `\1`, `\2` :

```
gary@vostro:~$ echo "Gary Nkweka" | sed -E 's/(\w+) (\w+)/\2 \1/'
Nkweka Gary
```

Modifier le fichier lui-même

Par défaut, `sed` affiche le résultat et ne touche pas au fichier. L'option `-i` le modifie sur place.

```
sed -i 's/ancien/nouveau/g' fichier.conf          # sans filet
sed -i.orig 's/ancien/nouveau/g' fichier.conf     # en gardant l'original
dans fichier.conf.orig
```

> **À RETENIR · Essayez d'abord sans -i**
>
> Lancez la commande sans `-i` et lisez le résultat à l'écran. Ajoutez `-i` seulement quand il vous convient, et de préférence `-i.orig`. Une substitution trop large sur un fichier de configuration se remarque rarement tout de suite.

Choisir les lignes

Une adresse, placée devant la commande, limite son effet à certaines lignes.

```
# Agir seulement sur la ligne 5
sed '5s/ancien/nouveau/' fichier.txt

# Lignes 10 à 20
sed '10,20s/ancien/nouveau/g' fichier.txt

# De la ligne 5 à la fin
sed '5,$s/ancien/nouveau/g' fichier.txt

# Sur les lignes qui correspondent à un motif
sed '/^ERROR/s/ancien/nouveau/' log.txt

# Entre deux motifs
sed '/<<DEBUT>>/,/<<FIN>>/s/ancien/nouveau/g' fichier.txt
```

Supprimer, afficher, insérer

```
# d : supprimer des lignes
sed '/^#/d' config.conf                    # supprime les commentaires
sed '1,5d' fichier.txt                     # supprime les 5 premières lignes
sed '$d' fichier.txt                       # supprime la dernière ligne
sed '/^$/d' fichier.txt                    # supprime les lignes vides

# p : imprimer (usage : avec -n pour supprimer la sortie par défaut)
sed -n '/ERROR/p' log.txt                  # équivalent de grep ERROR
sed -n '10,20p' fichier.txt                # afficher lignes 10 à 20

# a : ajouter une ligne APRÈS
sed '/^Host/a\  Port 22' ~/.ssh/config

# i : insérer AVANT
sed '/^hosts:/i\# Modified by deploy script' /etc/nsswitch.conf

# c : remplacer la ligne entière
sed '/^timeout=/c\timeout=30' config.conf
```

`sed -n '10,20p' fichier` affiche les lignes 10 à 20. C'est bien pratique quand un message d'erreur donne un numéro de ligne.

9.4 awk : exploiter des colonnes

`awk` doit son nom à ses trois auteurs, Aho, Weinberger et Kernighan, qui l'ont écrit en 1977. C'est un vrai langage de programmation, spécialisé dans les données rangées en colonnes. Dès qu'un texte se découpe en champs, c'est l'outil qu'il faut.



Figure 9.1 — Comment awk découpe une ligne.

Le principe

`awk` lit chaque ligne et la découpe en champs, aux espaces. `$1` est le premier champ, `$2` le deuxième, `$NF` le dernier, et `$0` la ligne entière. Un programme a la forme `condition { action }`. L'action est exécutée pour chaque ligne qui satisfait la condition.

Prenons un fichier `gens.txt` :

```
alice 28 Douala
bob 35 Yaounde
claire 41 Douala
```

```
gary@vostro:~$ awk '{print $1, $NF}' gens.txt
alice Douala
bob Yaounde
claire Douala
gary@vostro:~$ awk '$2 > 30 {print $1}' gens.txt
bob
claire
```

Sans condition, l'action s'applique à toutes les lignes. Sans action, `awk` affiche les lignes retenues.

```
# Lignes dont le 3e champ > 100
awk '$3 > 100' fichier.txt

# Lignes où le 2e champ est "error"
```

```
awk '$2 == "error"' fichier.txt

# Lignes qui contiennent un motif
awk '/ERROR/' fichier.txt

# Motif appliqué à un champ précis
awk '$1 ~ /^admin/' fichier.txt

# Le champ ne correspond pas au motif
awk '$1 !~ /^guest/' fichier.txt

# Plusieurs conditions
awk '$3 > 100 && $4 == "active"' fichier.txt
```

Changer de séparateur

L'option `-F` indique le séparateur des champs. Avec `-F:`, on lit `/etc/passwd` :

```
gary@vostro:~$ awk -F: '$3 >= 1000 && $3 < 60000 {print $1, $7}' /etc/passwd
gary /bin/bash
alice /bin/bash
```

Voilà la liste des comptes de personnes, avec leur shell.

Compter et additionner

`awk` sait calculer, et il dispose de tableaux dont l'indice peut être un texte. Les blocs `BEGIN` et `END` s'exécutent avant la première ligne et après la dernière.

```
gary@vostro:~$ awk '{s += $2} END {print "Somme :", s}' gens.txt
Somme : 104
gary@vostro:~$ awk '{n[$3]++} END {for (v in n) print v, n[v]}' gens.txt
Douala 2
Yaounde 1
```

La seconde commande compte les lignes par ville. `n[$3]++` ajoute 1 à la case du tableau qui porte le nom de la ville. À la fin, on parcourt le tableau.

```
# Somme du 3e champ sur toutes les lignes
awk '{sum += $3} END {print sum}' fichier.txt

# Moyenne
awk '{sum += $3; n++} END {print sum/n}' fichier.txt

# Min et max
awk 'NR==1 {min=max=$3} {if ($3<min) min=$3; if ($3>max) max=$3}
     END {print "min:",min,"max:",max}' fichier.txt
```

```
# Compter les occurrences (comme sort | uniq -c)
awk '{count[$1]++} END {for (k in count) print k, count[k]}' fichier.txt

# Formatage d'en-tête et de total
awk 'BEGIN {print "--- Rapport ---"}
     {print $1,":",$2; total += $2}
     END {print "Total :", total}' fichier.txt
```

NOTE · mawk, gawk et la virgule

La commande `awk` d'Ubuntu est `mawk`, une version rapide et réduite à l'essentiel. La version GNU, `gawk`, offre davantage de fonctions. Elle s'installe à part. Tout ce chapitre fonctionne avec les deux.

Un détail peut surprendre. Sur un système en français, `mawk` affiche les nombres décimaux avec une virgule : 34,6667. Si un autre programme doit relire ces nombres, lancez la commande avec `LC_ALL=C awk …`.

9.5 Découper, trier, compter

cut et paste

```
# Par caractère (octets)
cut -c1-10 fichier.txt              # 10 premiers caractères
cut -c5- fichier.txt                # à partir du 5e

# Par champ (-d délimiteur, -f champs)
cut -d: -f1 /etc/passwd             # 1er champ
cut -d: -f1,3 /etc/passwd           # 1er et 3e
cut -d: -f1-3 /etc/passwd           # de 1 à 3
cut -d: -f1,3-5 /etc/passwd         # 1, et 3 à 5

# Avec CSV (attention : cut est naïf, pas de support des guillemets)
cut -d',' -f2 data.csv
```

`cut` est simple et rapide, mais limité. Il ne sait pas traiter plusieurs espaces de suite comme un seul séparateur. Pour la sortie d'une commande alignée par des espaces, comme celle de `ps` ou de `df`, prenez `awk`.

`paste` fait l'inverse : il met des fichiers côte à côte. `paste -sd, liste.txt` réunit toutes les lignes en une seule, séparées par des virgules.

sort

```
# Tri alphabétique simple
sort fichier.txt

# Tri numérique (sinon "10" < "2")
sort -n fichier.txt

# Tri numérique avec unités (K, M, G)
du -h * | sort -h

# Tri inverse (du plus grand au plus petit)
sort -rn fichier.txt

# Tri par colonne (champ)
sort -k 3 fichier.txt              # trier par 3e colonne
sort -k 3 -n fichier.txt           # par 3e colonne, numériquement
sort -t: -k 3 -n /etc/passwd       # séparateur :, par 3e colonne (UID), num.

# Trier et supprimer les doublons
sort -u fichier.txt

# Mélanger les lignes
shuf fichier.txt
```

Trois points à connaître.

Sans `-n`, le tri compare du texte, et 100 se range avant 9. Avec `-n`, il compare des nombres. Avec `-h`, il comprend les tailles comme `2K` ou `1,5G`.

`-k 3` trie à partir du troisième champ, jusqu'à la fin de la ligne. Pour trier sur le troisième champ seul, on écrit `-k 3,3`.

L'ordre du tri dépend de la langue du système :

```
gary@vostro:~$ printf 'b\nA\na\nB\n' | sort | tr '\n' ' '
a A b B
gary@vostro:~$ printf 'b\nA\na\nB\n' | LC_ALL=C sort | tr '\n' ' '
A B a b
```

En français, majuscules et minuscules sont mêlées. Avec `LC_ALL=C`, le tri suit l'ordre des codes des caractères : il est plus rapide et identique sur toutes les machines. Dans un script, c'est celui qu'il faut.

uniq

`uniq` retire les lignes identiques qui se suivent. Il faut donc trier avant.

```
# Doublons consécutifs supprimés
sort fichier.txt | uniq

# Compter les occurrences (le classique)
sort fichier.txt | uniq -c

# Seulement les lignes dupliquées
sort fichier.txt | uniq -d

# Seulement les lignes uniques (n'apparaissent qu'une fois)
sort fichier.txt | uniq -u

# Insensible à la casse
sort fichier.txt | uniq -i

# Les dix valeurs les plus fréquentes
sort fichier.txt | uniq -c | sort -rn | head -10
```

`sort | uniq -c | sort -rn` est l'enchaînement le plus utile de ce chapitre. Il répond à la question « qu'est-ce qui revient le plus souvent ? ».

join

`join` rapproche deux fichiers triés qui ont un champ en commun, comme une jointure dans une base de données.

```
gary@vostro:~$ cat noms.txt
1001 alice
1002 bob
gary@vostro:~$ cat salaires.txt
1001 2500
1002 3000
gary@vostro:~$ join noms.txt salaires.txt
1001 alice 2500
1002 bob 3000
```

9.6 Comparer

diff

```
# Format par défaut
diff ancien.txt nouveau.txt

# Format unifié : le plus lisible, celui de Git
diff -u ancien.txt nouveau.txt
--- ancien.txt   2026-04-23 10:00:00
```

```
+++ nouveau.txt  2026-04-24 11:30:00
@@ -1,3 +1,3 @@
 ligne inchangée
-ligne supprimée
+ligne ajoutée
 autre ligne inchangée

# Ignorer la casse, les espaces
diff -iw ancien.txt nouveau.txt

# Comparer deux dossiers récursivement
diff -r /etc/ /tmp/etc-backup/

# Produire un patch
diff -u ancien.txt nouveau.txt > modifs.diff

# Appliquer un patch
patch < modifs.diff
```

Dans le format unifié, les lignes précédées de `-` ne se trouvent que dans le premier fichier, celles précédées de `+` que dans le second. Un administrateur s'en sert sans cesse :

```
diff -u /etc/ssh/sshd_config.orig /etc/ssh/sshd_config
```

En un coup d'œil, on voit ce qui a été modifié par rapport à l'original.

cmp et comm

`cmp` compare deux fichiers octet par octet. Il convient aux fichiers binaires : `cmp -s a.iso b.iso && echo identiques`.

`comm` compare deux listes triées :

```
# Affiche 3 colonnes : unique à A | unique à B | commun aux deux
comm fichier1_trie.txt fichier2_trie.txt

# Lignes uniquement dans A
comm -23 a.txt b.txt

# Lignes uniquement dans B
comm -13 a.txt b.txt

# Lignes communes
comm -12 a.txt b.txt
```

9.7 Encodages et fins de ligne

Deux désagréments guettent les fichiers venus d'ailleurs.

L'encodage. Linux utilise UTF-8. Un fichier produit par un vieux logiciel Windows peut être en ISO-8859-1 ou en Windows-1252 : ses lettres accentuées s'affichent alors mal.

```
gary@vostro:~$ file -i ancien.txt
ancien.txt: text/plain; charset=iso-8859-1
gary@vostro:~$ iconv -f ISO-8859-1 -t UTF-8 ancien.txt -o ancien-utf8.txt
```

Les fins de ligne. Windows termine ses lignes par deux caractères, retour chariot et saut de ligne. Linux n'utilise que le second. Un script écrit sous Windows échoue sous Linux avec un message incompréhensible, parce que le retour chariot, invisible, s'est collé à la fin de chaque commande.

```
gary@vostro:~$ file script.sh
script.sh: ASCII text, with CRLF line terminators
gary@vostro:~$ sed -i 's/\r$//' script.sh
```

Le paquet `dos2unix` fournit une commande du même nom, qui fait la même chose.

tr

`tr` remplace ou supprime des caractères, un par un.

```
# Majuscules → minuscules
echo "Bonjour Gary" | tr 'A-Z' 'a-z'
bonjour gary

# Minuscules → majuscules
echo "hello" | tr 'a-z' 'A-Z'
HELLO

# Remplacer des caractères
echo "gary.nkweka" | tr '.' '-'
gary-nkweka

# Supprimer des caractères
echo "Hello World 123" | tr -d '0-9'
Hello World

# Supprimer les caractères non-imprimables (nettoyer un fichier)
tr -cd '[:print:]\n' < sale.txt > propre.txt
```

```
# Espaces multiples → un seul
echo "a     b      c" | tr -s ' '
a b c

# Casse par classes POSIX
echo "Hello" | tr '[:lower:]' '[:upper:]'
```

ATTENTION · tr ne connaît pas les accents

`tr` travaille octet par octet. Or une lettre accentuée en occupe deux en UTF-8. `echo "École" | tr '[:upper:]' '[:lower:]'` laisse donc le É intact. Pour un texte en français, utilisez `sed 's/.*/\L&/'`, qui donne bien « école ».

9.8 Mettre en forme

```
gary@vostro:~$ head -n 3 /etc/passwd | column -t -s:
root    x  0  0  root    /root      /bin/bash
daemon  x  1  1  daemon  /usr/sbin  /usr/sbin/nologin
bin     x  2  2  bin     /bin       /usr/sbin/nologin
```

`column -t` aligne les colonnes. Placé au bout d'un tube, il rend lisible une sortie qui ne l'était pas. `fold -s -w 80` coupe les lignes trop longues, `nl` numérote les lignes, et `wc -l` les compte.

9.9 jq : lire du JSON

Les interfaces de programmation des services en ligne, et de plus en plus d'outils d'administration, répondent en JSON. `jq` est à ce format ce qu'`awk` est aux colonnes. Il s'installe avec `sudo apt install jq`.

Prenons un fichier `depots.json` :

```
[
  {"name": "site-vitrine", "stars": 12, "private": false},
  {"name": "outils-internes", "stars": 0, "private": true},
  {"name": "sauvegardes", "stars": 5, "private": false}
]
```

Un filtre `jq` décrit un chemin dans les données. Le point désigne l'élément courant, `.[]` chaque élément d'un tableau, `.name` un champ. La barre verticale enchaîne les filtres, comme dans le shell.

```
gary@vostro:~$ jq '.[0].name' depots.json
"site-vitrine"
gary@vostro:~$ jq -r '.[].name' depots.json
site-vitrine
outils-internes
sauvegardes
gary@vostro:~$ jq -c '.[] | select(.private == false) | {depot: .name, stars}' depots.json
{"depot":"site-vitrine","stars":12}
{"depot":"sauvegardes","stars":5}
gary@vostro:~$ jq '[.[].stars] | add' depots.json
17
```

Filtre	Effet
`.`	Affiche le JSON entier, joliment mis en forme
`.champ`	Extrait un champ
`.[]`	Parcourt les éléments d'un tableau
`select(condition)`	Ne garde que les éléments qui satisfont la condition
`{a: .x, b: .y}`	Construit un nouvel objet
`length`, `add`, `sort_by(.champ)`	Compte, additionne, trie

L'option `-r` retire les guillemets autour des textes. Elle est indispensable pour utiliser le résultat dans un script.

L'usage typique consiste à interroger un service avec `curl` et à ne garder que ce qui intéresse :

```
curl -s https://api.github.com/repos/jqlang/jq | jq '{nom: .full_name,
etoiles: .stargazers_count}'
```

9.10 yq : lire du YAML

Le YAML est le format des fichiers de Netplan, de Docker Compose et d'Ansible. `yq` le manipule avec une syntaxe calquée sur celle de `jq`.

> **ATTENTION · Deux outils portent ce nom**
>
> Le paquet `yq` des dépôts d'Ubuntu est un outil différent, plus limité. Celui dont il est question ici est écrit par Mike Farah. Il s'installe avec `sudo snap install yq`.

```
# Un docker-compose.yml typique
cat << EOF > compose.yml
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    environment:
      API_URL: https://api.example.com
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: secret
EOF

# Extraire un champ
yq '.services.web.image' compose.yml
nginx:latest

# Lister les services
yq '.services | keys' compose.yml
- web
- db

# Modifier en place
yq -i '.services.web.image = "nginx:1.26"' compose.yml

# Ajouter un champ
yq -i '.services.web.restart = "unless-stopped"' compose.yml

# Convertir YAML → JSON (et inversement)
yq -o=json compose.yml > compose.json
yq -p=json -o=yaml compose.json > compose-rt.yml
```

Modifier un fichier YAML avec `yq -i` est plus sûr qu'avec `sed`. L'outil comprend la structure du fichier, et l'indentation, qui a un sens en YAML, reste correcte.

En résumé

- `grep` cherche. `-i`, `-v`, `-r`, `-c` et `-C` couvrent la plupart des besoins.
- Utilisez les expressions régulières étendues : `grep -E`, `sed -E`. `\d` n'existe qu'avec `grep -P`.
- `sed 's/a/b/g'` remplace. On essaie sans `-i`, puis on applique avec `-i.orig`.
- `awk '{print $1}'` extrait une colonne, `-F` change le séparateur, les tableaux servent à compter.

- `sort | uniq -c | sort -rn` classe par fréquence.
- `sort -n` pour les nombres, `-h` pour les tailles, `LC_ALL=C` pour un ordre stable.
- `diff -u` montre ce qui a changé entre deux versions d'un fichier.
- `file -i` et `iconv` pour les encodages, `sed 's/\r$//'` pour les fins de ligne de Windows.
- `jq` pour le JSON, `yq` pour le YAML.

Atelier — Faire parler un journal web

Objectif : tirer d'un journal d'accès Nginx les informations qui comptent. Comptez trente minutes.

Créez le fichier `access.log` :

```
192.168.1.10 - - [24/Apr/2026:10:15:32 +0100] "GET / HTTP/1.1" 200 1234 "-" "Mozilla/5.0"
192.168.1.10 - - [24/Apr/2026:10:15:45 +0100] "GET /a-propos HTTP/1.1" 200 2431 "-"
"Mozilla/5.0"
203.0.113.5 - - [24/Apr/2026:10:16:02 +0100] "GET /admin HTTP/1.1" 403 178 "-" "curl/8.5.0"
198.51.100.12 - - [24/Apr/2026:10:16:10 +0100] "GET /contact HTTP/1.1" 200 3100 "-"
"Mozilla/5.0"
192.168.1.10 - - [24/Apr/2026:10:16:15 +0100] "POST /connexion HTTP/1.1" 200 500 "-"
"Mozilla/5.0"
203.0.113.5 - - [24/Apr/2026:10:16:20 +0100] "GET /wp-admin HTTP/1.1" 404 212 "-" "curl/
8.5.0"
203.0.113.5 - - [24/Apr/2026:10:16:22 +0100] "GET /.env HTTP/1.1" 404 212 "-" "curl/8.5.0"
198.51.100.12 - - [24/Apr/2026:10:17:05 +0100] "GET /a-propos HTTP/1.1" 200 2431 "-"
"Mozilla/5.0"
192.168.1.10 - - [24/Apr/2026:10:17:20 +0100] "GET /contact HTTP/1.1" 200 3100 "-" "Mozilla/
5.0"
203.0.113.5 - - [24/Apr/2026:10:18:00 +0100] "GET /config.php HTTP/1.1" 500 212 "-" "curl/
8.5.0"
```

Dans ce format, l'adresse du visiteur est le champ 1, la page demandée le champ 7, le code de réponse le champ 9 et la taille de la réponse le champ 10.

Qui visite le site ?

```
gary@vostro:~$ awk '{print $1}' access.log | sort | uniq -c | sort -rn
      4 203.0.113.5
      4 192.168.1.10
      2 198.51.100.12
```

Comment le serveur répond-il ?

```
gary@vostro:~$ awk '{print $9}' access.log | sort | uniq -c | sort -rn
      6 200
      2 404
      1 500
      1 403
```

À vous. Écrivez la commande qui répond à chaque question. Les résultats attendus sont indiqués.

1. Quelles sont les trois pages les plus demandées ? (`/contact` et `/a-propos`, deux fois chacune)
2. Combien d'octets le serveur a-t-il envoyés en tout ? (13 610)
3. Quelles adresses reçoivent des réponses 404, et combien ? (`203.0.113.5`, deux fois)
4. Affichez l'adresse et la page de toutes les requêtes qui visent `/admin`, `/wp-admin`, `/.env` ou un fichier `.php`.
5. Quels programmes clients apparaissent dans le journal ? Pour découper aux guillemets, utilisez `awk -F'"'`.
6. Que cherche manifestement le visiteur `203.0.113.5` ? Le chapitre 21 montre comment lui fermer la porte.

Si vous disposez d'un vrai serveur web, rejouez ces commandes sur `/var/log/nginx/access.log`. Les résultats sont souvent instructifs.

Exercices

Exercice 1 — grep sur /etc/passwd

1. Affichez les lignes qui contiennent `bash`.
2. Affichez celles qui ne contiennent pas `nologin`.
3. Comptez les comptes dont le shell est `/usr/sbin/nologin`.
4. Affichez les comptes dont le dossier personnel se trouve dans `/var`.

Exercice 2 — Expressions régulières

Écrivez, en dialecte étendu, une expression qui correspond à :

1. un numéro de téléphone camerounais écrit `+237 6XX XX XX XX` ;
2. une date écrite `2026-04-23` ;

3. une heure écrite `14:05:09` ;
4. une ligne qui n'est ni vide ni un commentaire.

Exercice 3 — sed

1. Remplacez `exemple.com` par `exemple.cm` dans un fichier, en gardant une copie de l'original.
2. Supprimez les lignes vides et les commentaires.
3. Remplacez `http://` par `https://`, uniquement dans les lignes qui commencent par `Location`.
4. Affichez les lignes 20 à 30.

Exercice 4 — awk sur /etc/passwd

1. Affichez le nom et le shell de chaque compte, séparés par une tabulation.
2. Comptez les comptes par shell.
3. Trouvez l'UID le plus élevé.

Exercice 5 — En une ligne

1. Les cinq commandes que vous utilisez le plus, d'après votre historique.
2. Les cinq comptes de `/etc/passwd` qui ont l'UID le plus élevé.
3. Les lignes présentes dans `a.txt` et absentes de `b.txt`.
4. Le nombre d'adresses différentes dans `access.log`.

Exercice 6 — jq

Avec le fichier `depots.json` de la section 9.9 :

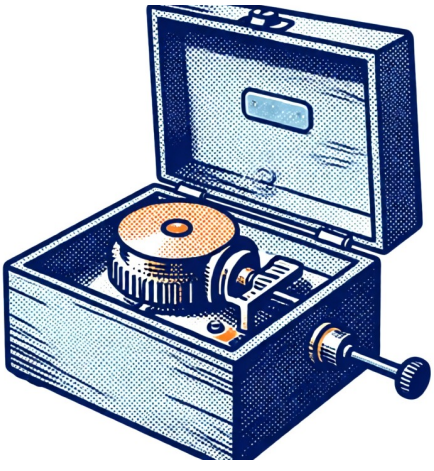
1. Affichez le nom des dépôts privés.
2. Comptez les dépôts.
3. Affichez le nom du dépôt qui a le plus d'étoiles.
4. Produisez, pour chaque dépôt, une ligne de la forme `site-vitrine : 12`.

Pour aller plus loin

Références. `man grep`, `man sed`, `man awk`. Le manuel de `gawk`, sur `gnu.org/software/gawk/manual/`, est aussi un excellent cours sur le langage. `jqlang.github.io/jq/manual/` pour `jq`.

Livres. *The AWK Programming Language*, d'Aho, Kernighan et Weinberger, réédité en 2023 : écrit par les auteurs du langage, bref et limpide. *sed & awk*, de Dale Dougherty et Arnold Robbins (O'Reilly). *Mastering Regular Expressions*, de Jeffrey Friedl (O'Reilly), pour tout savoir des expressions régulières.

Outils. `ripgrep` cherche dans une arborescence bien plus vite que `grep -r`. `miller` (`sudo apt install miller`, commande `mlr`) fait pour les fichiers CSV ce qu'`awk` fait pour les colonnes, en tenant compte des guillemets et des en-têtes.



CHAPITRE 10

Écrire des scripts Bash

Un script, c'est une suite de commandes que l'on n'aura plus à retaper. Encore faut-il qu'il s'arrête quand quelque chose échoue, et qu'on le comprenne encore dans six mois.

DANS CE CHAPITRE

- Écrire un script exécutable, avec son en-tête et ses droits.
- Utiliser variables, tableaux et tableaux associatifs.
- Lire des arguments, des options et des saisies au clavier.
- Écrire des conditions, des boucles et des aiguillages.
- Découper un script en fonctions.
- Faire échouer un script proprement, et nettoyer derrière lui.
- Déboguer, et contrôler ses scripts avec `shellcheck`.
- Écrire un script que l'on peut relancer sans dégât.

> **AVANT DE COMMENCER**
>
> Les chapitres 8 et 9 : expansions, guillemets, tubes et redirections. Tous les scripts et tous les comportements décrits ici ont été exécutés avec Bash 5.2, la version d'Ubuntu 24.04.

10.1 Anatomie d'un script

Un script est un fichier texte qui contient des commandes. Trois choses en font un programme que l'on peut lancer.

La première ligne indique quel programme doit lire le fichier. On l'appelle le *she-bang* :

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

Sans elle, le script est lu par le shell courant, qui n'est pas toujours Bash. Écrivez `#!/bin/bash`, et non `#!/bin/sh` : sur Ubuntu, `sh` est `dash`, qui ignore une bonne partie de ce chapitre.

Un en-tête dit ce que fait le script et comment on s'en sert. Quelques lignes suffisent.

Le droit d'exécution permet de le lancer par son nom :

```
gary@vostro:~$ chmod +x sauvegarde.sh
gary@vostro:~$ ./sauvegarde.sh
```

Voici un premier script complet :

```
#!/bin/bash
#
# bonjour.sh — salue la personne dont le nom est donné
#
# Usage : bonjour.sh NOM

set -euo pipefail

echo "Bonjour, $1 !"
```

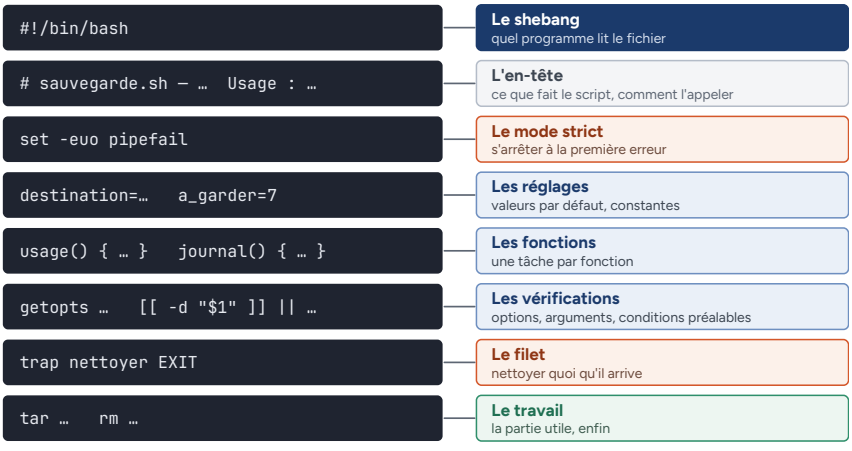


Figure 10.1 — Les parties d'un script bien construit, dans l'ordre où on les écrit.

La ligne `set -euo pipefail` figure en tête de tous les scripts de ce livre. Elle demande à Bash de s'arrêter à la première erreur. La section 10.8 l'explique en détail.

> **ASTUCE · Où ranger ses scripts**
>
> Placez vos scripts dans `~/bin`. Sur Ubuntu, ce dossier est ajouté à `PATH` s'il existe : vos scripts se lancent alors comme des commandes, de n'importe où. Dans ce cas, on leur retire souvent l'extension `.sh`.

10.2 Variables et tableaux

Les variables

```
nom="Gary"
age=33
readonly DOSSIER_JOURNAUX="/var/log/monappli"

echo "$nom a $age ans"
```

L'affectation s'écrit sans espace autour du signe égal. C'est l'erreur du débutant par excellence :

```
gary@vostro:~$ nom = "Gary"
nom : commande introuvable
```

Bash a compris qu'il fallait lancer une commande `nom` avec deux arguments.

Par convention, on écrit les constantes en majuscules et les variables ordinaires en minuscules. `readonly` interdit de modifier une valeur par la suite.

Les tableaux

Un tableau contient plusieurs valeurs, numérotées à partir de zéro.

```
fruits=(pomme banane orange)

echo "${fruits[0]}"        # pomme
echo "${#fruits[@]}"       # 3 : le nombre d'éléments
echo "${fruits[@]:1:2}"    # banane orange : deux éléments à partir du deuxième

fruits+=(kiwi)             # ajouter à la fin

for fruit in "${fruits[@]}"; do
    echo "- $fruit"
done
```

`"${fruits[@]}"`, avec ses guillemets, désigne tous les éléments, chacun restant entier même s'il contient une espace. C'est la forme à utiliser dans une boucle.

Les tableaux sont le bon moyen de construire une liste d'arguments :

```
options=(-a --delete --exclude ".cache")
rsync "${options[@]}" "$source/" "$destination/"
```

Les tableaux associatifs

Ici, l'indice n'est plus un numéro mais un mot. Il faut les déclarer avec `declare -A`.

```
declare -A port=([ssh]=22 [http]=80 [https]=443)

echo "${port[ssh]}"                # 22
port[smtp]=25                      # ajouter une entrée

for service in "${!port[@]}"; do   # ${!port[@]} donne les indices
    echo "$service : ${port[$service]}"
done

if [[ -v port[dns] ]]; then echo "connu"; else echo "inconnu"; fi
```

L'ordre dans lequel la boucle parcourt un tableau associatif n'est pas garanti.

10.3 Lire des entrées

Les arguments

Le script reçoit ses arguments dans `$1`, `$2` et ainsi de suite. `$#` en donne le nombre et `"$@"` les contient tous.

```
#!/bin/bash
set -euo pipefail

if [[ $# -ne 2 ]]; then
    echo "Usage : $0 SOURCE DESTINATION" >&2
    exit 2
fi

source="$1"
destination="$2"

[[ -d "$source" ]] || { echo "Dossier introuvable : $source" >&2; exit 1; }
```

Un script sérieux vérifie ce qu'on lui donne avant d'agir. Les messages d'erreur s'écrivent sur la sortie d'erreur, avec `>&2`. Le code de retour 2 signale par convention une erreur d'utilisation.

`shift` retire le premier argument et décale les autres : `$2` devient `$1`.

Les options, avec getopts

Pour accepter des options comme `-v` ou `-d dossier`, Bash fournit `getopts`.

```
#!/bin/bash
# backup.sh -d DIR -k N -v

# Valeurs par défaut
DIR="/var/backups"
KEEP=7
VERBOSE=0

# Aide
usage() {
    cat <<EOF
Usage: $0 [-d DIR] [-k N] [-v] [-h] <database>
  -d DIR   Dossier de destination (défaut: $DIR)
  -k N     Garder N sauvegardes (défaut: $KEEP)
  -v       Mode verbeux
  -h       Cette aide
EOF
    exit 1
}
```

```
# Parser les options
while getopts "d:k:vh" opt; do
    case "$opt" in
        d) DIR="$OPTARG" ;;
        k) KEEP="$OPTARG" ;;
        v) VERBOSE=1 ;;
        h) usage ;;
        *) usage ;;
    esac
done

# Décaler pour récupérer les arguments positionnels
shift $((OPTIND - 1))

# Vérifier qu'un argument positionnel reste (la database)
if [[ $# -lt 1 ]]; then
    echo "Erreur : nom de base manquant" >&2
    usage
fi

DATABASE="$1"

echo "Sauvegarde de $DATABASE vers $DIR (keep=$KEEP, verbose=$VERBOSE)"
```

La chaîne `"d:k:vh"` énumère les options admises. Les deux-points qui suivent `d` et `k` signalent qu'elles attendent une valeur, que `getopts` range dans `OPTARG`. Après la boucle, `shift $((OPTIND - 1))` retire les options, et il ne reste que les arguments ordinaires.

`getopts` ne connaît que les options d'une lettre. Pour des options longues comme `--verbose`, il faut écrire soi-même la boucle d'analyse, ou changer de langage.

Les saisies au clavier

```
#!/bin/bash

# Lecture simple
read -p "Votre nom : " name
echo "Bonjour $name"

# Lecture avec timeout (5 secondes)
read -t 5 -p "Réponse rapide : " answer

# Lecture sans écho (mots de passe)
read -sp "Mot de passe : " pwd
echo  # nouvelle ligne après l'invite

# Lecture de plusieurs valeurs
```

```
read -p "Prénom et nom : " prenom nom

# Lecture d'une seule touche (sans appuyer sur Entrée)
read -n 1 -p "Continuer ? (o/n) " choix
echo
[[ "$choix" != "o" ]] && exit 0

# Lecture ligne par ligne d'un fichier
while IFS= read -r line; do
    echo "Ligne : $line"
done < fichier.txt
```

L'expression `while IFS= read -r ligne` est la façon correcte de lire un fichier ligne par ligne. `-r` laisse les barres obliques inversées intactes, et `IFS=` conserve les espaces de début et de fin.

10.4 Les conditions

if

```
if [[ "$USER" == "root" ]]; then
    echo "Ne lancez pas ce script en tant que root." >&2
    exit 1
elif [[ -z "${1:-}" ]]; then
    echo "Argument manquant." >&2
    exit 2
else
    echo "C'est parti."
fi
```

Ce que teste `if`, c'est le code de retour d'une commande. On peut donc écrire directement :

```
if grep -q "^alice:" /etc/passwd; then
    echo "Le compte alice existe."
fi

if ! ping -c 1 -W 2 serveur > /dev/null 2>&1; then
    echo "Le serveur ne répond pas."
fi
```

Les doubles crochets

`[[ … ]]` est la commande de test de Bash. Vous rencontrerez aussi l'ancienne forme à simples crochets, `[ … ]`, qui existe dans tous les shells mais se montre

plus capricieuse avec les variables vides. Dans un script Bash, utilisez toujours les doubles crochets.

Test	Vrai si…
`-e chemin`	le fichier ou le dossier existe
`-f chemin`	c'est un fichier ordinaire
`-d chemin`	c'est un dossier
`-r`, `-w`, `-x chemin`	on peut le lire, le modifier, l'exécuter
`-s chemin`	le fichier existe et n'est pas vide
`-z "$var"`	la variable est vide
`-n "$var"`	la variable n'est pas vide
`"$a" == "$b"`, `"$a" != "$b"`	les deux textes sont identiques, différents
`"$a" == *.txt`	le texte correspond au motif
`"$a" =~ expression`	le texte correspond à l'expression régulière
`$a -eq $b`, `-ne`, `-lt`, `-le`, `-gt`, `-ge`	comparaison de nombres : égal, différent, inférieur…

```bash
# Égalité
[[ "$a" == "$b" ]]

# Inégalité
[[ "$a" != "$b" ]]

# Vide / non-vide
[[ -z "$var" ]]              # vide ou non définie
[[ -n "$var" ]]              # non-vide

# Comparaison avec un motif
[[ "$file" == *.txt ]]       # se termine par .txt
[[ "$file" == /etc/* ]]      # commence par /etc/

# Comparaison avec une expression régulière
if [[ "$ip" =~ ^([0-9]{1,3}\.){3}[0-9]{1,3}$ ]]; then
    echo "IP valide : $ip"
fi

# Les parenthèses capturent : le résultat est dans BASH_REMATCH
if [[ "$line" =~ ^([a-z]+):([0-9]+)$ ]]; then
    nom="${BASH_REMATCH[1]}"
    valeur="${BASH_REMATCH[2]}"
fi
```

Les nombres

Pour comparer des nombres, les doubles parenthèses sont plus lisibles que `-lt` et `-ge` :

```
# Avec [[ ]] : -eq, -ne, -lt, -le, -gt, -ge
[[ "$age" -ge 18 ]]
[[ "$x" -eq 10 ]]

# Avec (( )) : les opérateurs habituels, plus lisibles
(( age >= 18 ))
(( x == 10 ))
(( x < y && y < z ))

# Pas besoin de $ devant les variables dans (( ))
(( count++ ))
(( total = a + b ))

if (( age >= 18 )); then
    echo "Majeur"
fi
```

Combiner

```
[[ -f "$fichier" && -r "$fichier" ]]          # et
[[ "$nom" == *.jpg || "$nom" == *.png ]]      # ou
[[ ! -d "$dossier" ]]                         # non
```

On écrit souvent de petites conditions sur une ligne : `[[ -d "$dossier" ]] || mkdir "$dossier"`. C'est lisible tant que cela reste court. N'enchaînez pas `&&` et `||` sur la même ligne pour imiter un `if … else` : si la commande du milieu échoue, celle de la fin est exécutée aussi.

10.5 Les boucles

for

```
# Sur une liste explicite
for fruit in pomme banane orange; do
    echo "$fruit"
done

# Sur des fichiers (globbing)
for f in *.txt; do
    echo "Traitement : $f"
done
```

```bash
# Sur les arguments du script
for arg in "$@"; do
    echo "Arg: $arg"
done

# Une suite de nombres
for i in {1..10}; do
    echo "Itération $i"
done

# Avec un pas
for i in {0..100..10}; do
    echo "$i %"
done

# À la manière du langage C
for ((i = 0; i < 10; i++)); do
    echo "$i"
done

# Sur un tableau
fichiers=(/etc/hosts /etc/hostname /etc/resolv.conf)
for f in "${fichiers[@]}"; do
    [[ -r "$f" ]] && wc -l "$f"
done
```

ATTENTION · Quand le motif ne correspond à rien

S'il n'y a aucun fichier `.txt`, la boucle `for f in *.txt` s'exécute quand même une fois, avec `f` égal au texte `*.txt`. Deux parades : commencer la boucle par `[[ -e "$f" ]] || continue`, ou placer `shopt -s nullglob` en tête du script.

while et until

```bash
# Tant que la condition est vraie
count=0
while (( count < 5 )); do
    echo "count = $count"
    (( count++ ))
done

# Lecture ligne par ligne d'un fichier (idiome courant)
while IFS= read -r line; do
    echo "Ligne : $line"
done < /etc/hosts

# Boucle infinie
while true; do
```

```
    echo "Tour"
    sleep 1
done
```

`until` fait l'inverse : il boucle tant que la commande échoue. C'est commode pour attendre qu'un service soit prêt.

```
# Attendre qu'un service réponde
until curl -fs http://localhost:8080/health > /dev/null; do
    echo "Attente du serveur..."
    sleep 2
done
echo "Serveur prêt"
```

DANGER · Une boucle dans un tube oublie ses variables

Chaque élément d'un tube s'exécute dans un shell séparé. Une variable modifiée dans une boucle `while` placée après un tube retrouve donc son ancienne valeur à la sortie.

```
n=0
printf 'a\nb\nc\n' | while read -r ligne; do n=$((n+1)); done
echo "$n"       # affiche 0

n=0
while read -r ligne; do n=$((n+1)); done < <(printf 'a\nb\nc\n')
echo "$n"       # affiche 3
```

La seconde forme alimente la boucle par une redirection. La boucle reste dans le shell principal.

break et continue

`break` sort de la boucle. `continue` passe au tour suivant.

```
# break : sortir de la boucle immédiatement
for f in /etc/*.conf; do
    if grep -q "SECRET" "$f"; then
        echo "Trouvé dans $f"
        break
    fi
done

# continue : passer à l'itération suivante
for f in *.txt; do
    [[ ! -s "$f" ]] && continue   # passe les fichiers vides
    process "$f"
```

```
done

# break N : sortir de N boucles imbriquées
for i in {1..5}; do
    for j in {1..5}; do
        if (( i*j > 12 )); then
            break 2     # sort des deux boucles
        fi
    done
done
```

10.6 case

Quand une variable peut prendre plusieurs valeurs, `case` est plus clair qu'une cascade de `elif`.

```
#!/bin/bash
# deploie.sh ENVIRONNEMENT

case "$1" in
    dev|development)
        echo "Déploiement en développement"
        URL="https://dev.example.com"
        ;;
    staging|preprod)
        echo "Déploiement en staging"
        URL="https://staging.example.com"
        ;;
    prod|production)
        echo "Déploiement en PRODUCTION"
        read -p "Confirmer ? (oui/NON) " answer
        [[ "$answer" != "oui" ]] && exit 0
        URL="https://example.com"
        ;;
    -h|--help)
        echo "Usage: $0 {dev|staging|prod}"
        exit 0
        ;;
    *)
        echo "Erreur: environnement inconnu '$1'" >&2
        echo "Usage: $0 {dev|staging|prod}" >&2
        exit 1
        ;;
esac

# Avec extglob (motifs avancés)
shopt -s extglob
case "$file" in
    *.@(jpg|jpeg|png|gif))
```

```
        echo "Image : $file"
        ;;
    *.@(mp4|avi|mkv))
        echo "Vidéo : $file"
        ;;
esac
```

Chaque branche se termine par deux points-virgules. La barre verticale sépare les valeurs équivalentes. La branche `*)` recueille tout le reste : ne l'oubliez jamais, c'est elle qui attrape les fautes de frappe de l'utilisateur.

10.7 Les fonctions

Une fonction donne un nom à un groupe de commandes.

```
saluer() {
    local nom="$1"
    echo "Bonjour, $nom !"
}

saluer "Alice"
```

À l'intérieur, `$1`, `$2` et `$#` désignent les arguments de la fonction, non ceux du script.

> **À RETENIR · Déclarez vos variables avec local**
>
> Sans `local`, une variable créée dans une fonction est visible dans tout le script. Si une autre porte le même nom ailleurs, elle est écrasée, et l'erreur est très difficile à trouver. Toute variable d'une fonction se déclare avec `local`.

Une fonction ne renvoie pas de valeur comme dans d'autres langages. Elle dispose de deux moyens.

```
# 1. return : un code de retour
user_existe() {
    if id -u "$1" &>/dev/null; then
        return 0   # succès
    else
        return 1   # échec
    fi
}

if user_existe "alice"; then
    echo "alice existe"
```

```
fi

# 2. echo + $(...)
get_disk_usage() {
    df -h "$1" | awk 'NR==2 {print $5}'
}

usage=$(get_disk_usage /var)
echo "/var : $usage"

# 3. Une variable globale (à éviter quand on peut)
calculer() {
    RESULTAT=$(( $1 + $2 ))
}

calculer 5 3
echo "$RESULTAT"   # 8
```

`return` fixe le code de retour, un nombre de 0 à 255 qui sert à dire « réussi » ou « échoué ». Pour produire une valeur, la fonction l'affiche, et l'appelant la récupère avec `$( )`.

Quand plusieurs scripts ont besoin des mêmes fonctions, on les place dans un fichier à part, que chaque script charge avec `source` :

```
source /usr/local/lib/monprojet/fonctions.sh
```

10.8 Codes de retour et gestion des erreurs

Les codes de retour

Code	Signification
0	Succès
1	Échec
2	Mauvaise utilisation : arguments incorrects
126	Le fichier existe, mais n'est pas exécutable
127	Commande introuvable
130	Interrompu par Ctrl+C

Un script se termine par `exit` suivi d'un code. Sans `exit`, il renvoie le code de sa dernière commande.

set -euo pipefail

Par défaut, Bash poursuit sa route après une erreur. Un `cd` qui échoue, et le `rm -rf *` de la ligne suivante s'exécute dans le mauvais dossier. Trois réglages changent ce comportement.

Réglage	Effet
`set -e`	Le script s'arrête dès qu'une commande échoue
`set -u`	L'emploi d'une variable qui n'existe pas est une erreur
`set -o pipefail`	Un tube échoue si l'une de ses commandes échoue, et non plus seulement la dernière

```
gary@vostro:~$ bash -c 'false | true; echo $?'
0
gary@vostro:~$ bash -c 'set -o pipefail; false | true; echo $?'
1
```

`set -u` vous protège des fautes de frappe dans les noms de variables. Sans lui, `rm -rf "$dosier/"`, avec un `s` oublié, devient `rm -rf "/"`.

`set -e` connaît toutefois des exceptions, et il faut les connaître.

ATTENTION · Trois cas où set -e surprend

Dans une condition, il est suspendu. Une commande testée par `if`, `while`, `&&` ou `||` peut échouer sans arrêter le script. C'est voulu.

`local` masque les échecs. Dans `local v=$(commande)`, c'est le code de retour de `local` qui compte, et il vaut toujours 0. Séparez la déclaration de l'affectation :

```
local v
v=$(commande)
```

Une fonction qui finit par un test faux arrête le script. Regardez cette fonction :

```
note() { [[ "$bavard" -eq 1 ]] && echo "$1"; }
```

Quand `bavard` vaut 0, le test est faux, la fonction renvoie 1, et le script s'arrête sans un mot. Écrivez un vrai `if`.

trap : nettoyer en partant

`trap` enregistre une commande à exécuter quand le script se termine, quelle qu'en soit la raison : fin normale, erreur, Ctrl+C.

```
#!/bin/bash
set -euo pipefail

travail="$(mktemp -d)"
trap 'rm -rf "$travail"' EXIT

# … le script travaille dans "$travail" …
```

`mktemp -d` crée un dossier temporaire au nom imprévisible. Le `trap` garantit qu'il sera supprimé. On n'a plus à y penser.

Traiter une erreur soi-même

```
# Tester explicitement le code de sortie
if ! cp source dest; then
    echo "Échec de la copie" >&2
    exit 1
fi

# Avec || pour un fallback simple
command || { echo "Erreur"; exit 1; }

# Capturer la sortie ET le code
if output=$(commande 2>&1); then
    echo "OK : $output"
else
    rc=$?
    echo "Erreur ($rc) : $output" >&2
    exit $rc
fi

# Fonction d'erreur fatale
die() {
    echo "FATAL: $*" >&2
    exit 1
}

[[ -f config.yml ]] || die "config.yml absent"
```

10.9 Déboguer

Voir ce que fait le script

`bash -x script.sh` affiche chaque commande au moment où elle s'exécute, après toutes les expansions. On voit exactement ce que Bash a compris.

```
gary@vostro:~$ bash -x bonjour.sh "Alice Nkomo"
+ set -euo pipefail
+ echo 'Bonjour, Alice Nkomo !'
Bonjour, Alice Nkomo !
```

Pour ne tracer qu'un passage, encadrez-le de `set -x` et de `set +x`.

`bash -n script.sh` vérifie la syntaxe sans rien exécuter.

shellcheck

`shellcheck` lit un script et signale les erreurs classiques : guillemets oubliés, variables jamais utilisées, tests mal écrits. Il s'installe avec `sudo apt install shellcheck`.

```
gary@vostro:~$ shellcheck essai.sh

In essai.sh line 3:
rm $fichier
   ^------^ SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean:
rm "$fichier"
```

Chaque remarque porte un numéro. Le site `shellcheck.net/wiki` explique, pour chacun, le problème et la correction.

> **À RETENIR · Passez tous vos scripts à shellcheck**
>
> Il trouve en une seconde des erreurs qui vous coûteraient une heure. Prenez l'habitude de le lancer avant chaque utilisation d'un script nouveau ou modifié. Le chapitre 27 montre comment le faire tourner automatiquement à chaque enregistrement dans Git.

10.10 De bonnes habitudes

- Commencez par `#!/bin/bash` et `set -euo pipefail`.

- **Mettez toutes les variables entre guillemets :** `"$fichier"`, `"$@"`.
- **Déclarez les variables des fonctions avec** `local`.
- **Vérifiez les arguments avant d'agir**, et affichez un message d'usage quand ils sont incorrects.
- **Écrivez les erreurs sur la sortie d'erreur** et sortez avec un code différent de zéro.
- **Nommez clairement.** `dossier_source` vaut mieux que `ds`.
- **Commentez le pourquoi, pas le comment.** `# Le fournisseur limite à 100 requêtes par minute` est utile. `# incrémente i` ne l'est pas.
- **Nettoyez avec** `trap` tout ce que le script crée de temporaire.
- **Faites relire par** `shellcheck`.

10.11 Des scripts que l'on peut relancer

Un script est *idempotent* quand on peut l'exécuter plusieurs fois de suite sans autre effet que la première fois. Le mot est savant, l'idée est simple, et elle est au cœur de toute automatisation.

Pourquoi y tenir ? Parce qu'un script s'interrompt. Le réseau tombe au milieu d'une installation, et il faut pouvoir le relancer sans qu'il échoue sur ce qui est déjà fait, ni qu'il le fasse en double.

```
# 1. CRÉATION : utiliser -p, vérifier l'existence

# Mauvais : échoue si le dossier existe
mkdir /opt/myapp

# Bon : -p ne se plaint pas
mkdir -p /opt/myapp

# 2. AJOUT À UN FICHIER : vérifier avant d'ajouter

# Mauvais : ajoute la ligne à chaque exécution
echo "export PATH=$HOME/bin:\$PATH" >> ~/.bashrc

# Bon : grep avant
if ! grep -q "PATH=$HOME/bin" ~/.bashrc; then
    echo "export PATH=$HOME/bin:\$PATH" >> ~/.bashrc
fi

# 3. INSTALLATION DE PAQUET : apt sait déjà
sudo apt install -y nginx
# apt est idempotent : ne réinstalle pas si déjà à la bonne version

# 4. CRÉATION D'UTILISATEUR
```

```
id alice &>/dev/null || sudo adduser --disabled-password --gecos "" alice

# 5. SERVICE
sudo systemctl enable nginx       # idempotent
sudo systemctl start nginx        # idempotent (déjà actif = no-op)

# 6. CONFIGURATION : remplacer plutôt qu'ajouter

# Mauvais : append à chaque fois
echo "timeout=30" >> /etc/myapp.conf

# Bon : sed remplace si présent, sinon ajoute
if grep -q "^timeout=" /etc/myapp.conf; then
    sudo sed -i 's/^timeout=.*/timeout=30/' /etc/myapp.conf
else
    echo "timeout=30" | sudo tee -a /etc/myapp.conf
fi

# 7. SUPPRESSION : -f ignore l'absence
rm -f /tmp/oldfile      # pas d'erreur si absent

# 8. SYMLINK : ln -sf pour remplacer s'il existe
ln -sf /opt/myapp/bin/myapp /usr/local/bin/myapp
```

Le principe est toujours le même : décrire l'état voulu plutôt que l'action à faire. On n'écrit pas « ajouter cette ligne », mais « faire en sorte que cette ligne soit présente ».

> **À RETENIR · Le test des deux passages**
>
> Avant de considérer un script d'installation comme terminé, lancez-le deux fois de suite. Le second passage ne doit rien changer et ne doit afficher aucune erreur.

10.12 Quand quitter Bash

Bash excelle à enchaîner des commandes. Il devient pénible dès que l'on s'éloigne de cet usage. Quelques signes ne trompent pas :

- le script dépasse deux ou trois cents lignes ;
- il manipule des données structurées, comme du JSON, au-delà de ce qu'un appel à `jq` règle ;
- il calcule sur des nombres à virgule ;
- il a besoin de listes de listes, ou de dictionnaires imbriqués ;
- il interroge des services web et doit traiter finement leurs erreurs ;

- vous aimeriez écrire des tests pour vérifier qu'il fonctionne.

Dans tous ces cas, Python est le choix naturel. Il est installé sur tout système Ubuntu, et sa bibliothèque standard couvre presque tous les besoins d'administration. Bash reste le bon outil pour le reste : assembler des commandes, en une page.

En résumé

- Un script commence par `#!/bin/bash`, puis `set -euo pipefail`, et doit être exécutable.
- Pas d'espace autour du `=` d'une affectation. Des guillemets autour de toutes les variables.
- `$1`, `$#` et `"$@"` pour les arguments, `getopts` pour les options, `read` pour les saisies.
- `[[ … ]]` pour les tests sur les textes et les fichiers, `(( … ))` pour les nombres.
- `for` parcourt une liste, `while read` lit des lignes, `case` aiguille.
- Les variables d'une fonction se déclarent avec `local`. Une fonction produit une valeur en l'affichant.
- `trap … EXIT` nettoie, quoi qu'il arrive.
- `bash -x` montre ce que fait le script, `shellcheck` ce qu'il fait mal.
- Un bon script d'installation peut être relancé sans dégât.

Atelier — Un script de sauvegarde

Objectif : écrire un script qui archive un dossier, tient un journal et ne garde que les dernières archives. Comptez une heure.

Le cahier des charges

Le script reçoit un dossier en argument. Il accepte quatre options : `-d` pour le dossier de destination, `-k` pour le nombre d'archives à garder, `-v` pour afficher le détail, `-h` pour l'aide. Il crée une archive `.tar.gz` qui porte la date et l'heure, écrit chaque étape dans un journal, supprime les archives en trop, et ne laisse jamais d'archive incomplète derrière lui.

Le script

Essayez d'abord de l'écrire seul, en vous aidant des sections 10.3, 10.7 et 10.8. Comparez ensuite avec cette version, qui a été testée :

```
#!/bin/bash
#
# sauvegarde.sh — archive un dossier et ne garde que les archives les plus récentes
#
# Usage : sauvegarde.sh [-d DESTINATION] [-k NOMBRE] [-v] [-h] DOSSIER

set -euo pipefail

readonly NOM_SCRIPT="${0##*/}"

destination="$HOME/sauvegardes"
a_garder=7
bavard=0

usage() {
    cat <<EOF
Usage : $NOM_SCRIPT [-d DESTINATION] [-k NOMBRE] [-v] [-h] DOSSIER

Archive DOSSIER dans DESTINATION et ne garde que les NOMBRE archives les plus récentes.

  -d DESTINATION  dossier des archives (par défaut : $destination)
  -k NOMBRE       nombre d'archives à garder (par défaut : $a_garder)
  -v              affiche le détail des opérations
  -h              affiche cette aide
EOF
}

# Écrit une ligne datée dans le journal ; l'affiche aussi si c'est une erreur ou avec -v
journal() {
    local niveau="$1"
    shift
    local ligne
    ligne="$(date '+%F %T') [$niveau] $*"
    echo "$ligne" >> "$fichier_journal"
    if [[ "$niveau" != "INFO" || "$bavard" -eq 1 ]]; then
        echo "$ligne" >&2
    fi
}

erreur() {
    echo "$NOM_SCRIPT : $*" >&2
    exit 1
}

# --- Options ---
while getopts ":d:k:vh" option; do
    case "$option" in
        d) destination="$OPTARG" ;;
        k) a_garder="$OPTARG" ;;
        v) bavard=1 ;;
        h) usage; exit 0 ;;
        :) erreur "l'option -$OPTARG attend une valeur" ;;
        *) usage >&2; exit 2 ;;
    esac
done
shift $((OPTIND - 1))
```

```
# --- Vérifications ---
if [[ $# -ne 1 ]]; then
    usage >&2
    exit 2
fi
dossier="${1%/}"

[[ -d "$dossier" ]] || erreur "dossier introuvable : $dossier"
[[ "$a_garder" =~ ^[1-9][0-9]*$ ]] || erreur "-k attend un nombre entier positif : $a_gar
der"

mkdir -p "$destination"
readonly fichier_journal="$destination/sauvegarde.log"

nom="$(basename "$dossier")"
archive="$destination/$nom-$(date +%F-%H%M%S).tar.gz"
terminee=0

# --- En cas d'interruption, ne pas laisser d'archive incomplète ---
nettoyer() {
    if [[ "$terminee" -eq 0 && -f "$archive" ]]; then
        rm -f "$archive"
        journal ERREUR "sauvegarde interrompue, archive incomplète supprimée"
    fi
}
trap nettoyer EXIT

# --- Archivage ---
journal INFO "début : $dossier vers $archive"
tar -czf "$archive" -C "$(dirname "$dossier")" "$nom"
terminee=1
journal INFO "archive créée ($(du -h "$archive" | cut -f1))"

# --- Rotation ---
# Les archives portent leur date : l'ordre alphabétique est l'ordre chronologique.
mapfile -t archives < <(find "$destination" -maxdepth 1 -type f -name "$nom-*.tar.gz" | s
ort)
en_trop=$(( ${#archives[@]} - a_garder ))
for (( i = 0; i < en_trop; i++ )); do
    rm -f "${archives[i]}"
    journal INFO "ancienne archive supprimée : ${archives[i]##*/}"
done

journal INFO "fin de la sauvegarde"
```

Points à remarquer

- Les erreurs d'utilisation renvoient le code 2, les autres le code 1.
- La fonction `journal` contient un vrai `if`. Avec la forme courte `[[ … ]] && echo`, le script s'arrêterait au premier message, à cause de `set -e`.
- La variable `terminee` permet à la fonction `nettoyer` de savoir si l'archive est complète.
- Les archives portent leur date dans leur nom, sous la forme année, mois, jour. Les trier par ordre alphabétique revient à les trier par date.

Mettre à l'épreuve

1. Passez le script à `shellcheck`. Il ne doit rien signaler.
2. Lancez-le avec `-h`, puis sans argument, puis avec un dossier qui n'existe pas. Regardez le code de retour à chaque fois avec `echo $?`.
3. Sauvegardez un dossier cinq fois de suite avec `-k 3`. Il doit rester trois archives.
4. Lisez le journal. Chaque passage a laissé ses lignes, datées.
5. Lancez une sauvegarde d'un gros dossier et interrompez-la avec Ctrl+C. Vérifiez qu'aucune archive incomplète ne reste, et que le journal en garde la trace.

Le chapitre 13 montre comment lancer ce script chaque nuit.

Exercices

Exercice 1 — Trouver les erreurs

Ce script contient au moins cinq erreurs. Trouvez-les, corrigez-les, puis comparez avec le verdict de `shellcheck`.

```
#!/bin/bash

# Sauvegarde d'un dossier

DEST = /var/backups
SOURCE=$1

if [ $# == 1 ]; then
    cp -r $SOURCE $DEST
    echo Sauvegarde de $SOURCE
fi

rm $DEST/old.tar.gz

tar czf $DEST/backup.tar.gz $SOURCE
```

Exercice 2 — Compter des fichiers

Écrivez `compte.sh DOSSIER`, qui affiche pour le dossier donné : le nombre total de fichiers, les cinq extensions les plus fréquentes, la taille totale, et le plus gros fichier. Le script vérifie son argument et affiche un message d'usage.

Exercice 3 — Réécrire

Ce script fonctionne, mais il est mal écrit. Réécrivez-le.

```
#!/bin/bash
for f in `ls *.log`; do
    echo Processing $f
    cat $f | grep ERROR | wc -l > $f.errors
    if [ $? -eq 0 ]; then
        echo OK
    fi
done
```

Exercice 4 — Surveiller un service

Écrivez `surveille.sh`, qui vérifie à intervalle régulier qu'un service tourne, avec `systemctl is-active`. Options : `-s` pour le nom du service (obligatoire), `-i` pour l'intervalle en secondes (30 par défaut), `-h` pour l'aide. Chaque vérification écrit une ligne datée.

Exercice 5 — Un script d'installation

Écrivez `prepare-poste.sh`, que l'on doit pouvoir lancer dix fois de suite sans erreur ni doublon. Il installe `git`, `curl`, `vim` et `tmux`, crée `~/bin` et `~/projets`, et ajoute la ligne `export EDITOR=vim` à `~/.profile` si elle n'y figure pas déjà.

Pour aller plus loin

Références. `mywiki.wooledge.org/BashGuide` est le meilleur cours en ligne sur Bash. Sa page `BashPitfalls` recense une soixantaine d'erreurs classiques, avec leur correction. Le guide de style de Google, sur `google.github.io/styleguide/shellguide.html`, donne des conventions éprouvées.

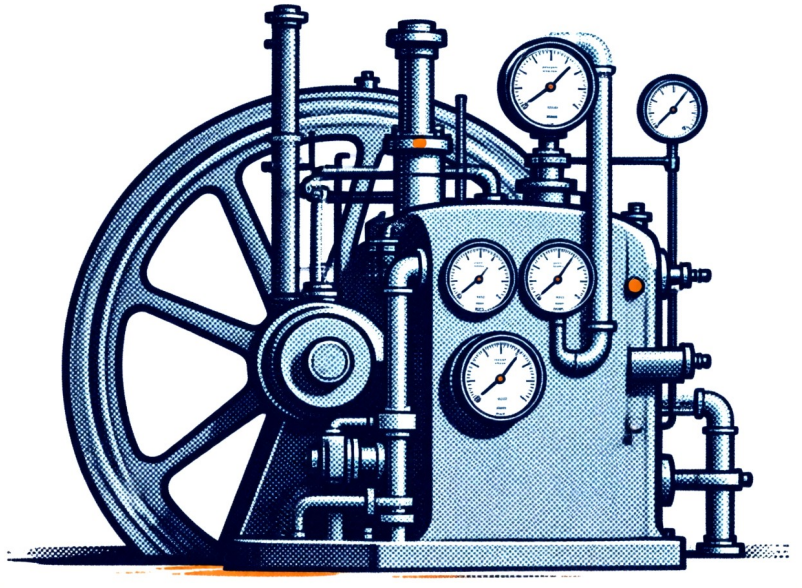
Outils. `shellcheck` pour le contrôle, `shfmt` pour la mise en forme, `bats` pour écrire des tests.

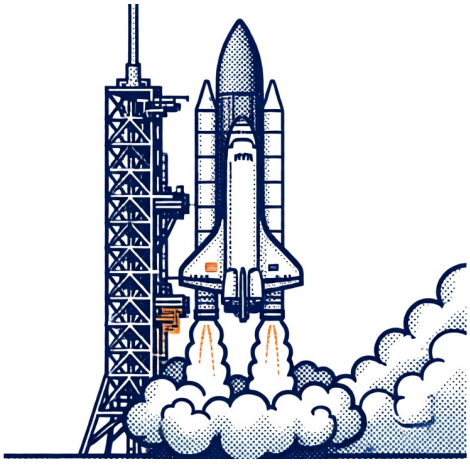
Lectures. *Classic Shell Scripting*, d'Arnold Robbins et Nelson Beebe (O'Reilly). *Wicked Cool Shell Scripts*, de Dave Taylor et Brandon Perry (No Starch Press), pour une centaine de scripts commentés.

PARTIE IV

systemd et gestion du système

Ouvrir le capot d'un Ubuntu moderne : le démarrage, les processus, les services, les journaux et les tâches planifiées.





CHAPITRE 11

Le démarrage du système

De la mise sous tension à l'invite de connexion, six étapes se passent le relais. Les connaître, c'est savoir où chercher le jour où la machine ne démarre plus.

DANS CE CHAPITRE

- Décrire, dans l'ordre, les étapes du démarrage d'Ubuntu.
- Distinguer BIOS et UEFI, et comprendre le rôle de Secure Boot.
- Configurer GRUB et passer des paramètres au noyau.
- Expliquer à quoi sert l'initramfs et le régénérer.
- Lister, charger et bloquer des modules du noyau.
- Changer de cible systemd et démarrer en mode de secours.
- Mesurer le temps de démarrage et trouver ce qui le ralentit.
- Réparer un chargeur d'amorçage depuis une clé d'installation.

> **AVANT DE COMMENCER**
>
> Les chapitres 4 à 6. Faites les manipulations dans une machine virtuelle, après avoir pris un instantané : on touche ici à ce qui permet au système de démarrer. Les sorties de commandes ont été relevées sur une machine Ubuntu 24.04 démarrée en UEFI.

11.1 Les six étapes du démarrage

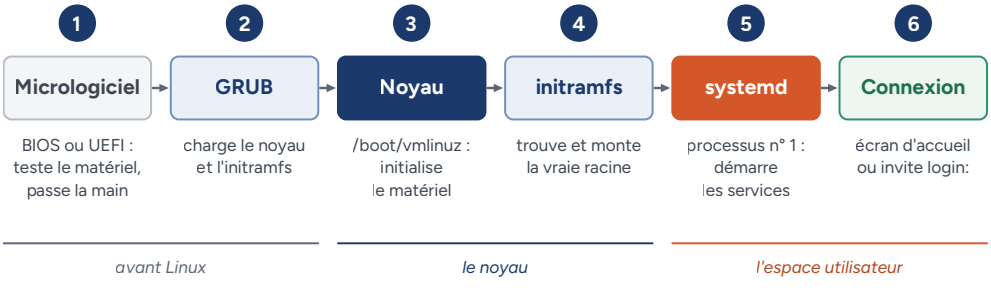


Figure 11.1 — Du bouton d'allumage à l'invite de connexion.

1. Le micrologiciel

À la mise sous tension, le processeur exécute le programme inscrit sur la carte mère : le *micrologiciel* (en anglais *firmware*). Il vérifie le matériel, initialise la mémoire et les contrôleurs, puis cherche un système à démarrer.

Il en existe deux générations. Le **BIOS**, hérité du premier PC d'IBM en 1981, lit le tout premier secteur du disque et exécute le petit programme qu'il y trouve. L'**UEFI**, qui l'a remplacé à partir de 2012 environ, sait lire un système de fichiers. Il va chercher un fichier `.efi` dans une partition réservée, la *partition système EFI*, montée sous `/boot/efi`.

Pour savoir comment une machine a démarré :

```
gary@serveur:~$ [ -d /sys/firmware/efi ] && echo UEFI || echo BIOS
UEFI
```

2. Le chargeur d'amorçage

Le micrologiciel passe la main à **GRUB**, le chargeur d'amorçage d'Ubuntu. Son travail : afficher un menu si nécessaire, charger en mémoire le noyau et l'initramfs, puis lancer le noyau avec ses paramètres.

3. Le noyau

Le noyau est le fichier `/boot/vmlinuz-…`. Il se décompresse, initialise le processeur et la mémoire, puis détecte le matériel. Ses messages sont consultables après coup avec `sudo dmesg`.

4. L'initramfs

Le noyau doit maintenant monter la partition racine. Or les pilotes nécessaires se trouvent précisément sur cette partition : celui du contrôleur de disque, celui du système de fichiers, et les outils de déchiffrement si le disque est chiffré. Le serpent se mord la queue.

La solution s'appelle l'**initramfs**. C'est un petit système de fichiers complet, rangé dans le fichier `/boot/initrd.img-…`, que GRUB charge en mémoire avec le noyau. Il contient juste de quoi trouver, déverrouiller et monter la vraie racine. Une fois ce travail accompli, il s'efface.

C'est l'initramfs qui vous demande la phrase secrète d'un disque chiffré.

5. systemd

La racine montée, le noyau lance son premier programme, `/sbin/init`. Sur Ubuntu, c'est systemd :

```
gary@serveur:~$ ls -l /sbin/init
lrwxrwxrwx 1 root root 22 Jul 28 16:04 /sbin/init -> ../lib/systemd/systemd
```

Il reçoit le numéro de processus 1. Tous les autres processus descendront de lui. systemd monte les autres systèmes de fichiers, puis démarre les services, en parallèle quand c'est possible, jusqu'à atteindre l'état voulu : la *cible* par défaut.

6. La connexion

Sur Ubuntu Desktop, le gestionnaire de connexion GDM affiche l'écran d'accueil. Sur Ubuntu Server, une invite `login:` apparaît sur la console, et le serveur SSH attend les connexions.

> **À RETENIR · À quoi sert de connaître ces étapes**
> Quand une machine ne démarre pas, le dernier message affiché dit à quelle étape elle s'est arrêtée, et donc où chercher. Pas de menu GRUB : le chargeur d'amorçage est en cause. Un menu, puis « ALERT! UUID=... does not exist » : l'initramfs ne trouve pas la racine. Un démarrage qui s'éternise sur « A start job is running » : un service de systemd attend quelque chose qui ne vient pas.

11.2 GRUB

Les fichiers

Fichier	Rôle
`/etc/default/grub`	Vos réglages. C'est le seul fichier que l'on modifie.
`/etc/grub.d/`	Les scripts qui fabriquent le menu.
`/boot/grub/grub.cfg`	Le menu, produit à partir des deux précédents.
`/boot/efi/EFI/ubuntu/`	Les fichiers de démarrage en UEFI.

Voici `/etc/default/grub` sur une installation neuve :

```
GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`( . /etc/os-release; echo ${NAME:-Ubuntu} ) 2>/dev/null || echo Ubuntu`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

Réglage	Signification
`GRUB_DEFAULT`	L'entrée lancée par défaut. 0 désigne la première.
`GRUB_TIMEOUT_STYLE`	`hidden` cache le menu, `menu` l'affiche.
`GRUB_TIMEOUT`	Le délai avant le démarrage automatique, en secondes.
`GRUB_CMDLINE_LINUX_DEFAULT`	Les paramètres passés au noyau en démarrage normal.
`GRUB_CMDLINE_LINUX`	Les paramètres passés au noyau dans tous les cas, mode de secours compris.

Par défaut, le menu est donc caché, et le système démarre sans attendre. Sur un serveur, je préfère le voir quelques secondes :

```
GRUB_TIMEOUT_STYLE=menu
GRUB_TIMEOUT=5
```

Après toute modification, il faut reconstruire le menu :

```
gary@serveur:~$ sudo update-grub
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.8.0-146-generic
Found initrd image: /boot/initrd.img-6.8.0-146-generic
done
```

ATTENTION · On ne modifie jamais grub.cfg à la main

Ce fichier est réécrit à chaque `update-grub`, et donc à chaque mise à jour du noyau. Vos modifications seraient perdues.

Le menu



Figure 11.2 — Le menu de GRUB sur une machine en UEFI.

Si le menu est caché, on le fait apparaître en appuyant sur Échap pendant le démarrage, ou en maintenant Maj sur une machine en BIOS.

Touche	Effet
↑, ↓, Entrée	Choisir une entrée et la lancer
e	Modifier l'entrée pour ce démarrage seulement
c	Ouvrir la ligne de commande de GRUB

L'entrée *Advanced options for Ubuntu* liste tous les noyaux installés, chacun avec sa variante *recovery mode*. Ubuntu garde toujours le noyau précédent. Si une mise à jour du noyau pose un problème, c'est ici qu'on redémarre sur l'ancien.

Passer des paramètres au noyau

Le noyau reçoit de GRUB une ligne de paramètres. On la consulte ainsi :

```
gary@serveur:~$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.8.0-146-generic root=UUID=4f1c… ro quiet splash
```

`root=` désigne la partition racine. `ro` demande de la monter d'abord en lecture seule, le temps de la vérifier. `quiet` et `splash` masquent les messages derrière un logo.

Pour un essai, on modifie cette ligne au démarrage : touche `e` dans le menu, on se place à la fin de la ligne qui commence par `linux`, on ajoute le paramètre, puis `Ctrl`+`X`. Rien n'est enregistré. Pour un changement durable, on modifie `GRUB_CMDLINE_LINUX_DEFAULT`, puis on lance `update-grub`.

Paramètre	Effet
Retirer `quiet splash`	Affiche tous les messages du démarrage : le premier geste devant une machine qui se bloque
`nomodeset`	Garde un affichage basique ; à essayer quand l'écran reste noir
`systemd.unit=rescue.target`	Démarre en mode de secours
`systemd.unit=multi-user.target`	Démarre sans interface graphique
`init=/bin/bash`	Lance un shell à la place de systemd : le dernier recours

11.3 Secure Boot

Secure Boot est une fonction de l'UEFI. Le micrologiciel n'accepte de lancer que des programmes signés par une autorité qu'il connaît. Le but est d'empêcher un logiciel malveillant de s'installer avant le système d'exploitation, là où aucun antivirus ne le verrait.

Ubuntu fonctionne avec Secure Boot activé. La chaîne est la suivante : le micrologiciel vérifie `shim`, un petit programme signé par Microsoft, dont la clé est présente

sur presque tous les PC. `shim` vérifie GRUB, signé par Canonical. GRUB vérifie le noyau. Le noyau n'accepte que des modules signés.

```
gary@serveur:~$ mokutil --sb-state
SecureBoot enabled
```

C'est avec ce dernier maillon que l'on rencontre Secure Boot. Un module compilé sur place n'est pas signé par Canonical. C'est le cas du pilote Nvidia, des modules de VirtualBox et de tout ce qui passe par DKMS. Ubuntu génère alors une clé propre à votre machine, appelée MOK (*Machine Owner Key*), et vous fait choisir un mot de passe. Au redémarrage suivant, un écran bleu intitulé *Perform MOK management* demande ce mot de passe, pour enregistrer la clé dans le micrologiciel. L'opération ne se fait qu'une fois.

Laissez Secure Boot activé. Les tutoriels qui recommandent de le couper datent d'une époque révolue.

11.4 L'initramfs

```
gary@serveur:~$ ls -lh /boot/initrd.img-$(uname -r)
-rw-r--r-- 1 root root 76M Sep 12 09:31 /boot/initrd.img-6.8.0-146-generic
gary@serveur:~$ lsinitramfs /boot/initrd.img-$(uname -r) | wc -l
2025
```

Un initramfs pèse plusieurs dizaines de mégaoctets et contient des milliers de fichiers : des modules du noyau, quelques commandes de base et des scripts. `lsinitramfs` en affiche la liste.

Ubuntu le reconstruit seul à chaque mise à jour du noyau. Vous devrez le faire vous-même après avoir modifié ce qu'il contient : un module à bloquer, la configuration d'un disque chiffré, un fichier de `/etc/initramfs-tools/`.

```
sudo update-initramfs -u            # pour le noyau en cours
sudo update-initramfs -u -k all     # pour tous les noyaux installés
```

> **DANGER · /boot plein, la panne sournoise**
>
> Quand la partition `/boot` est pleine, la reconstruction de l'initramfs échoue au milieu d'une mise à jour, et le nouveau noyau se retrouve avec un initramfs incomplet. Surveillez `df -h /boot`, et lancez régulièrement `sudo apt autoremove` pour retirer les anciens noyaux. Si le mal est fait, redémarrez sur l'ancien noyau par le menu de GRUB.

11.5 Le noyau et ses modules

Les noyaux installés

```
gary@serveur:~$ uname -r
6.8.0-146-generic
gary@serveur:~$ dpkg -l 'linux-image-*' | grep ^ii | awk '{print $2}'
linux-image-6.8.0-141-generic
linux-image-6.8.0-146-generic
linux-image-generic
```

`linux-image-generic` est un paquet vide. Son seul rôle est de dépendre du noyau le plus récent : c'est grâce à lui que les nouveaux noyaux arrivent avec les mises à jour. Ne le supprimez pas.

Ubuntu propose deux séries de noyaux pour une version LTS. La série d'origine, 6.8 pour la 24.04, est maintenue pendant toute la vie de la version. C'est celle d'Ubuntu Server. La série HWE (*Hardware Enablement*) suit les noyaux des versions plus récentes d'Ubuntu, pour prendre en charge le matériel nouveau. C'est celle d'Ubuntu Desktop. On passe à la seconde avec `sudo apt install linux-generic-hwe-24.04`.

> **DANGER · Ne supprimez jamais le noyau en cours d'utilisation**
>
> `sudo apt purge linux-image-$(uname -r)` retire le noyau sur lequel la machine tourne. Si c'était le dernier, elle ne redémarrera pas. Pour faire de la place, laissez faire `sudo apt autoremove` : il sait lesquels garder.

Les modules

La plupart des pilotes ne font pas partie du noyau proprement dit. Ce sont des *modules*, rangés dans `/lib/modules/`, que le noyau charge quand il détecte le matériel correspondant.

```
# Les modules chargés
lsmod | head -n 4
Module                  Size  Used by
tcp_diag               20480  0
inet_diag              20480  1 tcp_diag
nf_conntrack_netlink   57344  0

# La description d'un module
modinfo -F description kvm
Kernel-based Virtual Machine (KVM) Hypervisor

# Charger un module, puis le décharger
sudo modprobe nbd
sudo modprobe -r nbd

# Les modules à charger à chaque démarrage
cat /etc/modules
```

Pour empêcher le chargement d'un module, on l'inscrit sur une liste noire, dans un fichier de `/etc/modprobe.d/`, puis on reconstruit l'initramfs. Le chapitre 20 s'en sert pour désactiver le stockage USB sur un serveur.

11.6 Les cibles de systemd

Une *cible* (en anglais *target*) désigne un état du système : un ensemble de services qui doivent tourner. Au démarrage, systemd cherche à atteindre la cible par défaut, et démarre tout ce qu'elle exige.

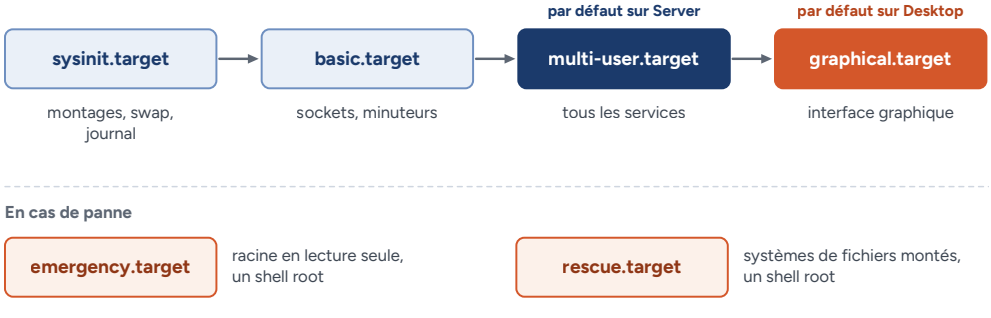


Figure 11.3 — Les principales cibles et leur enchaînement.

Cible	État du système
`graphical.target`	Tous les services et l'interface graphique. Par défaut sur Desktop.

Cible	État du système
`multi-user.target`	Tous les services, sans interface graphique. Par défaut sur Server.
`rescue.target`	Mode de secours : systèmes de fichiers montés, presque aucun service, un shell `root`.
`emergency.target`	Encore moins : la racine en lecture seule et un shell.
`reboot.target`, `poweroff.target`	Redémarrage, extinction.

```
# Cible par défaut
systemctl get-default
graphical.target

# Changer (utile pour transformer un Desktop en Server CLI)
sudo systemctl set-default multi-user.target

# Cible actuelle (peut différer si on a changé sans redémarrer)
systemctl is-active graphical.target

# Lister toutes les targets actives
systemctl list-units --type=target

# Détails d'une target
systemctl status multi-user.target
```

On change de cible à chaud avec `systemctl isolate` :

```
# Aller en mode texte (multi-user) sans redémarrer
sudo systemctl isolate multi-user.target
# 'isolate' = activer cette target et arrêter ce qui n'est pas dépendant

# Revenir au mode graphique
sudo systemctl isolate graphical.target

# Aller en rescue (demande mot de passe root)
sudo systemctl rescue

# Aller en emergency
sudo systemctl emergency
```

`sudo systemctl set-default multi-user.target` transforme un poste de travail en serveur sans écran graphique, sans rien désinstaller.

> **REPÈRE HISTORIQUE · Les niveaux d'exécution**
>
> Avant systemd, le système d'init comptait sept *runlevels*, numérotés de 0 à 6 : 0 pour l'arrêt, 1 pour le mode mono-utilisateur, 3 pour le mode texte, 5 pour le mode graphique, 6 pour le redémarrage. Vous croiserez encore ce vocabulaire dans d'anciennes documentations. Les cibles en sont les héritières.

11.7 Démarrer en mode de secours

Par le menu de GRUB

Dans le menu, choisissez *Advanced options for Ubuntu*, puis l'entrée *recovery mode* du noyau voulu. Un menu en mode texte propose alors :

Entrée	Effet
`resume`	Reprendre le démarrage normal
`clean`	Libérer de la place sur le disque
`dpkg`	Réparer des paquets restés à moitié installés
`fsck`	Vérifier les systèmes de fichiers
`grub`	Mettre à jour le chargeur d'amorçage
`network`	Activer le réseau
`root`	Ouvrir un shell `root`

Dans ce mode, la racine est montée en lecture seule. Pour modifier un fichier depuis le shell `root`, remontez-la d'abord en écriture : `mount -o remount,rw /`.

Par un paramètre du noyau

On obtient le même shell en ajoutant `systemd.unit=rescue.target` à la ligne du noyau, comme expliqué plus haut. Sur Ubuntu, où `root` n'a pas de mot de passe, il suffit d'appuyer sur Entrée pour obtenir le shell.

> **ATTENTION · Qui a la machine en main a tous les droits**
>
> Ces modes donnent un accès complet à quiconque se trouve devant le clavier. C'est ce qui permet de réparer un système ou de retrouver un mot de passe oublié, comme le montre le chapitre 31. C'est aussi ce qui rend un portable non chiffré si vulnérable. Seul le chiffrement du disque protège des données contre quelqu'un qui tient la machine entre ses mains.

11.8 Mesurer le temps de démarrage

```
gary@vostro:~$ systemd-analyze
Startup finished in 5.873s (firmware) + 16.835s (loader) + 2.632s (kernel) +
10.075s (userspace) = 35.417s
graphical.target reached after 10.049s in userspace.
```

Quatre durées : le micrologiciel, le chargeur d'amorçage, le noyau avec l'initramfs, puis les services. Ici, le chargeur d'amorçage a pris près de 17 secondes. C'est le temps passé devant la demande de phrase secrète du disque chiffré.

Quels services ont été les plus longs à démarrer ?

```
gary@vostro:~$ systemd-analyze blame | head -n 5
5.861s NetworkManager-wait-online.service
2.483s apt-daily-upgrade.service
2.439s plymouth-quit-wait.service
1.342s snapd.seeded.service
1.274s docker.service
```

Ce classement est trompeur. Les services démarrent en parallèle, et un service lent ne retarde le démarrage que si d'autres l'attendent. La bonne question est : qu'est-ce qui a déterminé l'heure d'arrivée ?

```
gary@vostro:~$ systemd-analyze critical-chain
graphical.target @10.049s
└─multi-user.target @10.049s
  └─docker.service @8.774s +1.274s
    └─network-online.target @8.757s
      └─NetworkManager-wait-online.service @2.894s +5.861s
        └─NetworkManager.service @2.237s +640ms
          └─dbus.service @2.028s +206ms
            └─basic.target @1.948s
```

On lit la chaîne de bas en haut. Le chiffre après `@` est le moment où l'unité a démarré, celui après `+` le temps qu'elle a mis. Tout a attendu `NetworkManager-wait-online`, qui a patienté près de six secondes en attendant que le réseau soit prêt. Docker ne pouvait pas démarrer avant.

`systemd-analyze plot > demarrage.svg` produit un diagramme de tout le démarrage, à ouvrir dans un navigateur.

Sur un serveur, on ne cherche pas à gagner trois secondes. Mais ces commandes servent le jour où un démarrage prend soudain deux minutes : elles désignent le coupable.

En résumé

- Le démarrage enchaîne micrologiciel, GRUB, noyau, initramfs, systemd, puis la connexion.
- On règle GRUB dans `/etc/default/grub`, puis on lance `sudo update-grub`.
- La touche `e` du menu permet de modifier les paramètres du noyau pour un seul démarrage.
- Secure Boot reste activé. Les modules compilés sur place sont signés avec une clé propre à la machine.
- L'initramfs contient de quoi monter la racine. On le reconstruit avec `update-initramfs -u`.
- Ubuntu garde le noyau précédent : le menu de GRUB permet d'y revenir.
- `graphical.target` et `multi-user.target` sont les deux cibles courantes. `systemctl set-default` choisit celle du démarrage.
- Le mode de secours donne un shell `root` sans mot de passe à qui tient le clavier.
- `systemd-analyze critical-chain` montre ce qui retarde le démarrage.

Atelier — Réparer un chargeur d'amorçage

Objectif : casser le démarrage d'une machine virtuelle, puis le réparer depuis l'image d'installation. Comptez quarante-cinq minutes.

La méthode sert bien au-delà de GRUB. Démarrer sur un support externe, monter le système en panne et « entrer dedans » avec `chroot`, c'est la technique de base de toute réparation.

Préparer

1. Utilisez une machine virtuelle Ubuntu 24.04 démarrée en UEFI, sans chiffrement du disque. Vérifiez avec `[ -d /sys/firmware/efi ] && echo UEFI`.
2. Notez le découpage du disque avec `lsblk -f`. Repérez la partition EFI, de type `vfat`, et la partition racine, de type `ext4`.
3. Éteignez la machine et prenez un instantané.

Casser

4. Redémarrez la machine, puis supprimez les fichiers de démarrage d'Ubuntu :

```
sudo rm -rf /boot/efi/EFI/ubuntu
sudo reboot
```

La machine ne trouve plus de système. Elle affiche un message d'erreur, ou le menu du micrologiciel.

Démarrer sur l'image d'installation

5. Attachez l'image d'Ubuntu au lecteur virtuel, démarrez dessus, choisissez *Try Ubuntu* et ouvrez un terminal. Sur l'image Server, Ctrl+Alt+F2 ouvre une console.

Monter le système en panne

6. Montez la racine, puis la partition EFI. Adaptez les noms à ce que montre `lsblk` :

```
sudo mount /dev/sda2 /mnt
sudo mount /dev/sda1 /mnt/boot/efi
```

7. Donnez au système monté l'accès au matériel, puis entrez-y :

```
for d in dev dev/pts proc sys sys/firmware/efi/efivars run; do
    sudo mount --bind /$d /mnt/$d
done
sudo chroot /mnt
```

À partir de là, les commandes s'exécutent dans le système installé, et non dans celui de la clé. Vérifiez-le avec `cat /etc/hostname`.

Réparer

8. Réinstallez GRUB, puis reconstruisez son menu :

```
grub-install
update-grub
```

9. Sortez avec `exit`, détachez l'image du lecteur et redémarrez.

Vous avez réussi si la machine démarre normalement. Sinon, revenez à l'instantané et recommencez. C'est ainsi qu'on apprend.

> **NOTE · Et sur une machine en BIOS, ou chiffrée ?**
> En BIOS, il n'y a pas de partition EFI. On indique le disque à `grub-install` : `grub-install /dev/sda`. Avec un disque chiffré ou des volumes LVM, il faut les ouvrir avant de les monter. Le chapitre 31 traite ces cas.

Exercices

Exercice 1 — À quelle étape ?

Pour chaque symptôme, dites quelle étape du démarrage est en cause.

1. L'écran affiche « No bootable device » juste après le logo du constructeur.
2. Le menu de GRUB s'affiche, puis « Kernel panic — not syncing: VFS: Unable to mount root fs ».
3. On lit « ALERT! UUID=... does not exist. Dropping to a shell! » suivi d'une invite `(initramfs)`.
4. Le démarrage reste bloqué une minute et demie sur « A start job is running for... ».
5. Tout démarre, mais l'écran de connexion graphique n'apparaît pas. La connexion par SSH fonctionne.

Exercice 2 — Régler GRUB

1. Faites en sorte que le menu s'affiche pendant cinq secondes à chaque démarrage.
2. Retirez `quiet splash` pour voir les messages du démarrage.
3. Appliquez, redémarrez et vérifiez avec `cat /proc/cmdline`.

Exercice 3 — Les cibles

1. Quelle est la cible par défaut de votre machine ?
2. Sur Ubuntu Desktop, passez à `multi-user.target` sans redémarrer. Que devient la session graphique ? Revenez ensuite à `graphical.target`.
3. Démarrez une fois en mode de secours par un paramètre du noyau, sans rien modifier de façon durable.

Exercice 4 — Le temps de démarrage

1. Combien de temps votre machine met-elle à démarrer ? Quelle part revient aux services ?
2. Quel service est en tête de `systemd-analyze blame` ? Figure-t-il dans la chaîne critique ?
3. Produisez le diagramme du démarrage et ouvrez-le.

Exercice 5 — Les modules

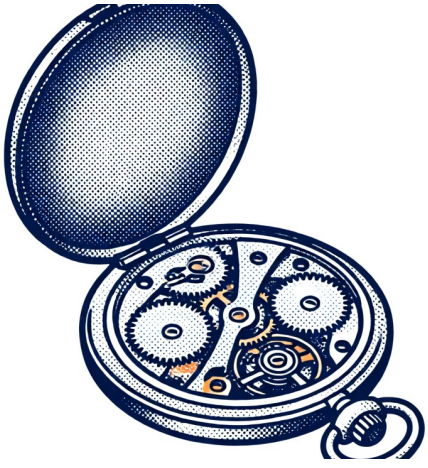
1. Combien de modules sont chargés ?
2. Affichez la description du module `kvm`.
3. Empêchez le chargement de `pcspkr`, le pilote du haut-parleur interne. Reconstruisez l'initramfs, redémarrez et vérifiez.

Pour aller plus loin

Sur la machine. `man bootup` décrit l'enchaînement des cibles de systemd au démarrage, schéma à l'appui. `man kernel-command-line` liste les paramètres compris par systemd. `man 7 bootparam` donne ceux du noyau.

Outils. `efibootmgr` affiche et modifie les entrées de démarrage de l'UEFI. `sudo dmesg` relit les messages du noyau, et `journalctl -b` tous ceux du dernier démarrage.

Lectures. *How Linux Works*, de Brian Ward (No Starch Press) : ses chapitres sur le démarrage sont les plus clairs que je connaisse. La page `wiki.archlinux.org/title/Arch_boot_process` détaille chaque étape, et presque tout s'y applique à Ubuntu.



CHAPITRE 12

Processus et services

Tout ce qui tourne sur la machine est un processus. Ce chapitre apprend à les observer et à leur parler, puis à confier à systemd ceux qui doivent tourner jour et nuit.

DANS CE CHAPITRE

- Lire la fiche d'un processus et le situer dans l'arbre.
- Observer la machine avec `ps`, `pstree`, `top` et `htop`.
- Arrêter un processus proprement, et savoir quand passer en force.
- Baisser la priorité d'un programme gourmand.
- Lancer une commande en arrière-plan et la faire survivre à la déconnexion.
- Piloter les services avec `systemctl`.
- Écrire une unité de service pour votre propre programme.
- Cloisonner un service et plafonner ses ressources.

> **AVANT DE COMMENCER**
>
> Les chapitres 6, 8 et 11. Travaillez dans une machine virtuelle Ubuntu Server, sur laquelle vous installerez le serveur web nginx : il sert d'exemple tout au long du chapitre (`sudo apt install nginx`). Les sorties ont été relevées sur un serveur Ubuntu 24.04 à deux processeurs, au cours d'une session SSH.

12.1 Qu'est-ce qu'un processus ?

Un programme est un fichier rangé sur le disque. Un **processus** est ce programme en train de s'exécuter : du code chargé en mémoire, des données, des fichiers ouverts, une identité. Le même programme peut tourner dix fois au même moment. Cela fait dix processus.

Le noyau donne à chacun un numéro, le **PID** (*process identifier*), et tient à jour sa fiche :

Attribut	Ce qu'il désigne
PID	Le numéro du processus
PPID	Le PID de son parent, le processus qui l'a créé
Utilisateur, groupe	L'identité au nom de laquelle il agit (chapitre 6)
État	En exécution, en attente, suspendu…
Priorité	Sa place dans la file d'attente du processeur
Mémoire	Ce qu'il a réservé (VSZ) et ce qu'il occupe vraiment en mémoire vive (RSS)
Répertoire courant	Celui d'où il a été lancé, sauf s'il en a changé
Environnement	Ses variables d'environnement (chapitre 8)
Fichiers ouverts	Fichiers, tubes, connexions réseau

`ps` affiche cette fiche. Essayez sur votre shell, dont le PID se trouve dans la variable `$$` :

```
gary@serveur:~$ ps -o pid,ppid,user,stat,ni,vsz,rss,comm -p $$
    PID    PPID USER     STAT  NI    VSZ   RSS COMMAND
    374     373 gary     Ss     0   5028  4244 bash
```

Le noyau publie aussi ces informations sous forme de fichiers. Chaque processus a son dossier dans `/proc` : `/proc/374/cwd` pointe vers le répertoire courant du processus 374, `/proc/374/environ` contient son environnement, `/proc/374/fd` la liste de ses fichiers ouverts. `ps` et `top` ne font que lire `/proc`.

L'arbre des processus

Un processus naît toujours d'un autre. Le parent se dédouble, puis la copie remplace son programme par celui qu'on lui demande d'exécuter. C'est ce que fait votre shell chaque fois que vous tapez une commande.

De parent en enfant, les processus forment un arbre. À sa racine se trouve le PID 1, que le noyau lance lui-même à la fin du démarrage : systemd, rencontré au chapitre 11.

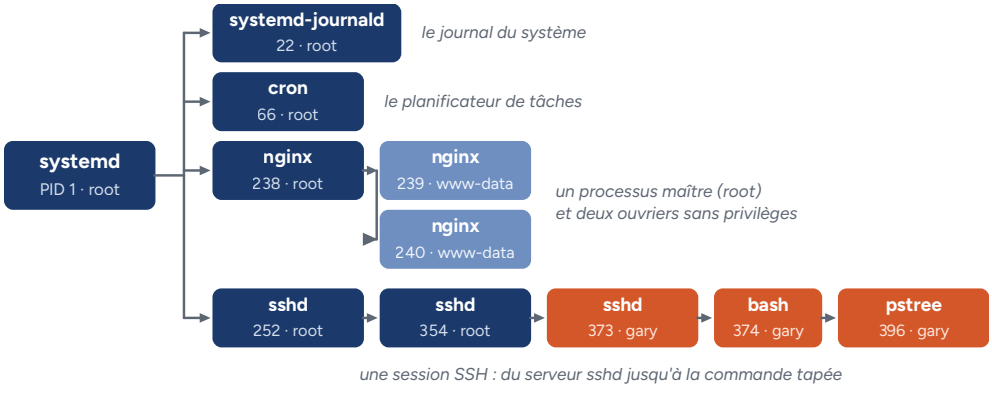


Figure 12.1 — L'arbre des processus d'un petit serveur. Chaque case donne le nom, le PID et l'utilisateur.

`pstree` dessine cet arbre :

```
gary@serveur:~$ pstree
systemd─┬─atd
        ├─cron
        ├─dbus-daemon
        ├─nginx───2*[nginx]
        ├─rsyslogd───3*[{rsyslogd}]
        ├─sshd───sshd───sshd───bash───pstree
        ├─systemd───(sd-pam)
        ├─systemd-journal
        └─systemd-logind
```

`2*[nginx]` signifie deux enfants identiques. Les noms entre accolades sont des fils d'exécution (*threads*) : plusieurs tâches menées de front à l'intérieur d'un même processus.

Avec `-p`, `pstree` ajoute les PID. Avec `-s`, il remonte d'un processus jusqu'à la racine :

```
gary@serveur:~$ pstree -ps $$
systemd(1)───sshd(252)───sshd(354)───sshd(373)───bash(374)───pstree(396)
```

On lit ici toute l'histoire d'une connexion SSH. Le serveur `sshd` (252) attend les connexions. Il a créé un premier enfant (354) pour gérer celle de Gary avec les droits de `root`, puis un second (373) qui travaille avec les droits de Gary. Ce dernier a lancé le shell (374), et le shell la commande `pstree`.

> **NOTE · Orphelins et zombies**
>
> Si un parent meurt avant ses enfants, ceux-ci ne restent pas orphelins : systemd les adopte.
>
> À l'inverse, un processus qui se termine ne disparaît pas tout de suite. Il reste inscrit dans la table des processus jusqu'à ce que son parent ait lu son code de retour. Entre-temps, c'est un *zombie*. Il ne consomme plus ni mémoire ni processeur, et `kill` ne peut rien contre lui, puisqu'il est déjà mort. Un zombie de passage est normal. Des zombies qui s'accumulent trahissent un parent mal écrit : c'est ce parent qu'il faut redémarrer.

12.2 Observer les processus

ps, la photographie

Sans option, `ps` ne montre que les processus de votre terminal :

```
gary@serveur:~$ ps
    PID TTY          TIME CMD
    374 pts/0    00:00:00 bash
    397 pts/0    00:00:00 ps
```

Pour tout voir, deux écritures cohabitent, héritées de deux familles d'Unix : `ps aux`, sans tiret, et `ps -ef`. Elles listent les mêmes processus avec des colonnes différentes. La première donne la consommation de processeur et de mémoire, la seconde le PID du parent.

```
gary@serveur:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0  21296 12672 ?        Ss   20:28   0:00 /sbin/init
root          22  0.0  0.1  50380 17016 ?        S<s  20:28   0:00 /usr/lib/sy
root          66  0.0  0.0   4248  2764 ?        Ss   20:28   0:00 /usr/sbin/c
[…]
root         238  0.0  0.0  11220  1776 ?        Ss   20:29   0:00 nginx: mast
www-data     239  0.0  0.0  11600  3028 ?        S    20:29   0:00 nginx: work
www-data     240  0.0  0.0  11600  2964 ?        S    20:29   0:00 nginx: work
[…]
gary         374  0.0  0.0   5028  4244 pts/0    Ss   20:35   0:00 -bash
gary         398  0.0  0.0   8296  4392 pts/0    R+   20:35   0:00 ps aux
```

`ps` coupe les lignes à la largeur du terminal. Ajoutez `ww` (`ps auxww`) pour les lire en entier. Sur votre machine, la liste commence par des dizaines de noms entre crochets, comme `[kthreadd]` : ce sont les tâches internes du noyau.

Colonne	Signification
`USER`, `PID`	Le propriétaire et le numéro du processus
`%CPU`, `%MEM`	La part de processeur et de mémoire vive qu'il utilise
`VSZ`, `RSS`	La mémoire réservée et la mémoire réellement occupée, en kilooctets
`TTY`	Son terminal. Un `?` signale un processus sans terminal : un service
`STAT`	Son état (voir ci-dessous)
`START`, `TIME`	L'heure de son lancement, et le temps de processeur consommé depuis
`COMMAND`	La commande qui l'a lancé

La première lettre de `STAT` donne l'état du processus :

Lettre	État
`R`	En cours d'exécution, ou prêt à s'exécuter
`S`	En attente d'un événement : une frappe au clavier, une connexion, la fin d'un délai
`D`	En attente d'une lecture ou d'une écriture sur disque, sans interruption possible
`T`	Suspendu
`Z`	Zombie
`I`	Tâche du noyau au repos

Les caractères suivants apportent des précisions : `s` pour le chef d'une session, `+` pour un processus au premier plan, `<` et `N` pour une priorité relevée ou abaissée, `l` pour un processus à plusieurs fils d'exécution.

Presque tous les processus sont à l'état `S`. C'est normal : un serveur passe l'essentiel de son temps à attendre qu'on lui demande quelque chose.

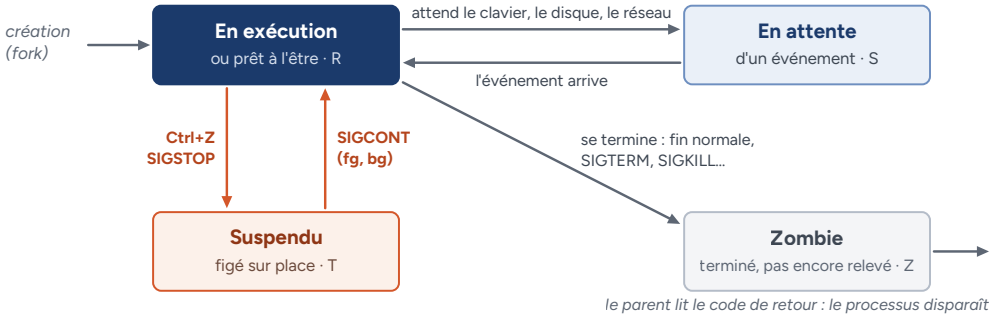


Figure 12.2 — Les états d'un processus et les événements qui le font passer de l'un à l'autre.

Quelques formes de `ps` rendent service tous les jours :

```
ps -ef                              # tous les processus, avec le PID du parent
ps -u gary                          # ceux d'un utilisateur
ps -C nginx -o pid,ppid,user,cmd    # ceux d'une commande, colonnes au choix

# les plus gros en mémoire, puis les plus gourmands en calcul
ps -eo pid,user,%cpu,%mem,rss,comm --sort=-rss | head
ps -eo pid,user,%cpu,comm --sort=-%cpu | head
```

Pour retrouver un PID d'après un nom, `pgrep` remplace avantageusement le vieux `ps aux | grep nginx`, qui trouve aussi la commande `grep` elle-même :

```
gary@serveur:~$ pgrep -a nginx
238 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
239 nginx: worker process
240 nginx: worker process
```

top, le film

`ps` prend une photo. `top` affiche la même liste, rafraîchie toutes les trois secondes et triée par consommation de processeur. Ses premières lignes résument l'état de la machine :

```
gary@serveur:~$ top
top - 20:35:55 up 1 day, 13:27,  1 user,  load average: 0.63, 0.48, 0.50
Tasks:  18 total,   1 running,  17 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.8 sy,  0.0 ni, 98.5 id,  0.8 wa,  0.0 hi,  0.0 si,  0.0 st
[…]
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0   21296  12672   9808 S   0.0   0.1   0:00.23 systemd
     22 root      19  -1   50380  17016  15956 S   0.0   0.1   0:00.09 systemd+
```

La **charge moyenne** (*load average*) est le nombre moyen de processus qui utilisent un processeur ou en attendent un, disque compris, sur la dernière minute, les cinq dernières et les quinze dernières. Elle se compare au nombre de processeurs, que donne `nproc`. Sur une machine à deux processeurs, une charge de 2 veut dire qu'elle travaille à plein. Au-delà, des processus font la queue.

La ligne `%Cpu(s)` répartit le temps du processeur : `us` pour les programmes, `sy` pour le noyau, `id` pour le temps inoccupé, `wa` pour l'attente du disque. Un `wa` élevé désigne un disque qui ne suit pas. Les deux lignes consacrées à la mémoire, coupées ici, seront lues en détail au chapitre 26.

Touche	Effet dans `top`
P, M	Trier par processeur, par mémoire
1	Détailler chaque processeur
c	Afficher la ligne de commande complète
u	Ne garder que les processus d'un utilisateur
V	Afficher l'arbre
k, r	Envoyer un signal, changer la priorité
q	Quitter

Dans un script, `top -b -n 1` écrit un seul relevé sur la sortie standard.

htop et btop

`htop` montre la même chose en plus lisible : une barre par processeur, le défilement, la souris. F5 affiche l'arbre, F4 filtre, F6 change le tri, F9 envoie un signal choisi dans un menu, F10 quitte. Il est installé d'origine sur Ubuntu Server. Sur Desktop, `sudo apt install htop`.

`btop` va plus loin dans le tableau de bord, avec des courbes pour le processeur, la mémoire, les disques et le réseau. Il vient du dépôt `universe` : `sudo apt install btop`.

Besoin	Outil
Un relevé dans un script, un filtre précis	`ps`, `pgrep`
Comprendre qui a lancé quoi	`pstree`
Surveiller une machine où rien d'autre n'est installé	`top`
Surveiller et agir au quotidien	`htop`

12.3 Les signaux

Un **signal** est un message très court envoyé à un processus : un simple numéro. `kill -l` en liste une soixantaine. Voici ceux qui servent :

N°	Nom	Effet par défaut	Qui l'envoie
15	`SIGTERM`	Arrêt	`kill` sans option, `systemctl stop`
9	`SIGKILL`	Arrêt immédiat, impossible à intercepter	`kill -9`
2	`SIGINT`	Arrêt	Ctrl+C
1	`SIGHUP`	Arrêt	Le terminal qui se ferme
20	`SIGTSTP`	Suspension	Ctrl+Z
19	`SIGSTOP`	Suspension, impossible à intercepter	`kill -STOP`
18	`SIGCONT`	Reprise	`fg`, `bg`, `kill -CONT`
10, 12	`SIGUSR1`, `SIGUSR2`	Arrêt	Libres : chaque programme en fait ce qu'il veut

Un programme peut intercepter un signal et décider de sa réaction. Beaucoup de services détournent ainsi `SIGHUP` : quand ils le reçoivent, ils relisent leur configuration au lieu de s'arrêter. Deux signaux échappent à cette règle, `SIGKILL` et `SIGSTOP`. Le noyau les applique lui-même, sans consulter le processus.

La commande `kill` envoie un signal à un PID. Son nom est trompeur : elle ne tue que si le signal le demande.

```
gary@serveur:~$ sleep 600 &
[1] 409
gary@serveur:~$ kill 409
gary@serveur:~$
[1]+  Terminated              sleep 600
gary@serveur:~$ sleep 600 &
[1] 410
gary@serveur:~$ kill -9 410
gary@serveur:~$
[1]+  Killed                  sleep 600
```

On désigne le signal par son numéro ou par son nom : `kill -9`, `kill -KILL` et `kill -s KILL` sont équivalents. Vous ne pouvez envoyer de signaux qu'à vos propres processus. Pour ceux des autres, il faut `sudo`.

> **À RETENIR · SIGTERM d'abord, SIGKILL en dernier**
>
> `SIGTERM` demande au programme de s'arrêter. Il a le temps de finir proprement : écrire ses données, fermer ses fichiers, effacer ses fichiers temporaires. `SIGKILL` ne lui demande rien. Le noyau supprime le processus sur-le-champ, et ce qui était à moitié écrit le reste.
>
> Envoyez `SIGTERM`, laissez quelques secondes, et ne sortez `kill -9` que si rien ne bouge. systemd applique la même règle quand il arrête un service : `SIGTERM`, puis `SIGKILL` au bout de quatre-vingt-dix secondes.

Un processus tué par un signal renvoie un code de retour égal à 128 plus le numéro du signal : 143 après `SIGTERM`, 137 après `SIGKILL`. Un 137 dans un journal veut dire que le programme a été tué de force, souvent par le noyau lui-même, à court de mémoire.

`pkill` et `killall` visent les processus par leur nom :

```
killall nginx                 # tous les processus nommés exactement nginx
pkill -u gary sleep           # les sleep de l'utilisateur gary
pkill -f 'sauvegarde.sh'      # recherche dans toute la ligne de commande
```

ATTENTION · Regardez avant de tirer

`pkill` prend un motif, pas un nom exact. `pkill ssh` atteint tout ce dont le nom contient « ssh » :

```
gary@serveur:~$ pgrep -a ssh
252 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
354 sshd: gary [priv]
373 sshd: gary@pts/0
```

Le troisième est votre propre connexion. Lancez toujours `pgrep -a` avec le motif avant de le confier à `pkill`.

12.4 Régler les priorités

Quand plusieurs processus réclament le processeur en même temps, le noyau le partage. Chacun porte une valeur de politesse, *nice* en anglais, de −20 à 19. Elle vaut 0 par défaut. Plus elle est haute, plus le processus s'efface devant les autres.

`nice` lance une commande avec une valeur choisie. `renice` modifie celle d'un processus qui tourne déjà.

```
gary@serveur:~$ nice -n 10 sleep 600 &
[1] 411
gary@serveur:~$ ps -o pid,ni,cmd -p 411
    PID  NI CMD
    411  10 sleep 600
gary@serveur:~$ renice 15 -p 411
411 (process ID) old priority 10, new priority 15
gary@serveur:~$ renice 5 -p 411
renice: failed to set priority for 411 (process ID): Permission denied
gary@serveur:~$ sudo renice -5 -p 411
411 (process ID) old priority 15, new priority -5
```

Un utilisateur ordinaire peut seulement augmenter la valeur, c'est-à-dire céder sa place. Pour la diminuer, il faut `sudo`.

L'usage courant consiste à lancer en `nice -n 19` les travaux lourds et peu pressés : une compression, une sauvegarde, une conversion de vidéos. Tant que la machine n'a rien d'autre à faire, ils avancent à pleine vitesse. Dès qu'un autre programme a besoin du processeur, ils lui laissent la place. Dans un essai mené sur un seul processeur avec deux boucles de calcul identiques, celle passée à 19 n'obtenait plus que 1,7 % du temps, et l'autre 98,3 %.

`ionice` joue le même rôle pour les accès au disque. `ionice -c 3 commande` place la commande dans la classe *idle* : elle ne lit et n'écrit que lorsque personne d'autre ne sollicite le disque.

> **SUR LE TERRAIN · ionice n'agit pas partout**
>
> Ces priorités ne sont appliquées que par deux ordonnanceurs de disque, `bfq` et `mq-deadline`. Or les SSD NVMe n'en utilisent aucun par défaut :
>
> ```
> gary@serveur:~$ cat /sys/block/nvme0n1/queue/scheduler
> [none] mq-deadline
> ```
>
> L'ordonnanceur actif est entre crochets. Avec `none`, `ionice` est accepté sans erreur et ne change rien.

12.5 Premier plan et arrière-plan

Une commande lancée normalement occupe le terminal jusqu'à ce qu'elle se termine : elle tourne au premier plan. Suivie de `&`, elle part en arrière-plan et le shell rend la main aussitôt. Le shell appelle *tâches* (*jobs*) les commandes qu'il a lancées, et les numérote.

```
gary@serveur:~$ sleep 300
^Z
[1]+  Stopped                 sleep 300
gary@serveur:~$ bg
[1]+ sleep 300 &
gary@serveur:~$ sleep 400 &
[2] 420
gary@serveur:~$ jobs
[1]-  Running                 sleep 300 &
[2]+  Running                 sleep 400 &
gary@serveur:~$ fg %1
sleep 300
^C
gary@serveur:~$ kill %2
gary@serveur:~$
[2]+  Terminated              sleep 400
```

Commande	Effet
`commande &`	Lancer en arrière-plan
Ctrl+Z	Suspendre la commande du premier plan

Commande	Effet
`jobs`	Lister les tâches du shell
`bg`	Relancer en arrière-plan la tâche suspendue
`fg %2`	Ramener la tâche 2 au premier plan
`kill %2`	Envoyer `SIGTERM` à la tâche 2

Le réflexe à prendre : vous avez lancé une longue commande en oubliant le `&`. Tapez Ctrl+Z, puis `bg`. Elle continue, et vous retrouvez l'invite.

Survivre à la déconnexion

Que deviennent les tâches de fond quand vous quittez le terminal ? Cela dépend de la manière.

Si vous sortez proprement, avec `exit`, Bash les laisse tourner. Si la fenêtre est fermée ou si la connexion SSH tombe, le noyau envoie `SIGHUP` au shell, qui le transmet à toutes ses tâches. Elles s'arrêtent. Sur un serveur distant, une coupure du réseau suffit donc à interrompre une mise à jour ou une copie de plusieurs heures.

Trois protections existent.

`nohup` lance une commande qui ignorera `SIGHUP`. Ce qu'elle aurait affiché à l'écran est écrit dans le fichier `nohup.out` :

```
gary@serveur:~$ nohup ./sauvegarde.sh ~/Documents &
[1] 1435
nohup: ignoring input and appending output to 'nohup.out'
```

`disown` rattrape un oubli. Il retire de la liste du shell une tâche déjà lancée, qui ne recevra donc pas le signal.

`tmux` règle le problème à la racine : il crée un terminal qui vit sur le serveur, indépendamment de votre connexion. Vous pouvez vous en détacher, vous déconnecter, revenir le lendemain depuis un autre poste et le retrouver tel quel. Il est installé d'origine sur Ubuntu Server.

```
tmux new -s travail        # ouvrir une session nommée « travail »
tmux ls                    # lister les sessions
tmux attach -t travail     # revenir dans la session
```

Dans la session, lancez votre commande, puis tapez Ctrl+B suivi de D pour vous détacher. La commande continue sans vous. Sur un serveur, lancez dans `tmux` toute opération longue ou risquée.

Ces trois outils conviennent à une commande ponctuelle. Un programme qui doit tourner en permanence relève d'une autre catégorie : c'est un service.

12.6 systemd et ses unités

Un *service* est un programme qui tourne en arrière-plan, sans terminal, en général depuis le démarrage de la machine : un serveur web, une base de données, le planificateur de tâches. On dit aussi *démon*, de l'anglais *daemon*.

systemd lance les services, les surveille, les relance s'ils tombent, les arrête dans l'ordre et recueille tout ce qu'ils écrivent dans un journal, que nous lirons au chapitre 13.

> **REPÈRE HISTORIQUE · D'init à systemd**
>
> À ses débuts, Ubuntu démarrait avec *SysV init*, hérité d'Unix : des scripts shell exécutés l'un après l'autre. En 2006, elle l'a remplacé par Upstart, écrit chez Canonical par Scott James Remnant. systemd est né en 2010 chez Red Hat. Lennart Poettering en a posé les principes dans un billet intitulé *Rethinking PID 1* : démarrer moins de choses, et les démarrer en parallèle. Debian l'a choisi en février 2014. Mark Shuttleworth a aussitôt annoncé qu'Ubuntu suivrait, dans un billet au titre beau joueur, *Losing graciously*. Depuis la version 15.04, en avril 2015, le PID 1 d'Ubuntu est systemd.

Tout ce que gère systemd s'appelle une **unité**. Chacune est décrite par un fichier texte, dont le suffixe indique le type :

Suffixe	Ce que décrit l'unité
`.service`	Un programme à lancer et à surveiller
`.socket`	Un port en écoute. Le service associé n'est lancé qu'à la première connexion
`.timer`	Une planification (chapitre 13)
`.mount`, `.swap`	Un système de fichiers à monter, un espace d'échange (chapitre 14)
`.target`	Un groupe d'unités : les cibles du chapitre 11
`.slice`, `.scope`	Un groupe de processus dont on mesure et limite les ressources

Sur Ubuntu 24.04, SSH donne un bon exemple du type `.socket`. C'est `ssh.socket` qui écoute sur le port 22, et `ssh.service` ne démarre qu'à l'arrivée de la première connexion.

Les fichiers d'unités sont rangés à trois endroits :

Dossier	Contenu
`/usr/lib/systemd/system/`	Les unités livrées par les paquets. On n'y touche pas : la mise à jour suivante écraserait vos modifications
`/etc/systemd/system/`	Vos unités et vos retouches. Ce dossier l'emporte sur le précédent
`/run/systemd/system/`	Des unités temporaires, perdues au redémarrage

D'anciennes documentations citent `/lib/systemd/system/`. C'est le même dossier : sur Ubuntu, `/lib` est un lien vers `/usr/lib`.

12.7 Piloter les services avec systemctl

Quatre verbes agissent sur un service qui tourne, ou qui devrait tourner :

```
sudo systemctl start nginx       # démarrer
sudo systemctl stop nginx        # arrêter
sudo systemctl restart nginx     # arrêter, puis démarrer
sudo systemctl reload nginx      # relire la configuration sans couper le service
```

Sans suffixe, systemctl comprend `nginx.service`. Ces commandes n'affichent rien quand tout va bien. `reload` n'existe que pour les services qui savent relire leur configuration à chaud. Préférez-le à `restart` quand il est proposé : aucune connexion en cours n'est coupée.

Lire l'état d'un service

```
gary@serveur:~$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Sun 2026-10-04 20:29:18 UTC; 8min ago
       Docs: man:nginx(8)
    Process: 235 ExecStartPre=/usr/sbin/nginx -t -q … (code=exited, status=0/SUCCESS)
    Process: 237 ExecStart=/usr/sbin/nginx … (code=exited, status=0/SUCCESS)
   Main PID: 238 (nginx)
      Tasks: 3 (limit: 2493)
     Memory: 2.2M (peak: 2.7M)
```

```
         CPU: 7ms
      CGroup: /system.slice/nginx.service
              ├─238 "nginx: master process /usr/sbin/nginx -g daemon on; maste…
              ├─239 "nginx: worker process"
              └─240 "nginx: worker process"

Oct 04 20:29:18 serveur systemd[1]: Starting nginx.service - A high perfo…r...
Oct 04 20:29:18 serveur systemd[1]: Started nginx.service - A high perfor…ver.
```

C'est la commande à lancer en premier quand un service pose problème. Elle se lit ainsi :

- **Loaded** donne le fichier de l'unité, puis `enabled` ou `disabled` : le service est-il lancé au démarrage de la machine ?
- **Active** indique s'il tourne en ce moment, et depuis quand. Un service en panne affiche `failed` et une croix rouge à la place du point vert.
- **Main PID**, **Tasks**, **Memory**, **CPU** : le processus principal, le nombre de processus et de fils d'exécution, la mémoire occupée, le temps de calcul consommé.
- **CGroup** liste tous les processus du service.
- Les dernières lignes sont les derniers messages du service. En cas de panne, la cause y figure presque toujours.

Démarrer n'est pas activer

`start` et `stop` agissent sur l'instant. Ils ne disent rien du prochain démarrage de la machine. C'est le rôle de deux autres verbes : `enable` inscrit le service au démarrage, `disable` l'en retire.

```
gary@serveur:~$ sudo systemctl disable nginx
[…]
Removed "/etc/systemd/system/multi-user.target.wants/nginx.service".
gary@serveur:~$ systemctl is-enabled nginx
disabled
gary@serveur:~$ systemctl is-active nginx
active
```

Le service tourne toujours, mais il ne reviendra pas au prochain redémarrage. Activer un service revient à créer un lien dans le dossier de la cible qui doit l'entraîner, ici `multi-user.target`. Le désactiver supprime ce lien.

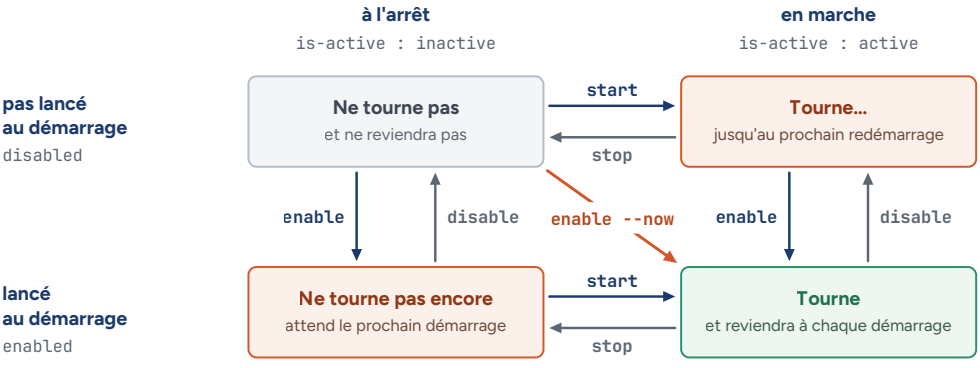


Figure 12.3 — Deux questions indépendantes : le service tourne-t-il maintenant ? Sera-t-il lancé au démarrage ?

> **ATTENTION · L'oubli classique**
>
> Vous installez un service à la main, vous le démarrez avec `start`, tout fonctionne. Trois mois plus tard, la machine redémarre après une mise à jour du noyau, et le service ne revient pas. Prenez l'habitude d'écrire `sudo systemctl enable --now service`, qui active et démarre d'un seul geste. De même, `disable --now` désactive et arrête.

`is-active` et `is-enabled` répondent d'un mot et par leur code de retour. Ils sont faits pour les scripts :

```
if ! systemctl is-active --quiet nginx; then
    echo "nginx est arrêté" >&2
fi
```

Faire l'inventaire

```
gary@serveur:~$ systemctl list-units --type=service --state=running
  UNIT                     LOAD   ACTIVE SUB     DESCRIPTION
  atd.service              loaded active running Deferred execution scheduler
  cron.service             loaded active running Regular background program p…
  dbus.service             loaded active running D-Bus System Message Bus
  nginx.service            loaded active running A high performance web serve…
  rsyslog.service          loaded active running System Logging Service
  ssh.service              loaded active running OpenBSD Secure Shell server
  systemd-journald.service loaded active running Journal Service
  systemd-logind.service   loaded active running User Login Management
```

```
  user@1000.service        loaded active running User Manager for UID 1000
[…]
9 loaded units listed.
```

Commande	Ce qu'elle liste
`systemctl list-units --type=service`	Les services chargés et leur état
`systemctl --failed`	Les unités en échec. À regarder après chaque redémarrage
`systemctl list-unit-files --type=service`	Tous les services installés, activés ou non
`systemctl list-timers`	Les planifications et leur prochaine échéance

Dans `list-unit-files`, la colonne `STATE` contient `enabled` ou `disabled`, mais aussi `static` : l'unité ne peut pas être activée directement, elle est lancée par une autre qui en dépend.

Interdire un service

`disable` n'empêche pas un service d'être démarré à la main, ni par un autre service qui en a besoin. Pour l'interdire tout à fait, on le masque :

```
gary@serveur:~$ sudo systemctl mask nginx
Created symlink /etc/systemd/system/nginx.service → /dev/null.
gary@serveur:~$ sudo systemctl start nginx
Failed to start nginx.service: Unit nginx.service is masked.
gary@serveur:~$ sudo systemctl unmask nginx
Removed "/etc/systemd/system/nginx.service".
```

Le procédé se lit dans la sortie : un fichier vide, placé dans `/etc/systemd/system/`, cache l'unité du paquet.

12.8 Anatomie d'une unité de service

`systemctl cat` affiche le fichier d'une unité. Celui de cron est court :

```
gary@serveur:~$ systemctl cat cron
# /usr/lib/systemd/system/cron.service
[Unit]
Description=Regular background program processing daemon
Documentation=man:cron(8)
After=remote-fs.target nss-user-lookup.target
```

```
[Service]
EnvironmentFile=-/etc/default/cron
ExecStart=/usr/sbin/cron -f -P $EXTRA_OPTS
IgnoreSIGPIPE=false
KillMode=process
Restart=on-failure
SyslogFacility=cron

[Install]
WantedBy=multi-user.target
```

Le fichier compte trois sections.

`[Unit]` présente l'unité et la situe par rapport aux autres.

Directive	Rôle
`Description=`	Le libellé affiché par `status` et dans le journal
`After=`, `Before=`	L'ordre : démarrer après, ou avant, les unités citées
`Wants=`	Demander aussi le démarrage des unités citées. Si elles échouent, tant pis
`Requires=`	Même chose, en plus strict : si elles échouent, cette unité n'est pas démarrée

`After=` ne déclenche rien. Il dit seulement : si les deux doivent démarrer, moi en second. Pour qu'une unité en entraîne une autre, il faut `Wants=` ou `Requires=`. Les deux s'emploient souvent ensemble. L'unité de nginx contient ainsi `After=network-online.target` et `Wants=network-online.target` : elle réclame le réseau et attend qu'il soit prêt.

`[Service]` dit comment lancer le programme et quoi faire s'il s'arrête.

Directive	Rôle
`ExecStart=`	La commande à lancer, avec son chemin complet
`ExecReload=`	La commande exécutée par `systemctl reload`
`User=`, `Group=`	Le compte sous lequel tourne le service. Sans cette ligne, c'est `root`
`WorkingDirectory=`	Son répertoire courant
`Environment=`	Une variable d'environnement, sous la forme `NOM=valeur`
`EnvironmentFile=`	Un fichier de variables. Précédé d'un tiret, il est facultatif
`Type=`	La façon dont le programme démarre (voir ci-dessous)

Directive	Rôle
`Restart=`	Le relancer s'il s'arrête : `no`, `on-failure` ou `always`
`RestartSec=`	Le délai avant de le relancer. Par défaut, un dixième de seconde

`Restart=on-failure` convient à la plupart des services. Le programme est relancé s'il sort avec un code d'erreur ou s'il est tué par un signal. Il ne l'est pas après un arrêt normal.

`Type=` répond à une question : à quel moment le service est-il vraiment démarré ?

Type	Comportement du programme
`simple`	Il reste au premier plan. C'est le type par défaut, et celui des programmes que vous écrirez
`forking`	Il se dédouble et rend la main, à l'ancienne. C'est le cas de nginx
`notify`	Il prévient lui-même systemd quand il est prêt. C'est le cas de sshd et de rsyslog
`oneshot`	Il fait son travail et se termine : un script d'initialisation, une sauvegarde

`[Install]` ne sert qu'à `enable`. `WantedBy=multi-user.target` signifie : quand ce service est activé, la cible `multi-user.target` l'entraîne au démarrage.

> **ATTENTION · Pas de commentaire en fin de ligne**
>
> Dans un fichier d'unité, un commentaire occupe une ligne entière, commencée par `#`. Placé après une valeur, il en fait partie. La ligne `MemoryMax=300M # plafond` est rejetée, et il faut lire le journal pour le savoir :
>
> ```
> override.conf:2: Invalid memory limit '300M    # plafond', ignoring: Invalid
> argument
> ```
>
> Le service démarre quand même, sans la limite que vous pensiez avoir posée.

12.9 Retoucher une unité sans la modifier

Vous voulez que nginx soit relancé automatiquement s'il tombe. Son unité, livrée par le paquet, ne le prévoit pas. La modifier dans `/usr/lib` serait une erreur : la prochaine mise à jour remettrait le fichier d'origine.

systemd prévoit pour cela les fichiers de *surcharge*. `sudo systemctl edit nginx` ouvre un éditeur sur un fichier vide. Vous n'y écrivez que ce qui change :

```
[Service]
Restart=on-failure
RestartSec=5
```

À l'enregistrement, le fichier est créé sous le nom `/etc/systemd/system/nginx.service.d/override.conf` et systemd relit sa configuration. `systemctl cat nginx` montre l'unité d'origine suivie de la surcharge, et `status` la signale :

```
gary@serveur:~$ systemctl status nginx | head -5
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
    Drop-In: /etc/systemd/system/nginx.service.d
             └─override.conf
     Active: active (running) since Sun 2026-10-04 20:37:28 UTC; 7s ago
```

Vérifions en tuant le processus principal :

```
gary@serveur:~$ sudo kill -9 438
gary@serveur:~$ journalctl -u nginx -n 4 -o cat
nginx.service: Failed with result 'signal'.
nginx.service: Scheduled restart job, restart counter is at 1.
Starting nginx.service - A high performance web server and a reverse proxy server...
Started nginx.service - A high performance web server and a reverse proxy server.
```

Cinq secondes plus tard, nginx tournait de nouveau. `sudo systemctl revert nginx` supprime toutes les surcharges et rend à l'unité son état d'origine.

Quand vous modifiez à la main un fichier de `/etc/systemd/system/`, systemd ne s'en aperçoit pas tout seul. Il vous le rappelle à la commande suivante :

```
Warning: The unit file, source configuration file or drop-ins of veilleur.service
changed on disk. Run 'systemctl daemon-reload' to reload units.
```

Après chaque modification : `sudo systemctl daemon-reload`, puis `sudo systemctl restart` du service concerné.

12.10 Mesurer et plafonner les ressources

systemd range les processus de chaque service dans un **groupe de contrôle**, ou *cgroup*. C'est un mécanisme du noyau qui rassemble des processus, mesure ce qu'ils

consomment ensemble et peut le limiter. Grâce à lui, `status` connaît la mémoire d'un service entier, et `stop` atteint tous ses processus, y compris ceux qu'il a lancés entre-temps.

`systemd-cgls` affiche l'arbre des groupes :

```
gary@serveur:~$ systemd-cgls
CGroup /:
-.slice
├─user.slice
│ └─user-1000.slice
│   ├─user@1000.service …
│   │ └─init.scope
│   │   ├─358 /usr/lib/systemd/systemd --user
│   │   └─359 (sd-pam)
│   └─session-309.scope
│     ├─354 sshd: gary [priv]
│     ├─373 sshd: gary@pts/0
│     ├─374 -bash
│     └─720 systemd-cgls
├─init.scope
│ └─1 /sbin/init
└─system.slice
  ├─cron.service
  │ └─66 /usr/sbin/cron -f -P
  ├─nginx.service
  │ ├─690 nginx: master process /usr/sbin/nginx -g daemon on; master_process …
  │ ├─691 nginx: worker process
  │ └─692 nginx: worker process
[…]
```

Deux grandes branches se partagent la machine. `system.slice` contient les services. `user.slice` contient les sessions des utilisateurs, avec une sous-branche par personne et un groupe par connexion. Votre shell et tout ce que vous lancez vivent dans `session-309.scope`.

Chaque groupe peut recevoir des limites :

Directive	Effet
`MemoryMax=500M`	Plafond de mémoire. S'il est atteint, le noyau tue un processus du groupe
`MemoryHigh=400M`	Seuil d'alerte : au-delà, le groupe est ralenti, pas tué
`CPUQuota=50%`	Au plus la moitié d'un processeur. `200%` vaut deux processeurs entiers
`CPUWeight=50`	Une part réduite quand le processeur est disputé. La valeur normale est 100

Directive	Effet
`TasksMax=100`	Nombre maximal de processus et de fils d'exécution

Ces directives s'écrivent dans la section `[Service]` d'une unité ou d'une surcharge. On peut aussi les poser à chaud :

```
gary@serveur:~$ sudo systemctl set-property nginx.service MemoryMax=500M
gary@serveur:~$ systemctl status nginx | grep Memory:
     Memory: 2.2M (max: 500.0M available: 497.7M peak: 2.6M)
```

La limite s'applique immédiatement et survivra au redémarrage : `set-property` l'a enregistrée dans une surcharge. Avec `--runtime`, elle ne vaudrait que jusqu'au prochain redémarrage.

`systemd-cgtop` classe les groupes par consommation, à la manière de `top`. Il répond à la question « quel service occupe la machine ? » quand `top` ne montre qu'une foule de processus.

> **NOTE · cgroups, version 2**
>
> Les groupes de contrôle sont apparus dans le noyau 2.6.24. Leur seconde version, plus cohérente, est officielle depuis le noyau 4.5, et Ubuntu l'utilise par défaut depuis la 21.10. Les conteneurs du chapitre 28 reposent sur le même mécanisme.

12.11 Cloisonner un service

Un service n'a besoin que de très peu de choses : lire son programme, écrire dans un dossier, parfois écouter sur un port. Tout le reste de la machine lui est inutile. Si une faille permet un jour d'en prendre le contrôle, autant que l'attaquant se retrouve dans une pièce vide.

systemd peut restreindre ce que voit et ce que peut faire un service, sans toucher au programme. Il suffit d'ajouter des lignes dans `[Service]` :

Directive	Effet
`User=`, `Group=`	Tourner sous un compte sans privilèges
`NoNewPrivileges=yes`	Interdire d'acquérir des droits, par exemple en lançant `sudo`
`ProtectSystem=strict`	Tout le système de fichiers passe en lecture seule

Directive	Effet
`StateDirectory=nom`	Créer `/var/lib/nom`, qui appartient au service et où il peut écrire
`ReadWritePaths=`	Autoriser l'écriture dans d'autres dossiers
`ProtectHome=yes`	Rendre `/home` et `/root` invisibles
`PrivateTmp=yes`	Donner au service un `/tmp` rien qu'à lui
`PrivateDevices=yes`	Lui cacher les périphériques
`RestrictAddressFamilies=`	Limiter les types de connexions réseau
`SystemCallFilter=@system-service`	Limiter les appels au noyau à ceux d'un service ordinaire

`ProtectSystem=` a deux réglages plus souples. Avec `yes`, seuls `/usr` et `/boot` sont en lecture seule. Avec `full`, `/etc` l'est aussi.

`systemd-analyze security` note l'exposition de chaque service, de 0 pour un service entièrement cloisonné à 10 pour un service libre de tout faire :

```
gary@serveur:~$ systemd-analyze security
UNIT                                EXPOSURE PREDICATE
cron.service                             9.6 UNSAFE
nginx.service                            9.6 UNSAFE
rsyslog.service                          6.3 MEDIUM
ssh.service                              9.6 UNSAFE
[…]
```

Sur votre écran, une dernière colonne ajoute à chaque ligne une frimousse plus ou moins inquiète.

Ne lisez pas `UNSAFE` comme « vulnérable ». La note mesure ce qu'un attaquant pourrait faire une fois dans la place, pas la probabilité qu'il y entre. Certains services ont réellement besoin d'un large accès : cron doit pouvoir lancer n'importe quelle tâche pour n'importe quel utilisateur. Pour vos propres services, en revanche, rien ne justifie une note élevée. Suivie d'un nom d'unité, la commande détaille chaque directive et ce qu'elle rapporterait.

La méthode est toujours la même : ajouter les protections une par une, redémarrer, vérifier que le service fonctionne, lire son journal s'il échoue. L'atelier la met en pratique.

12.12 Lancer une commande comme un service

`systemd-run` crée une unité temporaire autour d'une commande, sans écrire de fichier. C'est le moyen le plus simple d'essayer une limite ou une protection :

```
gary@serveur:~$ sudo systemd-run --wait -p MemoryMax=50M -p MemorySwapMax=0 \
    python3 -c 'x = b"x" * (200 * 1024 * 1024)'
Running as unit: run-u35.service; invocation ID: ec97397d53724865847aebe5492ba03d
Finished with result: oom-kill
Main processes terminated with: code=killed/status=KILL
Service runtime: 26ms
CPU time consumed: 25ms
Memory peak: 50.0M
Memory swap peak: 0B
```

Le programme réclamait 200 Mo, le groupe n'avait droit qu'à 50. Le noyau l'a tué, et systemd le dit : `oom-kill`, pour *out of memory*.

La même commande sait aussi différer une exécution. Elle crée alors une planification temporaire :

```
gary@serveur:~$ sudo systemd-run --on-active=30min --unit=rappel touch /tmp/fait
Running timer as unit: rappel.timer
Will run service as unit: rappel.service
```

En résumé

- Un processus a un PID, un parent, un propriétaire et un état. `ps aux` les liste, `pstree` montre qui a lancé qui, `top` et `htop` les suivent en direct.
- La charge moyenne se compare au nombre de processeurs.
- `kill` envoie un signal. `SIGTERM` d'abord, `SIGKILL` seulement si le processus ne réagit pas.
- `pgrep -a` avant `pkill`.
- `nice -n 19` pour les travaux lourds et peu pressés.
- Ctrl+Z puis `bg` envoient en arrière-plan une commande déjà lancée. `tmux` protège des déconnexions.
- `systemctl status` est le premier réflexe quand un service pose problème.
- `start` agit maintenant, `enable` au prochain démarrage. `enable --now` fait les deux.

- On ne modifie pas une unité livrée par un paquet : `systemctl edit` crée une surcharge.
- Après une modification à la main, `systemctl daemon-reload`.
- Un service bien écrit tourne sous son propre compte, ne voit que ce dont il a besoin et a un plafond de mémoire.

Atelier — Un service sur mesure

Objectif : faire d'un script Python un vrai service. Il démarrera avec la machine, sera relancé s'il tombe, tournera sous un compte sans privilèges, dans un espace cloisonné, avec des ressources plafonnées. Comptez quarante-cinq minutes.

Le programme

1. Créez le dossier du programme et saisissez le script :

```
sudo mkdir /opt/veilleur
sudo nano /opt/veilleur/veilleur.py
```

```
#!/usr/bin/env python3
"""veilleur — note la charge de la machine à intervalle régulier."""
import os
import signal
import sys
import time

INTERVALLE = int(os.environ.get("VEILLEUR_INTERVALLE", "10"))
SEUIL = float(os.environ.get("VEILLEUR_SEUIL", "2.0"))
RELEVES = "/var/lib/veilleur/charge.csv"


def arreter(numero, _cadre):
    print(f"signal {numero} reçu, arrêt propre")
    sys.exit(0)


signal.signal(signal.SIGTERM, arreter)
signal.signal(signal.SIGINT, arreter)

print(f"démarrage : PID {os.getpid()}, UID {os.getuid()}, "
      f"un relevé toutes les {INTERVALLE} s")

while True:
    charge = os.getloadavg()[0]
    horodatage = time.strftime("%Y-%m-%d %H:%M:%S")
```

```
    with open(RELEVES, "a", encoding="utf-8") as releves:
        releves.write(f"{horodatage};{charge:.2f}\n")
    if charge > SEUIL:
        print(f"charge élevée : {charge:.2f} (seuil : {SEUIL})")
    time.sleep(INTERVALLE)
```

Toutes les dix secondes, le script note la charge de la machine dans un fichier. Il signale dans le journal les dépassements du seuil. Il intercepte `SIGTERM` pour s'arrêter proprement.

Le fichier appartient à `root`. Le service pourra le lire, pas le modifier : un programme ne doit pas pouvoir réécrire son propre code.

Le compte

2. Créez un compte système réservé au service :

```
gary@serveur:~$ sudo adduser --system --group --no-create-home veilleur
info: Selecting UID from range 100 to 999 ...

info: Selecting GID from range 100 to 999 ...
info: Adding system user `veilleur' (UID 103) ...
info: Adding new group `veilleur' (GID 104) ...
info: Adding new user `veilleur' (UID 103) with group `veilleur' ...
info: Not creating `/nonexistent'.
```

Ce compte n'a ni mot de passe, ni répertoire personnel, ni shell. Personne ne peut s'y connecter.

L'unité

3. Créez l'unité avec `sudo nano /etc/systemd/system/veilleur.service` :

```
[Unit]
Description=Veilleur (relevé de la charge)
StartLimitIntervalSec=60
StartLimitBurst=3

[Service]
User=veilleur
Group=veilleur
ExecStart=/usr/bin/python3 /opt/veilleur/veilleur.py
Environment=PYTHONUNBUFFERED=1
StateDirectory=veilleur
Restart=on-failure
RestartSec=5
```

```
[Install]
WantedBy=multi-user.target
```

Trois lignes méritent une explication.

`StateDirectory=veilleur` crée `/var/lib/veilleur` et le donne au compte du service. C'est là que le script écrit ses relevés.

`Environment=PYTHONUNBUFFERED=1` demande à Python d'écrire ses messages sans attendre. Sans cette variable, il les garde en réserve et le journal ne les reçoit que bien plus tard : lors de l'essai, le message de démarrage n'y est apparu qu'à l'arrêt du service.

`StartLimitIntervalSec=` et `StartLimitBurst=` fixent une limite : pas plus de trois démarrages en soixante secondes. Un service qui plante dès son lancement ne tournera pas en boucle. Ces deux lignes se placent dans `[Unit]`. Dans `[Service]`, systemd les ignorerait avec un simple avertissement.

Démarrer et observer

4. Faites connaître l'unité à systemd, puis activez et démarrez le service :

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl enable --now veilleur
Created symlink /etc/systemd/system/multi-user.target.wants/veilleur.service
→ /etc/systemd/system/veilleur.service.
```

5. Vérifiez son état, lisez son journal et regardez ses relevés :

```
gary@serveur:~$ systemctl status veilleur
● veilleur.service - Veilleur (relevé de la charge)
     Loaded: loaded (/etc/systemd/system/veilleur.service; enabled; preset: enabled)
     Active: active (running) since Sun 2026-10-04 20:37:53 UTC; 374ms ago
   Main PID: 911 (python3)
      Tasks: 1 (limit: 2493)
     Memory: 3.7M (peak: 3.9M)
        CPU: 10ms
     CGroup: /system.slice/veilleur.service
             └─911 /usr/bin/python3 /opt/veilleur/veilleur.py
gary@serveur:~$ journalctl -u veilleur -n 1 -o cat
démarrage : PID 911, UID 103, un relevé toutes les 10 s
gary@serveur:~$ cat /var/lib/veilleur/charge.csv
2026-10-04 20:37:53;0.44
```

Le faire tomber

6. Tuez le processus principal sans ménagement, attendez six secondes et lisez le journal :

```
gary@serveur:~$ sudo kill -9 911
gary@serveur:~$ journalctl -u veilleur -n 5 -o cat
veilleur.service: Main process exited, code=killed, status=9/KILL
veilleur.service: Failed with result 'signal'.
veilleur.service: Scheduled restart job, restart counter is at 1.
Started veilleur.service - Veilleur (relevé de la charge).
démarrage : PID 924, UID 103, un relevé toutes les 10 s
```

systemd a constaté la mort du processus, attendu les cinq secondes de `RestartSec=`, puis relancé le service.

7. Arrêtez-le maintenant dans les règles et comparez :

```
gary@serveur:~$ sudo systemctl stop veilleur
gary@serveur:~$ journalctl -u veilleur -n 4 -o cat
Stopping veilleur.service - Veilleur (relevé de la charge)...
signal 15 reçu, arrêt propre
veilleur.service: Deactivated successfully.
Stopped veilleur.service - Veilleur (relevé de la charge).
```

Le script a reçu `SIGTERM`, écrit son message et rendu la main. Aucun redémarrage : `on-failure` ne relance pas après un arrêt demandé.

8. Attendez une minute, redémarrez le service, puis tuez-le trois fois de suite en laissant six secondes entre chaque essai. `systemctl kill` évite de chercher le PID :

```
sudo systemctl start veilleur
sudo systemctl kill -s KILL veilleur
```

À la troisième fois, systemd renonce :

```
gary@serveur:~$ journalctl -u veilleur -n 3 -o cat
veilleur.service: Start request repeated too quickly.
veilleur.service: Failed with result 'signal'.
Failed to start veilleur.service - Veilleur (relevé de la charge).
```

La limite de démarrages a joué. Elle compte aussi les démarrages manuels : pendant vos essais, vous la toucherez sans le vouloir. Pour repartir, remettez le compteur à zéro :

```
sudo systemctl reset-failed veilleur
sudo systemctl start veilleur
```

Cloisonner

9. Relevez la note de départ :

```
gary@serveur:~$ systemd-analyze security veilleur | tail -1
→ Overall exposure level for veilleur.service: 9.2 UNSAFE
```

10. Ajoutez ces lignes à la fin de la section `[Service]` de l'unité :

```
# Cloisonnement
NoNewPrivileges=yes
ProtectSystem=strict
ProtectHome=yes
PrivateTmp=yes
PrivateDevices=yes
ProtectKernelTunables=yes
ProtectKernelModules=yes
ProtectKernelLogs=yes
ProtectControlGroups=yes
ProtectClock=yes
ProtectHostname=yes
RestrictAddressFamilies=AF_UNIX
RestrictNamespaces=yes
RestrictRealtime=yes
RestrictSUIDSGID=yes
LockPersonality=yes
MemoryDenyWriteExecute=yes
SystemCallFilter=@system-service
SystemCallArchitectures=native
CapabilityBoundingSet=

# Ressources
MemoryMax=100M
CPUQuota=10%
TasksMax=10
```

Le script n'utilise pas le réseau : `RestrictAddressFamilies=AF_UNIX` ne lui laisse que les échanges internes à la machine. `CapabilityBoundingSet=`, sans valeur, lui retire tous les privilèges particuliers de `root`. `man systemd.exec` décrit chacune des autres lignes.

11. Rechargez, redémarrez et mesurez de nouveau :

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl restart veilleur
gary@serveur:~$ systemd-analyze security veilleur | tail -1
→ Overall exposure level for veilleur.service: 1.4 OK
gary@serveur:~$ systemctl status veilleur | head -6
● veilleur.service - Veilleur (relevé de la charge)
     Loaded: loaded (/etc/systemd/system/veilleur.service; enabled; preset: enabled)
     Active: active (running) since Sun 2026-10-04 20:38:30 UTC; 735ms ago
   Main PID: 1016 (python3)
      Tasks: 1 (limit: 10)
     Memory: 4.0M (max: 100.0M available: 96.0M peak: 4.1M)
```

La note est passée de 9,2 à 1,4, et les plafonds apparaissent dans `status`.

12. Constatez le cloisonnement. Le dossier `/proc/PID/root` montre la machine telle que le processus la voit :

```
gary@serveur:~$ PID=$(systemctl show -p MainPID --value veilleur)
gary@serveur:~$ sudo ls -A /proc/$PID/root/home /proc/$PID/root/tmp
/proc/1016/root/home:

/proc/1016/root/tmp:
gary@serveur:~$ sudo grep -E 'CapEff|NoNewPrivs' /proc/$PID/status
CapEff: 0000000000000000
NoNewPrivs:     1
```

Pour le service, `/home` est vide et `/tmp` aussi, alors que les vôtres ne le sont pas. Sa liste de privilèges (`CapEff`) est à zéro.

Vérifier

- `systemctl is-enabled veilleur` répond `enabled`, `is-active` répond `active`.
- Après `sudo reboot`, le service tourne sans que vous ayez rien fait.
- `/var/lib/veilleur/charge.csv` gagne une ligne toutes les dix secondes.

Gardez ce service : le chapitre 13 se servira de son journal.

Exercices

Exercice 1 — Observer

1. Combien de processus tournent sur votre machine ? Combien vous appartiennent ?
2. Quels sont les cinq processus qui occupent le plus de mémoire vive ?

3. Quel est le PID de `systemd-journald` ? Sous quel compte tourne-t-il ?
4. Affichez la lignée complète de votre shell, de systemd jusqu'à lui.
5. Dans quel état se trouvent la plupart des processus ? Pourquoi ?

Exercice 2 — Signaux et tâches

1. Lancez `sleep 300 &`. Suspendez-le avec `kill -STOP`. Que montrent `jobs` et `ps -o pid,stat,cmd` ?
2. Faites-le repartir, puis arrêtez-le proprement.
3. Lancez `sleep 300` au premier plan. Envoyez-le en arrière-plan sans l'arrêter, ramenez-le au premier plan, puis interrompez-le.
4. Lancez `sleep 300`, tuez-le depuis un second terminal avec `kill`, puis affichez `$?` dans le premier. Recommencez avec `kill -9`. Expliquez les deux valeurs.

Exercice 3 — Priorités

1. Lancez deux fois `taskset -c 0 yes > /dev/null &`. `taskset -c 0` oblige les deux commandes à se partager le premier processeur.
2. Observez leur part de processeur dans `top`.
3. Passez l'une des deux à la valeur de politesse 19. Que devient le partage ?
4. Arrêtez-les avec une seule commande.

Exercice 4 — Un service à vous

Écrivez une unité `bonjour.service` qui :

1. exécute `/bin/bash -c 'while true; do echo Bonjour; sleep 30; done'` ;
2. tourne sous un compte créé à la volée, grâce à `DynamicUser=yes` ;
3. est relancée dans tous les cas, après dix secondes ;
4. ne dépasse ni 50 Mo de mémoire ni 5 % d'un processeur ;
5. dispose d'un `/tmp` privé et ne peut pas acquérir de privilèges ;
6. démarre avec la machine.

Contrôlez avec `systemctl status`, `journalctl -u bonjour` et `systemd-analyze security bonjour`. Sous quel compte tourne le processus ? Supprimez ensuite le service proprement.

Exercice 5 — Diagnostic

Que faites-vous dans chacune de ces situations ?

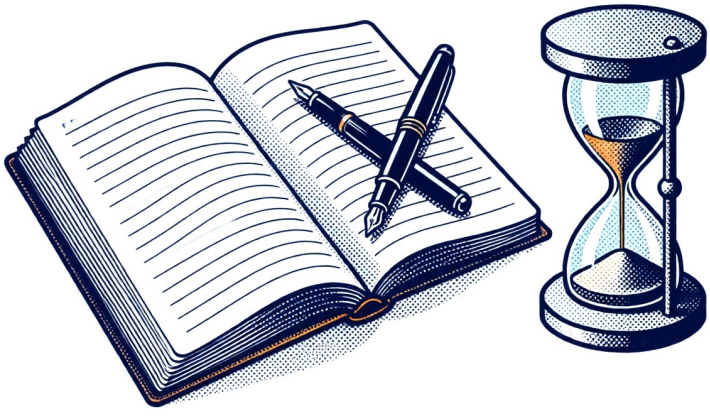
1. `systemctl status nginx` affiche `failed`.
2. Un service occupe trop de mémoire, et vous ne voulez pas modifier son unité.
3. Vous avez corrigé `/etc/systemd/system/veilleur.service`, mais le service se comporte comme avant.
4. `systemctl status` affiche `activating (auto-restart)` chaque fois que vous regardez.
5. Un service désactivé se retrouve en marche après chaque redémarrage.

Pour aller plus loin

Sur la machine. `man systemd.service` décrit la section `[Service]`, `man systemd.exec` les directives de cloisonnement, `man systemd.resource-control` les limites de ressources. `man systemd.directives` est l'index général : il indique dans quelle page chaque directive est documentée. `man 7 signal` donne la liste complète des signaux.

Outils. `strace -p PID` affiche en direct les appels qu'un processus adresse au noyau, et `lsof -p PID` les fichiers qu'il tient ouverts. Ce sont les deux instruments de base quand un programme se comporte de façon incompréhensible. `loginctl` liste les sessions ouvertes sur la machine.

Lectures. Le billet *Rethinking PID 1* de Lennart Poettering (`0pointer.de/blog/projects/systemd.html`) explique les choix de systemd mieux que n'importe quel résumé. *The Linux Programming Interface*, de Michael Kerrisk (No Starch Press), est la référence sur les processus et les signaux vus par le programmeur.



CHAPITRE 13

Journaux et planification

Les journaux racontent ce qui s'est passé. Les planificateurs décident de ce qui se passera cette nuit. Avec les deux, une machine s'entretient seule et s'explique quand elle a un problème.

DANS CE CHAPITRE

- Savoir où Ubuntu écrit ses journaux, et lequel consulter.
- Interroger le journal avec `journalctl` : par service, par date, par gravité.
- Écrire dans le journal depuis un script.
- Limiter la place que prennent les journaux.
- Renouveler les fichiers de journaux avec logrotate.
- Planifier une tâche avec cron, en évitant ses pièges.
- Planifier une tâche avec une minuterie systemd.
- Trouver pourquoi une tâche planifiée ne s'est pas exécutée.

> **AVANT DE COMMENCER**
>
> Les chapitres 10 et 12. Les exemples continuent sur le serveur du chapitre 12, avec nginx et le service `veilleur` construit dans son atelier. L'atelier de ce chapitre planifie le script de sauvegarde écrit au chapitre 10.

13.1 Où Ubuntu écrit ses journaux

Un *journal* (en anglais *log*) est la trace que laisse un programme de ce qu'il fait : un démarrage, une requête reçue, une erreur. Tant que tout va bien, personne ne le lit. Le jour où un service refuse de démarrer, c'est là qu'on trouve la raison.

Sur Ubuntu 24.04, les messages suivent trois chemins.

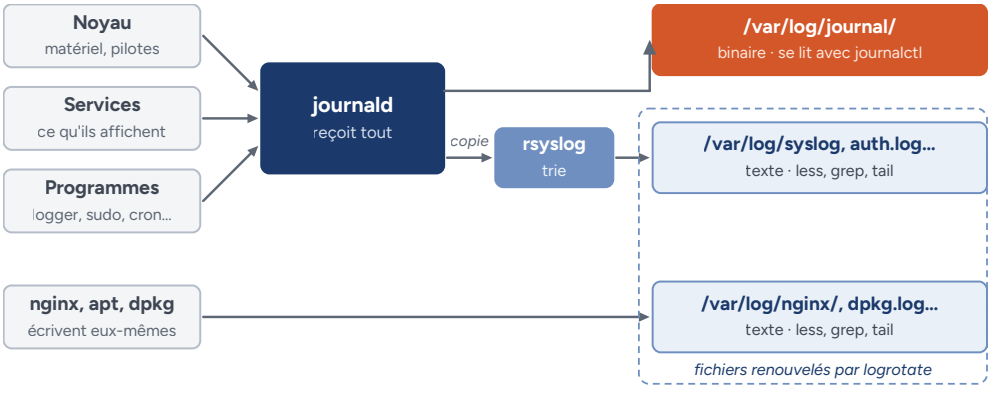


Figure 13.1 — Le trajet des messages, de leur source jusqu'aux fichiers.

journald fait partie de systemd. Il reçoit tout : les messages du noyau, ce qu'affichent les services, ce que les programmes envoient par l'interface traditionnelle d'Unix, *syslog*. Il range l'ensemble dans une base indexée, sous `/var/log/journal/`, qui s'interroge avec `journalctl`.

rsyslog reçoit de journald une copie des messages et les trie dans des fichiers texte. Ce sont les journaux classiques d'Unix, que l'on lit avec `less`, `grep` ou `tail`. Il est installé d'origine, sur Server comme sur Desktop.

Enfin, certains programmes tiennent eux-mêmes leurs fichiers : nginx, `apt`, `dpkg`.

Tout cela se trouve dans `/var/log` :

Fichier	Contenu
`syslog`	Presque tout, sauf l'authentification
`auth.log`	Les connexions, les appels à `sudo`, SSH
`kern.log`	Les messages du noyau
`dpkg.log`, `apt/history.log`	Les paquets installés, mis à jour, retirés (chapitre 7)
`nginx/access.log`, `nginx/error.log`	Les requêtes reçues par nginx, et ses erreurs
`journal/`	La base de journald
`wtmp`, `btmp`	Les connexions réussies et échouées. Ils se lisent avec `last` et `sudo lastb`

Une ligne de `syslog` donne la date, la machine, le programme avec son PID, puis le message. En voici deux :

```
2026-10-04T20:59:14.047497+00:00 serveur crontab[1952]: (gary) REPLACE (gary)
2026-10-04T21:00:01.113660+00:00 serveur CRON[1960]: (gary) CMD (env > /tmp/env-cron.txt)
```

Gary a modifié sa table de tâches planifiées, puis le planificateur a lancé une de ses commandes.

`syslog` et `auth.log` appartiennent au groupe `adm`. Le premier compte créé à l'installation en est membre : il les lit sans `sudo`.

Les deux systèmes contiennent à peu près la même chose. Alors lequel consulter ? Le journal de systemd, dans presque tous les cas : il sait filtrer. Les fichiers texte restent utiles pour les outils du chapitre 9, pour les programmes de surveillance qui ne connaissent qu'eux, et sur une machine en panne dont on ne peut plus lire que le disque.

13.2 Lire le journal avec journalctl

Commande	Ce qu'elle affiche
`journalctl`	Tout le journal, du plus ancien au plus récent
`journalctl -e`	La fin du journal
`journalctl -n 20`	Les vingt dernières lignes
`journalctl -f`	Les nouveaux messages, au fur et à mesure

Commande	Ce qu'elle affiche
`journalctl -r`	Du plus récent au plus ancien
`journalctl -b`	Les messages depuis le dernier démarrage
`journalctl -b -1`	Ceux du démarrage précédent
`journalctl -k`	Les messages du noyau

La sortie s'ouvre dans `less` : `/` pour chercher, `G` pour aller à la fin, `q` pour quitter. Les lignes trop longues sont coupées au bord de l'écran, et la flèche droite montre la suite.

Les membres du groupe `adm` voient tout le journal. Les autres utilisateurs ne voient que leurs propres messages, et `journalctl` les en avertit.

Par service

L'option `-u`, comme *unit*, limite l'affichage à une unité systemd. C'est de loin la plus utile.

```
gary@serveur:~$ journalctl -u veilleur -n 3
Oct 04 20:58:39 serveur systemd[1]: Stopped veilleur.service - Veilleur (relevé de…
Oct 04 20:58:39 serveur systemd[1]: Started veilleur.service - Veilleur (relevé de…
Oct 04 20:58:40 serveur python3[1805]: démarrage : PID 1805, UID 103, un relevé to…
```

On y lit à la fois ce que systemd dit du service et ce que le service écrit lui-même. Les options se combinent : `journalctl -u nginx -u ssh -b` affiche les messages de deux services depuis le démarrage.

Par date

`--since` et `--until` bornent la période. Ils acceptent une date, une heure, ou une expression en anglais :

```
journalctl --since "1 hour ago"
journalctl --since yesterday --until today
journalctl -u nginx --since "2026-10-04 20:00" --until "2026-10-04 21:00"
```

Par gravité

Chaque message porte un niveau de gravité, de 0 à 7 :

Niveau	Nom	Signification
0	`emerg`	Le système est inutilisable
1	`alert`	Il faut intervenir tout de suite
2	`crit`	Situation critique
3	`err`	Erreur
4	`warning`	Avertissement
5	`notice`	Événement normal, mais à noter
6	`info`	Information courante
7	`debug`	Détails pour la mise au point

`-p err` retient les messages de niveau `err` et plus graves :

```
gary@serveur:~$ journalctl -b -p err -n 3
Oct 04 20:38:24 serveur systemd[1]: Failed to start veilleur.service - Veilleur (r…
Oct 04 20:38:27 serveur systemd[1]: Failed to start veilleur.service - Veilleur (r…
Oct 04 20:44:58 serveur systemd[1]: Failed to start veilleur.service - Veilleur (r…
```

Ce sont les traces de l'atelier du chapitre 12, quand nous avons fait tomber le service jusqu'à ce que systemd renonce à le relancer. Sur une machine que vous découvrez, ou après un redémarrage, `journalctl -b -p err` est la première commande à lancer.

Par programme, par utilisateur, par mot

```
journalctl -t sudo              # les messages d'un programme, d'après son nom
journalctl _UID=1000            # ceux des processus d'un utilisateur
journalctl -u ssh -g Accepted   # ceux qui contiennent un motif
```

L'option `-g` attend une expression régulière. Elle ignore la casse tant que le motif est écrit en minuscules.

```
gary@serveur:~$ journalctl -u ssh -g Accepted -n 1 -o cat
Accepted publickey for gary from 172.17.0.1 port 52682 ssh2: ED25519 SHA256:KPqw…
```

Ce que contient vraiment une entrée

Le journal n'enregistre pas des lignes de texte, mais des fiches. Le format `verbose` les montre en entier :

```
gary@serveur:~$ journalctl -u veilleur -n 1 -o verbose
Sun 2026-10-04 20:58:40.183179 UTC […]
    PRIORITY=6
    SYSLOG_IDENTIFIER=python3
    _UID=103
    _COMM=python3
    _EXE=/usr/bin/python3.12
    _CMDLINE=/usr/bin/python3 /opt/veilleur/veilleur.py
    _SYSTEMD_UNIT=veilleur.service
    MESSAGE=démarrage : PID 1805, UID 103, un relevé toutes les 10 s
    _PID=1805
[…]
```

Les champs dont le nom commence par un tiret bas sont ajoutés par journald lui-même. Un programme ne peut pas les falsifier. Chacun peut servir de filtre : `journalctl _UID=103` affiche tout ce qu'ont écrit les processus du compte `veilleur`.

L'option `-o` choisit la présentation :

Format	Usage
`-o cat`	Le message seul, sans date ni programme
`-o short-iso`	La date au format ISO, avec le fuseau horaire
`-o verbose`	Tous les champs
`-o json`	Une fiche JSON par ligne, pour `jq` ou un script

> **À RETENIR · Le réflexe**
>
> Un service ne démarre pas? `systemctl status service`, puis `journalctl -u service -b -e`. La cause est presque toujours écrite dans les dernières lignes. S'il démarre puis retombe, ouvrez un second terminal, lancez-y `journalctl -u service -f` et redémarrez le service : vous le regardez échouer en direct.

13.3 Écrire dans le journal

Vos scripts aussi peuvent écrire dans le journal. La commande `logger` s'en charge. `-t` fixe le nom du programme, `-p` la gravité :

```
gary@serveur:~$ logger -t essai "Bonjour le journal"
gary@serveur:~$ logger -t essai -p user.err "Quelque chose a mal tourné"
gary@serveur:~$ journalctl -t essai
```

```
Oct 04 20:58:45 serveur essai[1827]: Bonjour le journal
Oct 04 20:58:45 serveur essai[1828]: Quelque chose a mal tourné
gary@serveur:~$ journalctl -t essai -p err -o cat
Quelque chose a mal tourné
```

Un script lancé la nuit par cron devrait finir par une ligne `logger`. Le matin, une seule commande dit s'il a tourné et comment il s'est terminé.

Pour un service, c'est encore plus simple : tout ce qu'il affiche part dans le journal, au niveau `info`. Pour qu'un message soit classé autrement, il suffit de le faire précéder du numéro de niveau entre chevrons :

```
gary@serveur:~$ sudo systemd-run --unit=niveaux bash -c \
    'echo normal; echo "<4>attention"; echo "<3>erreur"'
Running as unit: niveaux.service; invocation ID: 9322b2ae501840a4aac9a27324254bd2
gary@serveur:~$ journalctl -u niveaux -p warning -o cat
attention
erreur
```

Dans le script `veilleur.py` du chapitre 12, écrire `print(f"<4>charge élevée…")` suffit donc à transformer l'alerte en véritable avertissement, que `journalctl -u veilleur -p warning` saura retrouver.

13.4 La place occupée par le journal

Sur Ubuntu, le journal survit aux redémarrages : le dossier `/var/log/journal/` existe d'origine, et journald y écrit. Sans ce dossier, il garderait tout en mémoire et le perdrait à chaque arrêt.

```
gary@serveur:~$ journalctl --disk-usage
Archived and active journals take up 16.0M in the file system.
```

journald se limite lui-même à 10 % de la partition, sans dépasser 4 Go. Quand la limite est atteinte, il efface les entrées les plus anciennes. Pour choisir vos propres limites, créez un fichier dans `/etc/systemd/journald.conf.d/` :

```
gary@serveur:~$ cat /etc/systemd/journald.conf.d/taille.conf
[Journal]
SystemMaxUse=500M
MaxRetentionSec=1month
gary@serveur:~$ sudo systemctl restart systemd-journald
gary@serveur:~$ journalctl -u systemd-journald -n 1 -o cat
System Journal (/var/log/journal/77719ccb…) is 16.0M, max 500.0M, 483.9M free.
```

`SystemMaxUse=` plafonne la taille, `MaxRetentionSec=` l'ancienneté. Le fichier `/etc/systemd/journald.conf` liste tous les réglages possibles, en commentaire, avec leur valeur par défaut.

Pour faire de la place sans attendre :

```
sudo journalctl --vacuum-size=200M     # ne garder que 200 Mo
sudo journalctl --vacuum-time=2weeks   # ne garder que deux semaines
```

13.5 Renouveler les fichiers avec logrotate

Les fichiers texte, eux, grossissent sans limite. Un `access.log` de serveur web fréquenté prend plusieurs gigaoctets en quelques semaines, et un disque plein arrête tout.

logrotate s'en occupe. À intervalles réguliers, il renomme le fichier courant, en ouvre un nouveau, compresse les anciens et supprime ceux qui dépassent l'ancienneté voulue.



Figure 13.2 — La rotation des journaux de nginx.

Chaque paquet dépose sa règle dans `/etc/logrotate.d/`. Voici celle de nginx, sans ses lignes secondaires :

```
/var/log/nginx/*.log {
    daily
    missingok
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 www-data adm
    sharedscripts
    postrotate
        invoke-rc.d nginx rotate >/dev/null 2>&1
    endscript
}
```

Directive	Effet
`daily`, `weekly`, `monthly`	Le rythme de la rotation
`rotate 14`	Garder quatorze anciens fichiers
`compress`	Compresser les anciens avec gzip
`delaycompress`	Sauf le plus récent, pour qu'il reste lisible directement
`missingok`	Ne pas se plaindre si le fichier n'existe pas
`notifempty`	Ne rien faire si le fichier est vide
`create 0640 www-data adm`	Créer le nouveau fichier avec ces droits et ce propriétaire
`size 100M`	Tourner dès que le fichier dépasse cette taille
`postrotate … endscript`	Les commandes à lancer après la rotation
`sharedscripts`	Ne lancer ces commandes qu'une fois, même si plusieurs fichiers ont tourné

`-f` force une rotation immédiate. Après deux rotations forcées :

```
gary@serveur:~$ sudo logrotate -f /etc/logrotate.d/nginx
gary@serveur:~$ ls -l /var/log/nginx
total 12
-rw-r----- 1 www-data adm  79 Oct  4 21:01 access.log
-rw-r----- 1 www-data adm  79 Oct  4 21:01 access.log.1
-rw-r----- 1 www-data adm 117 Oct  4 21:01 access.log.2.gz
-rw-r----- 1 www-data adm   0 Oct  4 20:28 error.log
```

`error.log` était vide : `notifempty` l'a laissé tranquille.

Le bloc `postrotate` est indispensable. Un programme qui a ouvert un fichier continue d'y écrire même si on le renomme (c'est l'inode du chapitre 4 qui compte, pas le nom). Sans prévenir nginx, il remplirait `access.log.1` pendant que le nouveau `access.log` resterait vide. La commande du `postrotate` lui demande de rouvrir ses fichiers. Pour un programme qui ne sait pas le faire, la directive `copytruncate` copie le journal puis vide l'original sur place. Quelques lignes peuvent se perdre entre les deux opérations.

logrotate n'est pas un service qui tourne en permanence. Sur Ubuntu 24.04, la minuterie `logrotate.timer` le lance une fois par jour. Nous y revenons plus bas.

Écrire sa propre règle

Le service `veilleur` ajoute une ligne à `/var/lib/veilleur/charge.csv` toutes les dix secondes. Rien ne limite ce fichier. Donnons-lui une règle, dans `/etc/logrotate.d/veilleur` :

```
/var/lib/veilleur/charge.csv {
    weekly
    rotate 8
    compress
    delaycompress
    missingok
    notifempty
    su veilleur veilleur
}
```

Le dossier appartient au compte `veilleur`. La directive `su` demande à logrotate d'y travailler sous cette identité plutôt que sous `root`. Aucun `postrotate` n'est nécessaire, car le script rouvre son fichier à chaque relevé.

Avant de faire confiance à une règle, essayez-la à blanc avec `-d`. logrotate lit la configuration, dit ce qu'il ferait et ne touche à rien :

```
gary@serveur:~$ sudo logrotate -d /etc/logrotate.d/veilleur
[…]
rotating pattern: /var/lib/veilleur/charge.csv  weekly (8 rotations)
empty log files are not rotated, old logs are removed
switching euid from 0 to 103 and egid from 0 to 104 (pid 1902)
considering log /var/lib/veilleur/charge.csv
[…]
  log does not need rotating (log has already been rotated)
```

Puis forcez une première rotation :

```
gary@serveur:~$ sudo logrotate -f /etc/logrotate.d/veilleur
gary@serveur:~$ sudo ls -l /var/lib/veilleur
total 8
-rw-r--r-- 1 veilleur veilleur   25 Oct  4 20:59 charge.csv
-rw-r--r-- 1 veilleur veilleur 3350 Oct  4 20:58 charge.csv.1
```

13.6 rsyslog, pour trier et centraliser

rsyslog range les messages d'après deux critères : leur origine (`auth`, `kern`, `mail`…) et leur gravité. Ses règles tiennent en une ligne, dans `/etc/rsyslog.d/50-default.conf` :

```
auth,authpriv.*              /var/log/auth.log
*.*;auth,authpriv.none       -/var/log/syslog
kern.*                       -/var/log/kern.log
```

La première envoie tout ce qui touche à l'authentification dans `auth.log`. La deuxième envoie tout le reste dans `syslog`.

Son grand intérêt est ailleurs : il sait expédier les messages vers une autre machine. Sur un parc de serveurs, on réunit ainsi tous les journaux au même endroit. Ils y restent consultables quand un serveur est en panne, et hors de portée d'un intrus qui voudrait effacer ses traces.

Sur chaque serveur, une ligne suffit, dans un fichier comme `/etc/rsyslog.d/90-distant.conf` :

```
*.* @@journaux.exemple.net:514
```

Les deux arobases demandent une connexion TCP. Une seule, et l'envoi se ferait en UDP, sans garantie d'arrivée. Sur le serveur central, deux lignes ouvrent le port :

```
# Accepte les messages envoyés par d'autres machines, en TCP sur le port 514
module(load="imtcp")
input(type="imtcp" port="514")
```

Après chaque modification, `sudo rsyslogd -N1` vérifie la syntaxe, puis `sudo systemctl restart rsyslog` l'applique. Le chapitre 21 dira comment n'ouvrir ce port qu'aux machines voulues.

13.7 Planifier avec cron

> **REPÈRE HISTORIQUE · Un demi-siècle de service**
>
> cron existe depuis 1975. Son nom vient du grec *chronos*, le temps. La version qu'utilise Ubuntu descend de celle qu'a écrite Paul Vixie en 1987. Sa syntaxe n'a presque pas changé depuis, et on la retrouve sur tous les Unix.

cron lit des tables où chaque ligne décrit une tâche : cinq champs pour le moment, puis la commande.



Se lit : à 2 h 30, du lundi au vendredi, lancer la sauvegarde.

Figure 13.3 — Une ligne de cron.

Chaque champ accepte une valeur, une étoile pour « toutes », une liste (`1,15`), un intervalle (`9-17`) ou un pas (`*/10`).

Ligne	Moment
`0 * * * *`	À chaque heure pile
`*/10 * * * *`	Toutes les dix minutes
`30 7 * * *`	Tous les jours à 7 h 30
`0 18 * * 1,4`	Les lundis et jeudis à 18 h
`0 9-17 * * 1-5`	Chaque heure de 9 h à 17 h, du lundi au vendredi
`45 3 5 * *`	Le 5 de chaque mois à 3 h 45
`@reboot`	À chaque démarrage de la machine
`@daily`	Chaque jour à minuit

> **ATTENTION · Jour du mois et jour de la semaine**
>
> Quand ces deux champs sont remplis tous les deux, la tâche s'exécute dès que l'un ou l'autre correspond. `0 8 13 * 5` ne veut pas dire « les vendredis 13 à 8 h », mais « le 13 de chaque mois, et aussi tous les vendredis ».

Où écrire la ligne

Dans votre table personnelle. `crontab -e` l'ouvre dans un éditeur. La première fois, Ubuntu demande lequel utiliser. À l'enregistrement, cron prend la nouvelle table en compte. Les tâches s'exécutent sous votre identité.

```
crontab -e     # modifier sa table
crontab -l     # l'afficher
crontab -r     # la supprimer en entier, sans demander confirmation
```

Sur un clavier, E et R sont voisins. Gardez une copie : `crontab -l > ~/crontab.txt`.

Dans `/etc/cron.d/`. Les paquets et l'administrateur y déposent des fichiers. Ils suivent la même syntaxe, avec un champ de plus avant la commande : le compte sous lequel la lancer.

Dans les dossiers `/etc/cron.daily`, `cron.weekly` et `cron.monthly`. Il n'y a plus de ligne à écrire : tout script exécutable placé là est lancé chaque jour, chaque semaine ou chaque mois. Les horaires sont fixés dans `/etc/crontab` :

```
25 6 * * * root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.daily; }
```

Les scripts quotidiens partent donc à 6 h 25. Sur Desktop, c'est anacron qui prend le relais : il rattrape au démarrage les tâches manquées pendant que le poste était éteint.

> **ATTENTION · Pas de point dans le nom**
>
> `run-parts` ignore les fichiers dont le nom contient un point. Un script nommé `nettoyage.sh` posé dans `/etc/cron.daily` ne sera jamais lancé, et rien ne le signalera. Appelez-le `nettoyage`, et vérifiez avec `run-parts --test /etc/cron.daily`, qui liste ce qui serait exécuté.

L'environnement de cron

Une commande qui marche dans votre terminal peut échouer sous cron. La raison est toujours la même : cron ne la lance pas dans les mêmes conditions. Une tâche qui écrit le résultat de `env` dans un fichier le montre :

```
gary@serveur:~$ cat /tmp/env-cron.txt
HOME=/home/gary
LOGNAME=gary
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:[…]
LANG=C.UTF-8
SHELL=/bin/sh
PWD=/home/gary
```

Le shell est `sh`, pas Bash : les `[[ … ]]` et les tableaux du chapitre 10 n'y fonctionnent pas. Ni `~/.bashrc` ni `~/.profile` ne sont lus : vos alias, vos variables et

votre dossier `~/bin` n'existent pas. Le répertoire courant est votre répertoire personnel.

La parade tient en une règle : dans la table, n'écrivez qu'un appel à un script, avec son chemin complet. Le script commence par `#!/bin/bash` et définit lui-même ce dont il a besoin.

Deux autres pièges sont propres à cron.

Le caractère `%` a un sens spécial dans une table : cron coupe la commande à cet endroit. Une ligne contenant `$(date +%F)` est exécutée tronquée, comme le montre le journal :

```
(gary) CMD (echo "Archive du $(date +)
```

Il faut écrire `\%`, ou mieux, mettre la commande dans un script.

Ce que la tâche affiche n'apparaît nulle part. cron essaie de l'envoyer par courrier à l'utilisateur. Sans serveur de messagerie sur la machine, il y renonce et le note :

```
(CRON) info (No MTA installed, discarding output)
```

Redirigez donc la sortie vers un fichier, ou passez par `logger`.

> **SUR LE TERRAIN · Deux exécutions en même temps**
>
> cron ne vérifie pas que l'exécution précédente est terminée. Une tâche lancée toutes les cinq minutes et qui en dure parfois sept finit par se marcher sur les pieds. `flock` pose un verrou. Avec `-n`, la commande renonce si le verrou est déjà pris :
>
> ```
> */5 * * * * flock -n /tmp/synchro.verrou /usr/local/bin/synchro
> ```

Vérifier qu'une tâche a tourné

cron note chaque lancement dans le journal :

```
gary@serveur:~$ journalctl -u cron -n 3 -o cat
pam_unix(cron:session): session opened for user gary(uid=1000) by gary(uid=0)
(gary) CMD (/usr/local/bin/sauvegarde "$HOME/Documents")
pam_unix(cron:session): session closed for user gary
```

La ligne `CMD` prouve que cron a lancé la commande. Elle ne dit pas si la commande a réussi. Pour le savoir, il faut que la tâche le dise elle-même, dans un fichier ou par `logger`.

13.8 Une seule fois : at

cron répète. `at` exécute une commande une fois, à l'heure dite. Il s'installe avec `sudo apt install at`.

```
gary@serveur:~$ echo "touch /tmp/fait" | at now + 2 minutes
warning: commands will be executed using /bin/sh
job 2 at Sun Oct  4 21:02:00 2026
gary@serveur:~$ atq
2       Sun Oct  4 21:02:00 2026 a gary
```

L'heure s'écrit de bien des manières : `at 23:00`, `at now + 3 hours`, `at 07:30 tomorrow`. `atq` liste les tâches en attente et `atrm 2` annule la tâche 2. Le cas d'usage typique : redémarrer un service cette nuit à 3 heures, sans veiller.

13.9 Planifier avec systemd : les minuteries

systemd a son propre planificateur. Une tâche y prend la forme de deux unités qui portent le même nom : un service, qui dit quoi faire, et une *minuterie* (en anglais *timer*), qui dit quand.



Figure 13.4 — Une minuterie lance un service, dont le résultat est consigné dans le journal.

Ubuntu s'en sert pour son propre entretien :

```
gary@serveur:~$ systemctl list-timers
NEXT                         LEFT UNIT                    ACTIVATES
Mon 2026-10-05 00:00:00 UTC  3h 3min dpkg-db-backup.timer  dpkg-db-backup.service
Mon 2026-10-05 00:00:00 UTC  3h 3min logrotate.timer       logrotate.service
Mon 2026-10-05 03:35:49 UTC     6h man-db.timer           man-db.service
```

```
Mon 2026-10-05 06:00:26 UTC       9h apt-daily-upgrade.timer apt-daily-upgrade.service
Mon 2026-10-05 09:41:22 UTC      12h apt-daily.timer         apt-daily.service
[…]
```

Deux colonnes ont été retirées ici pour tenir dans la page : `LAST` et `PASSED`, qui donnent la date de la dernière exécution. Cette commande répond d'un coup d'œil à deux questions que cron laisse sans réponse : quand la tâche a-t-elle tourné, et quand tournera-t-elle ?

Une minuterie est un fichier court :

```
gary@serveur:~$ systemctl cat apt-daily.timer
# /usr/lib/systemd/system/apt-daily.timer
[Unit]
Description=Daily apt download activities

[Timer]
OnCalendar=*-*-* 6,18:00
RandomizedDelaySec=12h
Persistent=true

[Install]
WantedBy=timers.target
```

Elle lance le service du même nom, `apt-daily.service`.

Dire quand

`OnCalendar=` fixe un moment du calendrier, sous la forme `jour-de-semaine année-mois-jour heure:minute:seconde`. L'étoile veut dire « tous », et ce qu'on n'écrit pas prend une valeur raisonnable.

OnCalendar=	Moment
`*-*-* 02:30:00`	Tous les jours à 2 h 30
`Mon..Fri 09:00`	Du lundi au vendredi à 9 h
`*:0/15`	Toutes les quinze minutes
`*-*-01 04:00`	Le premier de chaque mois à 4 h
`hourly`, `daily`, `weekly`	Chaque heure, chaque jour à minuit, chaque lundi à minuit

`systemd-analyze calendar` vérifie une expression et calcule les prochaines échéances. Aucun équivalent n'existe pour cron.

```
gary@serveur:~$ systemd-analyze calendar --iterations=3 "*-*-01 04:00"
  Original form: *-*-01 04:00
Normalized form: *-*-01 04:00:00
    Next elapse: Sun 2026-11-01 04:00:00 UTC
       From now: 3 weeks 6 days left
   Iteration #2: Tue 2026-12-01 04:00:00 UTC
       From now: 1 month 26 days left
   Iteration #3: Fri 2027-01-01 04:00:00 UTC
       From now: 2 months 27 days left
```

Une minuterie peut aussi compter à partir d'un événement :

Directive	Déclenchement
`OnBootSec=5min`	Cinq minutes après le démarrage de la machine
`OnUnitActiveSec=30min`	Trente minutes après le dernier lancement du service
`OnUnitInactiveSec=30min`	Trente minutes après la fin de sa dernière exécution

Trois réglages complètent le tableau.

`Persistent=true` rattrape une échéance manquée. Si la machine était éteinte à l'heure prévue, la tâche part au démarrage suivant.

`RandomizedDelaySec=` ajoute un retard tiré au hasard. Dans l'exemple ci-dessus, les téléchargements d'`apt` partent à 6 h ou à 18 h, plus un délai pouvant atteindre douze heures. Ainsi, tous les serveurs Ubuntu du monde ne sollicitent pas les dépôts à la même seconde.

`AccuracySec=` règle la précision. Par défaut, systemd s'autorise jusqu'à une minute de retard pour regrouper les réveils. `AccuracySec=1s` impose l'exactitude.

cron ou minuterie ?

	cron	Minuterie systemd
Mise en place	Une ligne	Deux fichiers
Résultat de la tâche	À organiser soi-même	Dans le journal, avec le code de retour
Machine éteinte à l'heure dite	Tâche perdue (sauf anacron)	Rattrapée avec `Persistent=true`
Exécution précédente pas finie	Une seconde démarre	La minuterie attend

	cron	Minuterie systemd
Limites et cloisonnement	Non	Toutes les directives du chapitre 12
Tâche personnelle sans `sudo`	`crontab -e`	Possible, mais moins simple

Pour une tâche personnelle ou un script de quelques lignes, cron reste imbattable : une ligne, comprise partout. Pour une tâche dont dépend la bonne marche d'un serveur, comme une sauvegarde, préférez une minuterie. Vous saurez toujours si elle a tourné, quand, et pourquoi elle a échoué.

En résumé

- journald reçoit tous les messages. rsyslog en écrit une copie dans les fichiers texte de `/var/log`.
- `journalctl -u service` pour un service, `-b` pour le dernier démarrage, `-p err` pour les erreurs, `-f` pour suivre en direct, `--since` pour une période.
- `logger` écrit dans le journal depuis un script.
- Le journal se limite à 10 % de la partition. `SystemMaxUse=` fixe une autre limite.
- logrotate renouvelle les fichiers texte. Une règle par application dans `/etc/logrotate.d/`, essayée avec `logrotate -d`.
- Une ligne de cron : minute, heure, jour du mois, mois, jour de la semaine, commande.
- cron lance les commandes avec `sh`, sans votre environnement, et jette ce qu'elles affichent. Appelez un script par son chemin complet.
- Une minuterie systemd associe un fichier `.timer` et un fichier `.service`. `systemctl list-timers` montre les échéances.
- `systemd-analyze calendar` vérifie une expression `OnCalendar=`.

Atelier — Planifier la sauvegarde

Objectif : faire tourner chaque nuit le script `sauvegarde.sh` du chapitre 10, d'abord avec cron, puis avec une minuterie systemd, et savoir vérifier qu'il a fait son travail. Comptez quarante minutes.

Préparer

1. Installez le script parmi les commandes de la machine et vérifiez qu'il répond :

```
gary@serveur:~$ sudo install -m 755 sauvegarde.sh /usr/local/bin/sauvegarde
gary@serveur:~$ sauvegarde -h | head -n 1
Usage : sauvegarde [-d DESTINATION] [-k NOMBRE] [-v] [-h] DOSSIER
```

2. Assurez-vous que `~/Documents` contient quelques fichiers.

Avec cron

3. Ouvrez votre table avec `crontab -e`. Pour ne pas attendre la nuit, planifiez d'abord la tâche deux ou trois minutes plus tard. S'il est 14 h 07 :

```
10 14 * * * /usr/local/bin/sauvegarde "$HOME/Documents"
```

4. L'heure passée, vérifiez :

```
gary@serveur:~$ ls ~/sauvegardes
Documents-2026-10-04-210501.tar.gz  sauvegarde.log
gary@serveur:~$ journalctl -u cron -n 3 -o cat
pam_unix(cron:session): session opened for user gary(uid=1000) by gary(uid=0)
(gary) CMD (/usr/local/bin/sauvegarde "$HOME/Documents")
pam_unix(cron:session): session closed for user gary
```

Le journal de cron montre le lancement. C'est le fichier `sauvegarde.log`, tenu par le script, qui dit comment il s'est terminé.

5. Supprimez la ligne d'essai avec `crontab -e`. Pour une sauvegarde chaque nuit à 2 h 30, la ligne définitive serait `30 2 * * *` suivie de la même commande. Ne l'ajoutez pas : nous allons confier ce travail à systemd.

Avec une minuterie

6. Créez le service, dans `/etc/systemd/system/sauvegarde.service` :

```
[Unit]
Description=Sauvegarde des documents de Gary

[Service]
Type=oneshot
User=gary
ExecStart=/usr/local/bin/sauvegarde -v -d /home/gary/sauvegardes /home/gary/Documents
Nice=19
IOSchedulingClass=idle
```

`Type=oneshot` convient à un programme qui fait son travail et s'arrête. `Nice=19` et `IOSchedulingClass=idle` lui donnent la plus basse priorité pour le processeur et pour le disque. Il n'y a pas de section `[Install]` : ce service ne démarre pas avec la machine, c'est la minuterie qui le lancera.

7. Créez la minuterie, dans `/etc/systemd/system/sauvegarde.timer` :

```
[Unit]
Description=Sauvegarde des documents, chaque nuit

[Timer]
OnCalendar=*-*-* 02:30:00
RandomizedDelaySec=15min
Persistent=true

[Install]
WantedBy=timers.target
```

8. Essayez d'abord le service seul. C'est tout l'avantage du découpage en deux unités : on teste la tâche sans attendre l'heure.

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl start sauvegarde.service
gary@serveur:~$ journalctl -u sauvegarde -o cat
Starting sauvegarde.service - Sauvegarde des documents de Gary...
2026-10-04 21:00:25 [INFO] début : /home/gary/Documents vers /home/gary/sauvegarde…
2026-10-04 21:00:25 [INFO] archive créée (4.0K)
2026-10-04 21:00:25 [INFO] fin de la sauvegarde
sauvegarde.service: Deactivated successfully.
Finished sauvegarde.service - Sauvegarde des documents de Gary.
```

Grâce à l'option `-v` du script, chaque étape figure dans le journal, encadrée par les messages de systemd.

9. Activez la minuterie. C'est elle qu'on active, pas le service.

```
gary@serveur:~$ sudo systemctl enable --now sauvegarde.timer
Created symlink /etc/systemd/system/timers.target.wants/sauvegarde.timer → /etc/
systemd/system/sauvegarde.timer.
gary@serveur:~$ systemctl status sauvegarde.timer
● sauvegarde.timer - Sauvegarde des documents, chaque nuit
     Loaded: loaded (/etc/systemd/system/sauvegarde.timer; enabled; preset: enabled)
     Active: active (waiting) since Sun 2026-10-04 21:00:26 UTC; 731ms ago
    Trigger: Mon 2026-10-05 02:38:22 UTC; 5h 37min left
   Triggers: ● sauvegarde.service
```

La ligne `Trigger` annonce la prochaine échéance : 2 h 38, soit 2 h 30 plus les huit minutes tirées au sort par `RandomizedDelaySec=`.

Provoquer un échec

10. Renommez provisoirement `~/Documents`, puis lancez le service :

```
gary@serveur:~$ mv ~/Documents ~/Documents.absent
gary@serveur:~$ sudo systemctl start sauvegarde.service
Job for sauvegarde.service failed because the control process exited with error code.
See "systemctl status sauvegarde.service" and "journalctl -xeu sauvegarde.service" for
details.
gary@serveur:~$ systemctl --failed
  UNIT                LOAD   ACTIVE SUB    DESCRIPTION
● sauvegarde.service loaded failed failed Sauvegarde des documents de Gary
[…]
gary@serveur:~$ journalctl -u sauvegarde -n 4 -o cat
sauvegarde : dossier introuvable : /home/gary/Documents
sauvegarde.service: Main process exited, code=exited, status=1/FAILURE
sauvegarde.service: Failed with result 'exit-code'.
Failed to start sauvegarde.service - Sauvegarde des documents de Gary.
```

Tout y est : le message du script, son code de retour, le verdict de systemd. Une sauvegarde qui échoue en silence est pire que pas de sauvegarde du tout. Ici, `systemctl --failed` la dénonce, et le journal dit pourquoi.

11. Remettez le dossier en place et effacez l'échec :

```
mv ~/Documents.absent ~/Documents
sudo systemctl reset-failed sauvegarde
```

Vérifier

- `systemctl list-timers sauvegarde.timer` affiche la prochaine échéance.
- `crontab -l` ne contient plus la ligne d'essai.
- Demain matin, `journalctl -u sauvegarde --since yesterday` montrera la sauvegarde de la nuit, et `ls ~/sauvegardes` une archive de plus.

Exercices

Exercice 1 — Interroger le journal

Donnez la commande qui affiche :

1. les cinquante dernières lignes du journal, puis les suivantes à mesure qu'elles arrivent ;
2. les messages du service `ssh` depuis vingt-quatre heures ;
3. les erreurs du démarrage précédent ;

4. les messages du noyau depuis le dernier démarrage ;
5. le nombre de connexions SSH acceptées depuis sept jours ;
6. la place occupée par le journal.

Exercice 2 — Écrire des lignes de cron

Donnez les cinq champs pour chaque moment.

1. Toutes les dix minutes.
2. Tous les jours à 7 h 30.
3. Le 15 de chaque mois à minuit.
4. Les lundis et jeudis à 18 h.
5. Une fois par heure entre 9 h et 17 h, du lundi au vendredi.
6. Le 5 de chaque mois à 3 h 45.

Écrivez ensuite les mêmes moments sous forme d'expressions `OnCalendar=`, et vérifiez-les avec `systemd-analyze calendar`.

Exercice 3 — Diagnostic

Que faites-vous dans chacune de ces situations ?

1. Une tâche de votre table cron devrait tourner chaque nuit. Vous doutez qu'elle le fasse.
2. Elle tourne, d'après le journal, mais ne produit pas le fichier attendu. Lancée à la main, la commande fonctionne.
3. La partition racine d'un serveur est pleine. Vous soupçonnez les journaux.
4. Vous voulez savoir quand `apt-daily.service` a tourné pour la dernière fois, et quand il tournera de nouveau.
5. Un script posé dans `/etc/cron.daily` n'est jamais exécuté.

Exercice 4 — Une minuterie

Écrivez le service et la minuterie qui lancent `/usr/local/bin/controle` :

1. une minute après le démarrage de la machine ;
2. puis toutes les cinq minutes ;
3. avec un retard aléatoire d'au plus trente secondes ;
4. sous le compte `veilleur`, avec un système de fichiers en lecture seule.

Vérifiez vos deux fichiers avec `systemd-analyze verify`, puis observez les échéances avec `systemctl list-timers`.

Exercice 5 — Une règle logrotate

Une application écrit dans `/var/log/monappli/acces.log` et `/var/log/monappli/erreurs.log`. Écrivez la règle qui :

1. fait tourner ces fichiers chaque jour, sauf s'ils sont vides ;
2. en garde soixante ;
3. les compresse, sauf le plus récent ;
4. recrée les fichiers avec les droits 640, pour le compte `monappli` et le groupe `adm` ;
5. ajoute la date au nom des anciens fichiers plutôt qu'un numéro (cherchez la directive dans `man logrotate`) ;
6. envoie le signal `SIGUSR1` à l'application après la rotation, pour qu'elle rouvre ses fichiers.

Essayez-la avec `logrotate -d`.

Pour aller plus loin

Sur la machine. `man journalctl` pour toutes les options de filtrage. `man systemd.time` décrit la syntaxe des dates, des durées et des expressions de calendrier. `man 5 crontab` celle des tables de cron. `man logrotate` liste toutes les directives.

Prolongements. La directive `OnFailure=`, dans la section `[Unit]` d'un service, désigne une unité à lancer quand il échoue : c'est le point de départ d'une alerte par courriel ou par messagerie. `loginctl enable-linger` permet à un utilisateur de faire tourner ses propres minuteries, avec `systemctl --user`, même quand il n'est pas connecté. Le chapitre 26 montrera comment surveiller un parc entier.

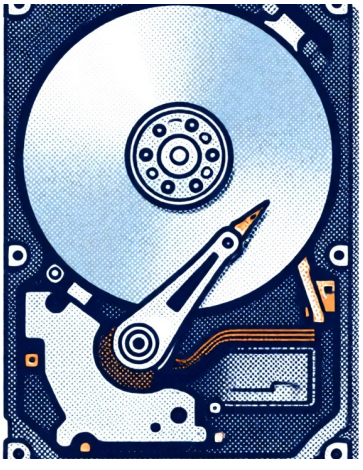
En ligne. Le site `crontab.guru` traduit une expression de cron en clair, au fur et à mesure qu'on la tape.

PARTIE V

Stockage et systèmes de fichiers

Du disque physique au fichier : partitionner, formater, monter, agrandir, chiffrer et sauvegarder.





CHAPITRE 14

Disques, partitions et systèmes de fichiers

Entre un disque neuf et un dossier où ranger des fichiers, il y a quatre gestes. Partitionner, formater, monter, inscrire dans fstab. Ce chapitre les fait dans l'ordre, sur un disque que vous pouvez vous permettre d'effacer.

DANS CE CHAPITRE

- Nommer les étapes qui séparent un disque d'un dossier utilisable.
- Reconnaître les disques et les partitions d'une machine avec `lsblk` et `blkid`.
- Partitionner un disque avec `fdisk` ou `parted`.
- Choisir un système de fichiers et le créer.
- Monter et démonter un système de fichiers, avec les options utiles.
- Rendre un montage permanent dans `/etc/fstab`, sans mettre le démarrage en danger.
- Créer et régler un espace d'échange.
- Vérifier un système de fichiers et lire l'état de santé d'un disque.

> **AVANT DE COMMENCER**
>
> Les chapitres 4, 6 et 11. Il vous faut une machine virtuelle Ubuntu Server à laquelle vous ajouterez un second disque de 10 Go : tout ce chapitre efface des disques, et ce ne doit pas être le vôtre. Les sorties ont été relevées sur un serveur installé avec les choix par défaut du chapitre 2.

14.1 Du disque au dossier

Un disque neuf est une suite de blocs numérotés, sans aucune organisation. Avant d'y écrire un fichier, il faut franchir quatre étapes.

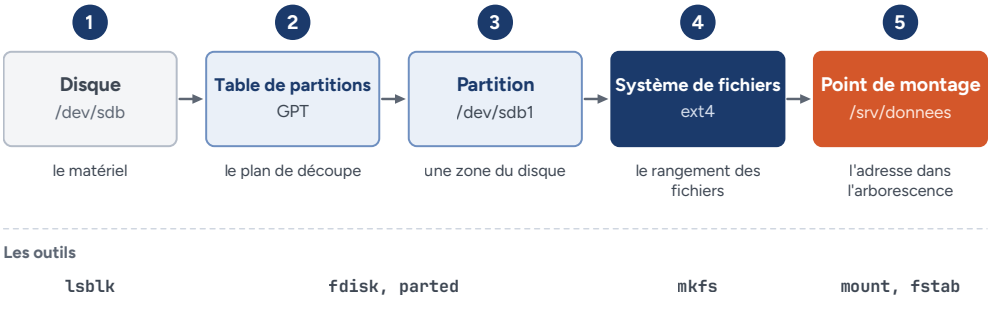


Figure 14.1 — Du disque au point de montage.

Le **disque** apparaît comme un fichier spécial dans `/dev`. Son nom dépend de la façon dont il est branché :

Nom	Type de disque
`/dev/sda`, `/dev/sdb`…	Disques SATA, SCSI, USB. La lettre suit l'ordre de détection
`/dev/nvme0n1`	SSD NVMe
`/dev/vda`	Disque d'une machine virtuelle (virtio)
`/dev/mmcblk0`	Carte SD

La **table de partitions**, écrite au début du disque, décrit son découpage.

Une **partition** est une zone du disque. Elle porte le nom du disque suivi d'un numéro : `/dev/sda1`, `/dev/sda2`. Sur les disques dont le nom finit par un chiffre, on intercale un `p` : `/dev/nvme0n1p1`.

Le **système de fichiers** organise l'intérieur de la partition : où commence chaque fichier, à qui il appartient, quels blocs sont libres. On le crée en *formatant* la partition.

Le **point de montage** est le dossier de l'arborescence par lequel on accède à ce système de fichiers.

> **ATTENTION · Trois vérifications avant d'écrire**
>
> Dans ce chapitre, une faute de frappe efface un disque. Avant chaque commande qui écrit, relisez le nom du périphérique, vérifiez avec `lsblk` que c'est le bon, et assurez-vous que rien d'important ne s'y trouve. `/dev/sda` est en général le disque du système. Dans nos exemples, on travaille sur `/dev/sdb`.

14.2 Reconnaître les disques

`lsblk` liste les périphériques de stockage sous forme d'arbre :

```
gary@serveur:~$ lsblk
NAME                      MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sda                         8:0    0  25G  0 disk
├─sda1                      8:1    0   1G  0 part /boot/efi
├─sda2                      8:2    0   2G  0 part /boot
└─sda3                      8:3    0  22G  0 part
  └─ubuntu--vg-ubuntu--lv 252:0    0  11G  0 lvm  /
sdb                         8:16   0  10G  0 disk
```

La machine a deux disques. `sda` porte le système, découpé en trois partitions par l'installateur : la partition EFI, `/boot`, et une troisième confiée à LVM, que nous étudierons au chapitre 15. `sdb` est le disque ajouté : 10 Go, aucune partition.

Avec `-f`, `lsblk` montre ce que contient chaque partition :

```
gary@serveur:~$ lsblk -f
NAME FSTYPE FSVER LABEL UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sda
├─sda1
│    vfat   FAT32       288A-FB74                            1013.3M     1% /boot/efi
├─sda2
│    ext4   1.0         4f1b3ff8-e317-43bc-a253-ca09a30deac1      1.7G     3% /boot
└─sda3
     LVM2_m LVM2        nLlSol-UKuj-fLEx-uwHd-6cra-h0LT-CpcD9u
  └─ubuntu--vg-ubuntu--lv
     ext4   1.0         b0cf6115-3360-4689-a417-44e80c1dfcf1      6.5G    34% /
sdb
```

La colonne `UUID` est la plus importante. Un **UUID** est un identifiant unique, tiré au hasard quand on crée le système de fichiers. Il le désigne sans ambiguïté, quel que soit le nom que le noyau donne au disque. Ce nom, lui, peut changer : branchez un disque de plus, et `sdb` devient parfois `sdc` au démarrage suivant.

Commande	Ce qu'elle montre
`lsblk`	L'arbre des disques et des partitions, avec les points de montage
`lsblk -f`	Les systèmes de fichiers, leurs étiquettes et leurs UUID
`sudo blkid /dev/sdb1`	Les identifiants d'une partition, sur une ligne
`sudo fdisk -l /dev/sda`	La table de partitions d'un disque
`df -hT`	L'espace occupé et libre de chaque système de fichiers monté

14.3 Deux tables de partitions : MBR et GPT

	MBR	GPT
Origine	PC DOS 2.0, en 1983	Norme UEFI
Taille maximale du disque	2 Tio	Sans limite pratique
Nombre de partitions	4 partitions principales	128
En cas de dommage	Table perdue	Une copie de secours en fin de disque

`fdisk -l` indique le type de table à la ligne `Disklabel type : gpt`, ou `dos` pour le MBR.

```
gary@serveur:~$ sudo fdisk -l /dev/sda
Disk /dev/sda: 25 GiB, 26843545600 bytes, 52428800 sectors
Disk model: QEMU HARDDISK
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 3E1678F6-1E39-4267-B06E-8430A88FA4B9

Device       Start      End  Sectors Size Type
/dev/sda1     2048  2099199  2097152   1G EFI System
/dev/sda2  2099200  6293503  4194304   2G Linux filesystem
/dev/sda3  6293504 52428766 46135263  22G Linux filesystem
```

Choisissez GPT. Le MBR ne se justifie plus que pour une clé USB destinée à de très vieilles machines.

14.4 Partitionner

Avec fdisk

`fdisk` dialogue. On lui donne des ordres d'une lettre, et rien n'est écrit sur le disque avant l'ordre `w`. Créons une table GPT, puis une partition de 4 Go :

```
gary@serveur:~$ sudo fdisk /dev/sdb

Welcome to fdisk (util-linux 2.39.3).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS (MBR) disklabel with disk identifier 0x46a816f7.

Command (m for help): g
Created a new GPT disklabel (GUID: 1D4A2570-FA35-4B34-B79D-07571E29C82D).

Command (m for help): n
Partition number (1-128, default 1):
First sector (2048-20971486, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-20971486, default 20969471): +4G

Created a new partition 1 of type 'Linux filesystem' and of size 4 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Aux deux premières questions, la touche Entrée accepte la valeur proposée. À la troisième, `+4G` donne la taille. Une simple pression sur Entrée aurait pris tout le disque.

Ordre	Effet
`m`	Afficher l'aide
`p`	Afficher la table telle qu'elle sera écrite
`g`	Créer une table GPT vide
`n`	Créer une partition
`d`	Supprimer une partition

Ordre	Effet
`t`	Changer le type d'une partition
`w`	Écrire sur le disque et quitter
`q`	Quitter sans rien écrire

Tant que vous n'avez pas tapé `w`, `q` annule tout. `cfdisk` propose les mêmes opérations dans un menu plein écran, plus confortable pour un usage occasionnel.

```
gary@serveur:~$ lsblk /dev/sdb
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sdb      8:16   0  10G  0 disk
└─sdb1   8:17   0   4G  0 part
```

Avec parted

`parted` fait la même chose en une ligne de commande, sans dialogue. C'est l'outil des scripts. L'option `-s` supprime les questions :

```
sudo parted -s /dev/sdb mklabel gpt
sudo parted -s /dev/sdb mkpart donnees ext4 1MiB 4GiB
```

`mkpart` attend un nom, un type indicatif, un début et une fin. Le début à `1MiB` aligne la partition comme le fait `fdisk`. Les positions s'écrivent aussi en pourcentage du disque : l'atelier s'en servira.

Avec `parted`, il n'y a pas d'ordre `w`. Chaque commande est appliquée sur-le-champ.

14.5 Choisir un système de fichiers

Système	Points forts	Usage
`ext4`	Éprouvé, bien outillé, peut être agrandi et réduit	Le choix par défaut d'Ubuntu, et le bon dans presque tous les cas
`xfs`	À l'aise avec les très gros volumes et les gros fichiers	Stockage volumineux, sauvegardes. Ne peut pas être réduit
`btrfs`	Instantanés, compression, plusieurs disques	Postes de travail, quand on veut revenir en arrière après une mise à jour

Système	Points forts	Usage
`zfs`	Instantanés, contrôle d'intégrité, gestion des disques intégrée	Serveurs de stockage
`vfat`	Lu par tous les systèmes	Partition EFI, clés USB. Pas de fichier de plus de 4 Go
`exfat`	Lu par tous les systèmes récents, sans la limite des 4 Go	Clés et disques externes
`ntfs`	Le système de fichiers de Windows	Disques partagés avec Windows

> **REPÈRE HISTORIQUE · Vingt ans d'ext**
>
> ext4 est entré dans le noyau Linux à la fin de 2008, sous la conduite de Theodore Ts'o. Il prolonge ext2 et ext3, les systèmes de fichiers de Linux depuis les années 1990. xfs est plus ancien : Silicon Graphics l'a créé en 1993 pour ses stations de travail, avant de le porter sous Linux. C'est le système par défaut de Red Hat depuis 2014.

Deux remarques sur les systèmes les plus riches.

btrfs gère lui-même plusieurs disques, mais ses modes RAID 5 et RAID 6 sont toujours classés instables par ses développeurs. On s'en tient à un seul disque ou au miroir.

zfs a une place à part sur Ubuntu. Sa licence empêche de l'intégrer au noyau Linux officiel, mais Canonical le livre tout de même avec ses noyaux. Il suffit d'installer les outils (`sudo apt install zfsutils-linux`), et l'installateur d'Ubuntu Desktop 24.04 propose de nouveau une installation sur zfs.

Sans raison précise de faire autrement, prenez ext4.

14.6 Créer le système de fichiers

Les commandes de formatage s'appellent `mkfs.` suivi du type. `-L` donne une étiquette.

```
gary@serveur:~$ sudo mkfs.ext4 -L essai /dev/sdb1
mke2fs 1.47.0 (5-Feb-2023)
Discarding device blocks: done
Creating filesystem with 1048576 4k blocks and 262144 inodes
Filesystem UUID: ddac814b-7863-4e5c-ba98-fd9c35abacff
Superblock backups stored on blocks:
```

```
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

La sortie annonce l'UUID et le nombre d'inodes : 262 144. Ce système de fichiers ne pourra donc pas contenir plus de 262 144 fichiers, quelle que soit leur taille. C'est rarement un problème. `df -i` montre où l'on en est.

```
sudo mkfs.xfs -L archives /dev/sdb2     # xfs
sudo mkfs.vfat -F 32 -n CLE /dev/sdc1   # FAT32, pour une clé USB
```

ATTENTION · mkfs ne demande pas toujours confirmation

Sur une partition vide, `mkfs.ext4` s'exécute sans poser de question. Il n'en pose une que s'il reconnaît un système de fichiers existant. Le formatage ne prend qu'une seconde, et ce qui se trouvait là n'est plus accessible.

14.7 Monter et démonter

Le principe

Monter un système de fichiers, c'est le greffer sur un dossier de l'arborescence. Ce dossier, le point de montage, doit exister. Il est en général vide.

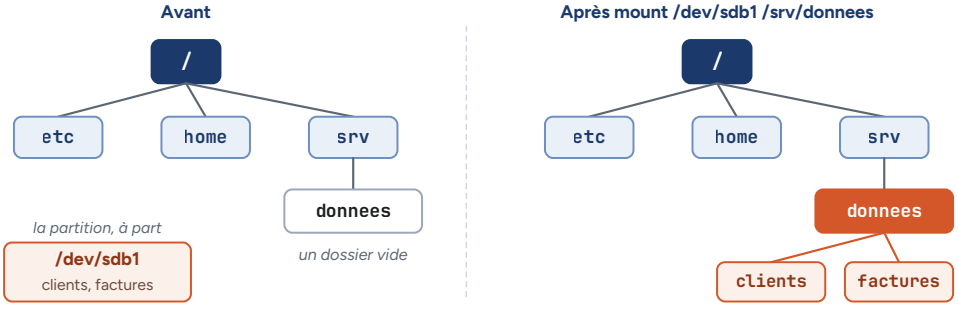


Figure 14.2 — Avant le montage, le dossier est vide. Après, il donne accès au contenu de la partition.

```
gary@serveur:~$ sudo mkdir /mnt/essai
gary@serveur:~$ sudo mount /dev/sdb1 /mnt/essai
gary@serveur:~$ findmnt /mnt/essai
```

```
TARGET     SOURCE    FSTYPE OPTIONS
/mnt/essai /dev/sdb1 ext4   rw,relatime
gary@serveur:~$ df -h /mnt/essai
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       3.9G   24K  3.7G   1% /mnt/essai
```

La partition fait 4 Go et `df` n'en annonce que 3,7 de disponibles. La différence tient aux structures du système de fichiers, et à une réserve de 5 % qu'ext4 garde pour `root`. Grâce à elle, l'administrateur peut encore travailler quand un utilisateur a rempli le disque. Sur un gros disque de données, on la réduit : `sudo tune2fs -m 1 /dev/sdb1`.

Le système de fichiers neuf appartient à `root`. Pour y écrire avec votre compte, changez le propriétaire de sa racine. Puis faites l'expérience suivante :

```
gary@serveur:~$ sudo chown gary: /mnt/essai
gary@serveur:~$ echo "premier fichier" > /mnt/essai/note.txt
gary@serveur:~$ ls /mnt/essai
lost+found  note.txt
gary@serveur:~$ sudo umount /mnt/essai
gary@serveur:~$ ls /mnt/essai
gary@serveur:~$ sudo mount /dev/sdb1 /mnt/essai
gary@serveur:~$ ls /mnt/essai
lost+found  note.txt
```

Une fois la partition démontée, le dossier est vide. Le fichier n'a pas disparu : il est sur la partition, pas dans le dossier. Le dossier `lost+found` est créé par ext4. La vérification du système de fichiers y range les fragments qu'elle ne sait plus rattacher à un fichier.

La commande s'écrit `umount`, sans `n`.

« Target is busy »

```
gary@serveur:~$ cd /mnt/essai
gary@serveur:/mnt/essai$ sudo umount /mnt/essai
umount: /mnt/essai: target is busy.
gary@serveur:/mnt/essai$ sudo fuser -vm /mnt/essai
                     USER        PID ACCESS COMMAND
/mnt/essai:          root     kernel mount /mnt/essai
                     gary       1287 ..c.. bash
                     root       1416 ..c.. sudo
                     root       1417 ..c.. sudo
```

On ne peut pas démonter un système de fichiers en cours d'utilisation. `fuser -vm` désigne les coupables. Ici, c'est notre propre shell, et la commande `sudo` lancée de-

puis lui : leur répertoire courant se trouve sur la partition, ce que signale la lettre `c`. Il suffit d'en sortir avec `cd`.

Les options de montage

L'option `-o` règle le comportement du montage :

```
gary@serveur:~$ sudo mount -o ro,noexec /dev/sdb1 /mnt/essai
gary@serveur:~$ touch /mnt/essai/autre.txt
touch: cannot touch '/mnt/essai/autre.txt': Read-only file system
gary@serveur:~$ sudo mount -o remount,rw /mnt/essai
gary@serveur:~$ findmnt -no OPTIONS /mnt/essai
rw,noexec,relatime
```

Option	Effet
`ro`, `rw`	Lecture seule, lecture et écriture
`noexec`	Interdit d'exécuter les programmes qui s'y trouvent
`nosuid`	Ignore les bits setuid et setgid (chapitre 6)
`nodev`	Ignore les fichiers de périphériques
`noatime`	N'enregistre pas la date du dernier accès aux fichiers
`remount`	Change les options d'un système de fichiers déjà monté
`defaults`	Les réglages habituels : `rw`, exécution permise

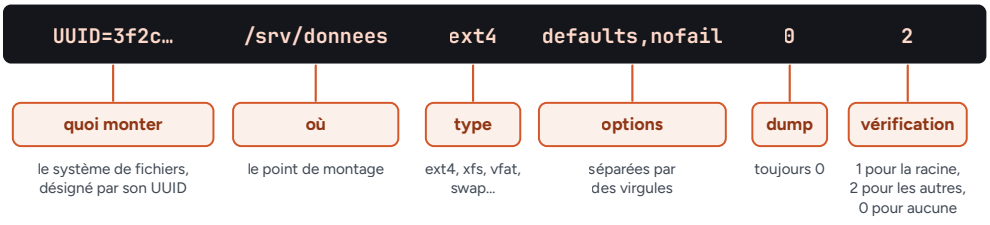
`remount` évite de démonter. C'est le moyen de repasser en écriture une partition montée en lecture seule, ou l'inverse.

Tout cela ne dure que jusqu'au prochain redémarrage. Pour un montage permanent, il faut l'écrire dans un fichier.

14.8 /etc/fstab : les montages permanents

Au démarrage, Ubuntu monte ce que décrit `/etc/fstab`. Voici celui de notre serveur, sans ses commentaires :

```
/dev/disk/by-id/dm-uuid-LVM-YhfIs5m2…U6FV / ext4 defaults 0 1
/dev/disk/by-uuid/4f1b3ff8-e317-43bc-a253-ca09a30deac1 /boot ext4 defaults 0 1
/dev/disk/by-uuid/288A-FB74 /boot/efi vfat defaults 0 1
/swap.img       none    swap    sw      0       0
```



Une ligne par système de fichiers, six champs séparés par des espaces.

Figure 14.3 — Les six champs d'une ligne de fstab.

Le premier champ désigne le système de fichiers. Trois écritures se valent : `UUID=…`, le chemin `/dev/disk/by-uuid/…` qu'emploie l'installateur, ou `LABEL=…` pour une étiquette. N'écrivez jamais `/dev/sdb1` : ce nom peut changer d'un démarrage à l'autre.

Le sixième champ règle la vérification au démarrage : `1` pour la racine, `2` pour les autres systèmes de fichiers, `0` pour ne rien vérifier.

Ajouter une ligne

Relevons l'UUID, puis ajoutons la ligne avec `sudo nano /etc/fstab` :

```
gary@serveur:~$ sudo blkid /dev/sdb1
/dev/sdb1: LABEL="essai" UUID="ddac814b-7863-4e5c-ba98-fd9c35abacff" […] TYPE="ext4" […]
gary@serveur:~$ tail -n 1 /etc/fstab
UUID=ddac814b-7863-4e5c-ba98-fd9c35abacff /mnt/essai ext4 defaults,nofail 0 2
```

Ne redémarrez pas pour voir si elle fonctionne. Trois commandes la vérifient sans risque :

```
gary@serveur:~$ sudo findmnt --verify
none
   [W] non-bind mount source /swap.img is a directory or regular file
   [W] your fstab has been modified, but systemd still uses the old version;
       use 'systemctl daemon-reload' to reload

0 parse errors, 0 errors, 2 warnings
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo mount -a
gary@serveur:~$ findmnt /mnt/essai
TARGET     SOURCE    FSTYPE OPTIONS
/mnt/essai /dev/sdb1 ext4   rw,relatime
```

`findmnt --verify` relit le fichier et signale les erreurs. Le premier avertissement concerne le fichier d'échange et n'a pas d'importance. Le second rappelle que sys-

temd garde en mémoire l'ancienne version du fichier : d'où le `daemon-reload`. `mount -a` monte ensuite tout ce que `fstab` décrit et qui ne l'est pas encore. S'il ne dit rien, la ligne est bonne.

Avec une ligne fautive, `findmnt --verify` aurait prévenu :

```
/mnt/rien
   [E] unreachable on boot required target: No such file or directory
   [E] unreachable on boot required source: UUID=0000
```

Quand fstab empêche de démarrer

Au démarrage, systemd attend chaque disque cité dans `fstab`. S'il en manque un, il patiente une minute et demie, puis abandonne le démarrage normal. L'écran affiche alors :

```
You are in emergency mode. After logging in, type "journalctl -xb" to view
system logs, "systemctl reboot" to reboot, or "exit"
to continue bootup.
Press Enter for maintenance
(or press Control-D to continue):
```

Un disque de données débranché, un UUID mal recopié, et le serveur ne démarre plus. S'il se trouve dans un centre de données, vous ne pouvez plus vous y connecter.

L'option `nofail` évite cette situation. Elle dit à systemd que ce système de fichiers n'est pas indispensable. S'il manque, le démarrage continue, et le journal en garde la trace :

```
Timed out waiting for device … /dev/disk/by-uuid/b438851b-bfda-4906-b540-980a0c417ca8.
Dependency failed for srv-donnees.mount - /srv/donnees.
```

> **À RETENIR · nofail partout, sauf pour le système**
>
> Mettez `nofail` sur toutes les lignes que vous ajoutez à `fstab`. Seuls la racine, `/boot` et `/boot/efi` s'en passent, car sans eux la machine ne peut de toute façon pas fonctionner.

Si vous vous retrouvez malgré tout devant le mode d'urgence, la réparation est courte. Sur Ubuntu, le compte `root` étant verrouillé, la touche Entrée ouvre directement un shell :

```
root@serveur:~# nano /etc/fstab
root@serveur:~# systemctl daemon-reload
root@serveur:~# systemctl default
```

Corrigez la ligne fautive, ou neutralisez-la d'un `#`. `systemctl default` reprend le démarrage là où il s'était arrêté. L'atelier vous fera vivre cette panne et sa réparation.

14.9 L'espace d'échange

L'**espace d'échange** (en anglais *swap*) est une zone du disque où le noyau range des pages de mémoire peu utilisées quand la mémoire vive vient à manquer. Il évite qu'un pic de consommation se termine par la mort d'un processus. Il ne remplace pas la mémoire : un disque est des milliers de fois plus lent.

```
gary@serveur:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           1.4Gi       284Mi       996Mi       4.9Mi       325Mi       1.1Gi
Swap:          2.0Gi          0B       2.0Gi
gary@serveur:~$ swapon --show
NAME      TYPE SIZE USED PRIO
/swap.img file   2G   0B   -2
```

Ubuntu n'utilise plus de partition d'échange. L'installateur crée un fichier, `/swap.img`, déclaré dans `fstab`. Un fichier s'agrandit ou se remplace bien plus facilement qu'une partition :

```
gary@serveur:~$ sudo fallocate -l 512M /swap2.img
gary@serveur:~$ sudo chmod 600 /swap2.img
gary@serveur:~$ sudo mkswap /swap2.img
Setting up swapspace version 1, size = 512 MiB (536866816 bytes)
no label, UUID=39f6e2ce-b00c-47a6-ab04-d9884c2f94bf
gary@serveur:~$ sudo swapon /swap2.img
gary@serveur:~$ swapon --show
NAME       TYPE SIZE USED PRIO
/swap.img  file   2G   0B   -2
/swap2.img file 512M   0B   -3
```

Le `chmod 600` n'est pas une formalité : ce fichier contiendra des morceaux de la mémoire des programmes, mots de passe compris. Pour garder ce second fichier après un redémarrage, ajoutez à `fstab` une ligne sur le modèle de celle de `/swap.img`. `sudo swapoff /swap2.img` le désactive.

Le paramètre `vm.swappiness` règle l'empressement du noyau à se servir de l'échange. Il vaut 60 par défaut. Une valeur plus basse l'incite à garder les programmes en mémoire :

```
cat /proc/sys/vm/swappiness
echo 'vm.swappiness=10' | sudo tee /etc/sysctl.d/99-echange.conf
sudo sysctl --system
```

Sur un serveur, un échange qui se remplit est un signal. La machine manque de mémoire, et la vraie réponse est d'en ajouter ou de limiter les services gourmands (chapitre 12).

14.10 Vérifier et surveiller

fsck

`fsck` vérifie la cohérence d'un système de fichiers et répare ce qu'il peut. Il travaille sur un système de fichiers démonté, et refuse le contraire :

```
gary@serveur:~$ sudo fsck /dev/sdb1
fsck from util-linux 2.39.3
e2fsck 1.47.0 (5-Feb-2023)
/dev/sdb1 is mounted.
e2fsck: Cannot continue, aborting.
gary@serveur:~$ sudo umount /mnt/essai
gary@serveur:~$ sudo fsck -f /dev/sdb1
fsck from util-linux 2.39.3
e2fsck 1.47.0 (5-Feb-2023)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
essai: 13/262144 files (0.0% non-contiguous), 36943/1048576 blocks
```

Sans `-f`, `fsck` se contente d'un coup d'œil si le système de fichiers est marqué comme sain. En temps normal, vous n'aurez pas à le lancer. ext4 tient un journal de ses opérations en cours, et le rejoue de lui-même après une coupure de courant.

La racine ne peut pas être démontée pendant que le système tourne. Pour la vérifier, ajoutez `fsck.mode=force` aux paramètres du noyau pour un seul démarrage, avec la touche `e` du menu de GRUB (chapitre 11). La vérification a lieu avant le montage, et son compte rendu se trouve ensuite dans `/run/initramfs/fsck.log`.

L'état de santé d'un disque

Les disques tiennent un journal de leur propre état, appelé SMART. `smartctl`, du paquet `smartmontools`, le lit. Voici l'essentiel de ce que répond le SSD d'un ordinateur portable :

```
gary@vostro:~$ sudo smartctl -a /dev/nvme0n1
[…]
Model Number:                       SKHynix_HFS512GEJ4X112N
[…]
SMART overall-health self-assessment test result: PASSED

Critical Warning:                   0x00
Temperature:                        38 Celsius
Available Spare:                    100%
Available Spare Threshold:          10%
Percentage Used:                    1%
Data Units Written:                 17,528,534 [8.97 TB]
Power On Hours:                     7,688
Unsafe Shutdowns:                   74
Media and Data Integrity Errors:    0
```

Ligne	Ce qu'il faut y lire
`self-assessment`	Le verdict du disque sur lui-même. Autre chose que `PASSED` : sauvegardez et remplacez
`Critical Warning`	Doit rester à `0x00`
`Available Spare`	La réserve de cellules de rechange. Elle ne doit pas approcher du seuil
`Percentage Used`	L'usure estimée par le fabricant. Au-delà de 100 %, le disque a dépassé sa durée de vie prévue
`Media and Data Integrity Errors`	Les erreurs de données. Doit rester à 0

Pour un disque dur classique, les lignes à surveiller s'appellent `Reallocated_Sector_Ct` et `Current_Pending_Sector` : des secteurs défectueux remplacés, ou en attente de l'être. Qu'elles se mettent à augmenter, et le disque est en fin de vie.

Les disques d'une machine virtuelle ne répondent pas à `smartctl`. C'est sur la machine physique qu'on les surveille.

En résumé

- Disque, table de partitions, partition, système de fichiers, point de montage : dans cet ordre.
- `lsblk` avant toute commande qui écrit. `lsblk -f` donne les UUID.
- `fdisk` dialogue et n'écrit qu'à l'ordre `w`. `parted -s` s'emploie dans les scripts.
- GPT pour la table, ext4 pour le système de fichiers, sauf raison précise.
- `mount` greffe un système de fichiers sur un dossier, `umount` le détache.
- Dans `fstab`, on désigne les systèmes de fichiers par leur UUID, et on ajoute `nofail` à tout ce qui n'est pas le système.
- `findmnt --verify`, `systemctl daemon-reload` puis `mount -a` avant tout redémarrage.
- Un `fstab` fautif mène au mode d'urgence : Entrée, corriger, `systemctl default`.
- L'espace d'échange d'Ubuntu est un fichier, `/swap.img`.

Atelier — Ajouter un disque à un serveur

Objectif : partir d'un disque vierge et arriver à deux systèmes de fichiers montés à chaque démarrage, puis vérifier que l'absence du disque n'empêche pas le serveur de démarrer. Comptez une heure.

Préparer

1. Éteignez la machine virtuelle. Dans votre logiciel de virtualisation, ajoutez-lui un disque de 10 Go, puis redémarrez-la. Prenez un instantané.
2. Repérez le nouveau disque avec `lsblk` : 10 Go, sans partition. Dans la suite, c'est `/dev/sdb`. Si vous avez suivi les exemples du chapitre sur ce même disque, démontez `/mnt/essai`, retirez sa ligne de `fstab` et effacez les signatures :

```
gary@serveur:~$ sudo wipefs -a /dev/sdb1 /dev/sdb
/dev/sdb1: 2 bytes were erased at offset 0x00000438 (ext4): 53 ef
/dev/sdb: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52 54
/dev/sdb: 8 bytes were erased at offset 0x27ffffe00 (gpt): 45 46 49 20 50 41 52 54
/dev/sdb: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
```

`wipefs` efface les quelques octets qui permettent de reconnaître un système de fichiers ou une table de partitions. On y voit au passage la copie de secours de la table GPT, en fin de disque.

Partitionner et formater

3. Créez une table GPT et deux partitions, l'une sur 60 % du disque, l'autre sur le reste :

```
gary@serveur:~$ sudo parted -s /dev/sdb mklabel gpt
gary@serveur:~$ sudo parted -s /dev/sdb mkpart donnees ext4 1MiB 60%
gary@serveur:~$ sudo parted -s /dev/sdb mkpart archives xfs 60% 100%
gary@serveur:~$ sudo parted /dev/sdb print
Model: QEMU QEMU HARDDISK (scsi)
Disk /dev/sdb: 10.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name      Flags
 1      1049kB  6442MB  6441MB               donnees
 2      6442MB  10.7GB  4294MB               archives
```

4. Formatez la première en ext4, la seconde en xfs, et relevez les UUID :

```
gary@serveur:~$ sudo mkfs.ext4 -q -L donnees /dev/sdb1
gary@serveur:~$ sudo mkfs.xfs -q -L archives /dev/sdb2
gary@serveur:~$ lsblk -f /dev/sdb
NAME FSTYPE FSVER LABEL    UUID                                 FSAVAIL FSUSE% MOUNTPOINTS
sdb
├─sdb1
│    ext4   1.0   donnees  b438851b-bfda-4906-b540-980a0c417ca8
└─sdb2
     xfs          archives 03ec4488-3e41-4d87-b57d-b0bd900d6fdf
```

Monter pour de bon

5. Créez les points de montage. Des données servies par la machine ont leur place sous `/srv` (chapitre 4). Gardez une copie de `fstab` avant d'y toucher.

```
sudo mkdir /srv/donnees /srv/archives
sudo cp /etc/fstab /etc/fstab.avant
```

6. Ajoutez les deux lignes à `/etc/fstab`, avec vos propres UUID :

```
UUID=b438851b-bfda-4906-b540-980a0c417ca8 /srv/donnees ext4 defaults,nofail 0 2
UUID=03ec4488-3e41-4d87-b57d-b0bd900d6fdf /srv/archives xfs defaults,nofail 0 2
```

7. Vérifiez, rechargez, montez :

```
gary@serveur:~$ sudo findmnt --verify
[…]
0 parse errors, 0 errors, 2 warnings
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo mount -a
gary@serveur:~$ df -hT /srv/donnees /srv/archives
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/sdb1      ext4  5.9G   24K  5.6G   1% /srv/donnees
/dev/sdb2      xfs   4.0G  110M  3.9G   3% /srv/archives
```

8. Donnez `/srv/donnees` à votre compte, écrivez-y un fichier, redémarrez, et vérifiez que le fichier est toujours là.

Retirer le disque

9. Éteignez la machine, détachez le disque de 10 Go sans le supprimer, et démarrez. Le serveur doit répondre comme d'habitude. Lors de l'essai, la connexion SSH était disponible au bout de dix secondes. Constatez ensuite :

```
findmnt /srv/donnees                    # rien : le disque n'est pas là
journalctl -b -p warning | grep srv     # systemd a attendu, puis renoncé
```

Provoquer la panne

10. Toujours sans le disque, retirez `nofail` de la ligne de `/srv/donnees` et redémarrez. Après une minute et demie d'attente, la console de la machine virtuelle affiche le mode d'urgence. Appuyez sur Entrée, remettez `nofail` avec `nano /etc/fstab`, puis :

```
root@serveur:~# systemctl daemon-reload
root@serveur:~# systemctl default
```

L'invite de connexion apparaît.

11. Éteignez, rattachez le disque, démarrez. Les deux systèmes de fichiers sont de nouveau montés, et votre fichier est intact.

Vérifier

- `findmnt --verify` ne signale aucune erreur.

- `df -hT /srv/donnees /srv/archives` montre les deux montages après un redémarrage.
- Vous savez ce qu'affiche un serveur dont le `fstab` est faux, et comment le remettre en route.

Exercices

Exercice 1 — Inventaire

1. Combien de disques votre machine compte-t-elle ? Lequel porte le système ?
2. Sa table de partitions est-elle de type MBR ou GPT ?
3. Quel est l'UUID de la partition montée sur `/boot` ?
4. Avec quelles options la racine est-elle montée ?
5. Combien d'inodes la racine peut-elle contenir, et combien sont utilisés ?

Exercice 2 — Choisir

Quel système de fichiers proposez-vous dans chaque cas ? Justifiez en une phrase.

1. La partition de données d'un serveur web.
2. Une clé USB qui passera d'Ubuntu à Windows et à macOS, avec des vidéos de 6 Go.
3. Un volume de 40 To destiné à des archives de sauvegarde de 100 Go chacune.
4. Un poste de développeur qui veut pouvoir revenir à l'état d'avant une mise à jour.

Exercice 3 — Lire une ligne de fstab

```
/dev/sdb1   /mnt/sauvegardes   ext4   defaults   1   2
```

1. Que signifie chacun des six champs ?
2. Qu'est-ce qui rend cette ligne fragile ? Récrivez-la.
3. Ce disque est externe et parfois débranché. Que se passe-t-il au démarrage quand il manque ? Corrigez.
4. On ne veut pas qu'un programme puisse être exécuté depuis ce disque. Quelle option ajouter ?

Exercice 4 — Manipuler

Sur le disque de l'atelier, après avoir démonté et effacé ce qui s'y trouve :

1. Créez avec `fdisk` une table GPT et une partition de 2 Go.
2. Formatez-la en ext4, avec l'étiquette `essai`.
3. Montez-la sur `/mnt/essai` en la désignant par son étiquette, et non par son nom de périphérique.
4. Remontez-la en lecture seule sans la démonter, et vérifiez.
5. Ramenez la réserve de `root` à 1 %. Combien d'espace disponible avez-vous gagné ?

Exercice 5 — Échange et diagnostic

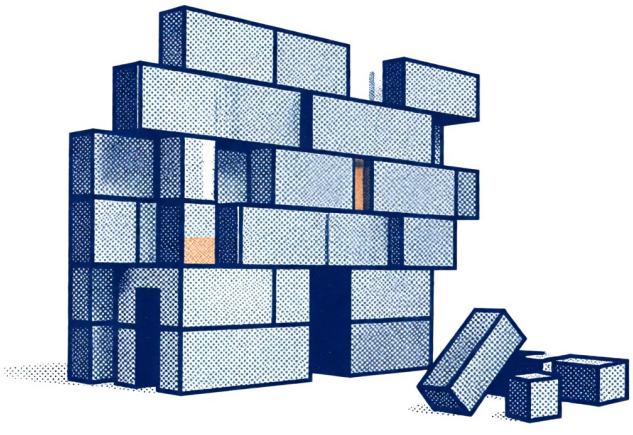
1. Quelle est la taille de l'espace d'échange de votre machine ? Quelle part est utilisée ?
2. Ajoutez un fichier d'échange de 1 Go, permanent. Vérifiez après un redémarrage, puis retirez-le proprement.
3. Tentez un `umount` de `/boot`. Que répond le système ? `fuser -vm /boot` ne désigne aucun processus : trouvez l'explication avec `findmnt -R /boot`.
4. Un utilisateur ne peut plus créer de fichier alors que `df -h` annonce 40 % d'espace libre. Quelle commande lancez-vous ?

Pour aller plus loin

Sur la machine. `man fstab` détaille les six champs, `man mount` toutes les options, communes ou propres à chaque système de fichiers. `man systemd.mount` explique comment systemd traduit `fstab` en unités, et décrit les options `x-systemd.…`, dont `x-systemd.automount`, qui ne monte un système de fichiers qu'au premier accès.

Outils. `sudo tune2fs -l /dev/sdb1` affiche tous les réglages d'un système de fichiers ext4. `sudo du -xh --max-depth=1 /` montre ce qui remplit une partition. `iostat -x 2`, du paquet `sysstat`, mesure l'activité des disques en direct.

La suite. Les partitions ont un défaut : leur taille est figée. Le chapitre 15 introduit LVM, qui permet d'agrandir un volume sans l'arrêter, et le RAID, qui survit à la panne d'un disque.



CHAPITRE 15

LVM, RAID et chiffrement

Une partition a une taille fixe, un disque finit par tomber en panne, et un disque volé se lit sur n'importe quelle machine. Trois couches logicielles répondent à ces trois faiblesses. Elles s'empilent.

DANS CE CHAPITRE

- Expliquer ce que sont un volume physique, un groupe de volumes et un volume logique.
- Agrandir la racine d'un serveur Ubuntu sans l'arrêter.
- Créer un groupe de volumes, y découper des volumes, les agrandir.
- Ajouter un disque à un groupe, en retirer un.
- Prendre un instantané avant une opération risquée, et y revenir.
- Monter un miroir RAID 1, simuler la panne d'un disque et le remplacer.
- Chiffrer un disque de données avec LUKS.
- Empiler RAID, chiffrement et LVM dans le bon ordre.

> **AVANT DE COMMENCER**
>
> Le chapitre 14. Ajoutez à votre machine virtuelle trois disques de 5 Go, qui apparaîtront sous les noms `/dev/sdc`, `/dev/sdd` et `/dev/sde`. Vérifiez-le avec `lsblk` avant chaque manipulation : tout ce chapitre efface des disques. Prenez un instantané de la machine avant de commencer.

15.1 Pourquoi LVM

Au chapitre 14, nous avons découpé un disque en partitions. Ce découpage est rigide. Une partition ne s'agrandit que s'il reste de la place juste derrière elle, sur le même disque. Quand `/srv/donnees` est plein et que le disque l'est aussi, il ne reste qu'à tout déménager.

LVM (*Logical Volume Manager*) glisse une couche entre les disques et les systèmes de fichiers. Les disques sont mis en commun dans une réserve, et l'on y taille des volumes à la demande. Un volume s'agrandit en une commande, pendant que le système tourne. La réserve grossit quand on lui ajoute un disque.

L'installateur d'Ubuntu Server met LVM en place d'office. Vous l'utilisez déjà.

15.2 Trois étages

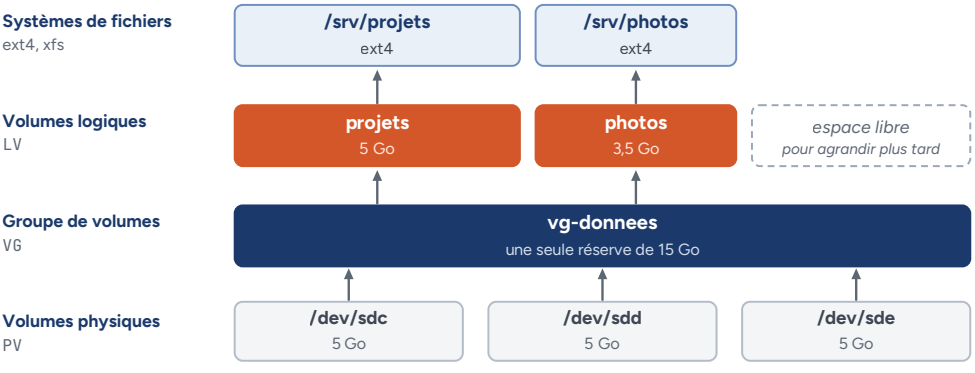


Figure 15.1 — Les trois étages de LVM, sous les systèmes de fichiers.

Un **volume physique** (PV, *physical volume*) est un disque ou une partition confié à LVM.

Un **groupe de volumes** (VG, *volume group*) réunit un ou plusieurs volumes physiques en une seule réserve d'espace.

Un **volume logique** (LV, *logical volume*) est taillé dans cette réserve. On le formate et on le monte comme une partition.

Chaque étage a ses commandes, construites sur le même modèle :

	Volume physique	Groupe de volumes	Volume logique
Lister	`pvs`	`vgs`	`lvs`
Détailler	`pvdisplay`	`vgdisplay`	`lvdisplay`
Créer	`pvcreate`	`vgcreate`	`lvcreate`
Agrandir		`vgextend`	`lvextend`
Réduire		`vgreduce`	`lvreduce`
Supprimer	`pvremove`	`vgremove`	`lvremove`

15.3 LVM est déjà là : agrandir la racine

Regardons ce qu'a fait l'installateur :

```
gary@serveur:~$ sudo pvs
  PV         VG        Fmt  Attr PSize   PFree
  /dev/sda3  ubuntu-vg lvm2 a--  <22.00g <11.00g
gary@serveur:~$ sudo vgs
  VG        #PV #LV #SN Attr   VSize   VFree
  ubuntu-vg   1   1   0 wz--n- <22.00g <11.00g
gary@serveur:~$ sudo lvs
  LV        VG        Attr       LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  ubuntu-lv ubuntu-vg -wi-ao---- 11.00g
gary@serveur:~$ df -h /
Filesystem                         Size  Used Avail Use% Mounted on
/dev/mapper/ubuntu--vg-ubuntu--lv   11G  3.9G  6.4G  38% /
```

La partition `sda3` est un volume physique de 22 Go. Elle forme à elle seule le groupe `ubuntu-vg`. Dans ce groupe, un seul volume logique, `ubuntu-lv`, porte la racine.

Un détail saute aux yeux : la racine ne fait que 11 Go, et la colonne `VFree` annonce 11 Go inutilisés. Ce n'est pas un oubli. L'installateur ne donne à la racine que la moitié d'un groupe de 20 à 200 Go, et garde le reste pour plus tard : un instantané, un second volume. Beaucoup d'administrateurs l'ignorent, et s'étonnent qu'un serveur doté d'un disque de 100 Go soit plein à 50.

Donnons 5 Go de plus à la racine :

```
gary@serveur:~$ sudo lvextend -r -L +5G /dev/ubuntu-vg/ubuntu-lv
  Size of logical volume ubuntu-vg/ubuntu-lv changed from 11.00 GiB (2816 extents) to 16.00
GiB (4096 extents).
  Logical volume ubuntu-vg/ubuntu-lv successfully resized.
resize2fs 1.47.0 (5-Feb-2023)
Filesystem at /dev/mapper/ubuntu--vg-ubuntu--lv is mounted on /; on-line resizing required
old_desc_blocks = 2, new_desc_blocks = 2
The filesystem on /dev/mapper/ubuntu--vg-ubuntu--lv is now 4194304 (4k) blocks long.
gary@serveur:~$ df -h /
Filesystem                         Size  Used Avail Use% Mounted on
/dev/mapper/ubuntu--vg-ubuntu--lv   16G  3.9G   12G  26% /
```

La racine est passée de 11 à 16 Go sans redémarrage, sans démontage, pendant que le système tournait dessus.

> **À RETENIR · Toujours avec -r**
>
> `lvextend` agrandit le volume. Le système de fichiers qu'il contient, lui, garde sa taille. L'option `-r` l'agrandit dans la foulée. Sans elle, `df` n'affiche aucun changement, et il faut lancer soi-même `resize2fs` pour ext4 ou `xfs_growfs` pour xfs.

La taille s'exprime de plusieurs façons : `-L +5G` ajoute 5 Go, `-L 16G` fixe la taille à 16 Go, `-l +100%FREE` prend tout l'espace libre du groupe. Résistez à cette dernière. Un groupe sans espace libre ne permet plus aucun instantané.

15.4 Construire un groupe de volumes

Créons notre propre réserve avec deux des disques neufs.

```
gary@serveur:~$ sudo pvcreate /dev/sdc /dev/sdd
  Physical volume "/dev/sdc" successfully created.
  Physical volume "/dev/sdd" successfully created.
gary@serveur:~$ sudo vgcreate vg-donnees /dev/sdc /dev/sdd
  Volume group "vg-donnees" successfully created
gary@serveur:~$ sudo lvcreate -n projets -L 3G vg-donnees
  Logical volume "projets" created.
gary@serveur:~$ sudo lvcreate -n photos -l 50%FREE vg-donnees
  Logical volume "photos" created.
gary@serveur:~$ sudo vgs
  VG         #PV #LV #SN Attr   VSize   VFree
  ubuntu-vg    1   1   0 wz--n- <22.00g <6.00g
  vg-donnees   2   2   0 wz--n-   9.99g <3.50g
```

Trois commandes, une par étage. On confie ici les disques entiers à LVM, sans table de partitions : c'est permis, et courant pour des disques de données.

Un volume logique se désigne par le chemin `/dev/groupe/volume`. Il se formate et se monte comme une partition :

```
gary@serveur:~$ sudo mkfs.ext4 -q /dev/vg-donnees/projets
gary@serveur:~$ sudo mkdir /srv/projets
gary@serveur:~$ sudo mount /dev/vg-donnees/projets /srv/projets
gary@serveur:~$ df -h /srv/projets
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/vg--donnees-projets  2.9G   24K  2.8G   1% /srv/projets
```

`df` affiche le même volume sous un second nom, `/dev/mapper/vg--donnees-projets`, où les tirets du groupe sont doublés. Les deux chemins sont des liens vers le même périphérique. Dans `fstab`, utilisez la forme `/dev/vg-donnees/projets` : ce nom ne dépend pas de l'ordre des disques.

`lsblk` montre où se trouvent physiquement les données :

```
gary@serveur:~$ lsblk /dev/sdc /dev/sdd
NAME                  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sdc                     8:32   0    5G  0 disk
└─vg--donnees-projets 252:1    0    3G  0 lvm  /srv/projets
sdd                     8:48   0    5G  0 disk
└─vg--donnees-photos  252:2    0  3.5G  0 lvm
```

NOTE · Les extents

LVM découpe la réserve en morceaux de 4 Mio, les *extents*. Un volume logique est une liste d'extents, qui peuvent se trouver sur n'importe quel disque du groupe. C'est ce qui lui permet de s'étendre d'un disque sur l'autre, et à LVM de déplacer des données d'un disque à un autre pendant qu'on s'en sert.

15.5 Faire évoluer le groupe

Ajouter un disque

Le groupe est trop petit ? Donnons-lui le troisième disque.

```
gary@serveur:~$ sudo pvcreate /dev/sde
  Physical volume "/dev/sde" successfully created.
gary@serveur:~$ sudo vgextend vg-donnees /dev/sde
  Volume group "vg-donnees" successfully extended
```

```
gary@serveur:~$ sudo vgs vg-donnees
  VG         #PV #LV #SN Attr   VSize   VFree
  vg-donnees   3   2   0 wz--n- <14.99g 6.49g
```

L'espace est disponible tout de suite, et les systèmes de fichiers n'ont pas été démontés un instant. Profitons-en pour agrandir `projets` :

```
gary@serveur:~$ sudo lvextend -r -L +2G /dev/vg-donnees/projets
[…]
gary@serveur:~$ df -h /srv/projets
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/vg--donnees-projets  4.9G   24K  4.7G   1% /srv/projets
```

ATTENTION · Plusieurs disques, plusieurs risques

Un volume logique étalé sur trois disques est perdu dès que l'un des trois tombe en panne. LVM additionne les capacités, pas la sécurité. Pour survivre à la panne d'un disque, il faut du RAID sous LVM : c'est l'objet de la seconde moitié du chapitre.

Retirer un disque

L'opération inverse est possible tant que les autres disques ont assez de place libre. `pvmove` vide un volume physique en déplaçant ses extents ailleurs. Les volumes restent montés et utilisables pendant ce temps.

```
gary@serveur:~$ sudo pvmove /dev/sdc
  /dev/sdc: Moved: 1.49%
  /dev/sdc: Moved: 100.00%
gary@serveur:~$ sudo vgreduce vg-donnees /dev/sdc
  Removed "/dev/sdc" from volume group "vg-donnees"
gary@serveur:~$ sudo pvremove /dev/sdc
  Labels on physical volume "/dev/sdc" successfully wiped.
```

C'est la manière de remplacer un vieux disque par un neuf sans interrompre le service : on ajoute le neuf, on vide l'ancien, on le retire.

Réduire un volume

Réduire est plus délicat qu'agrandir. Il faut d'abord rétrécir le système de fichiers, puis le volume, et dans cet ordre. Dans l'autre, la fin du système de fichiers est coupée. xfs ne sait pas rétrécir du tout. ext4 le peut, démonté.

Là encore, `-r` fait les deux opérations dans le bon ordre :

```
gary@serveur:~$ sudo umount /srv/projets
gary@serveur:~$ sudo lvreduce -r -L 2G /dev/vg-donnees/projets
fsck from util-linux 2.39.3
/dev/mapper/vg--donnees-projets: 12/327680 files (0.0% non-contiguous), 40031/1310720 blocks
resize2fs 1.47.0 (5-Feb-2023)
Resizing the filesystem on /dev/mapper/vg--donnees-projets to 524288 (4k) blocks.
The filesystem on /dev/mapper/vg--donnees-projets is now 524288 (4k) blocks long.

  Size of logical volume vg-donnees/projets changed from 5.00 GiB (1280 extents) to 2.00 GiB
(512 extents).
  Logical volume vg-donnees/projets successfully resized.
```

Remontez ensuite le volume avec `sudo mount /dev/vg-donnees/projets /srv/projets`.

Sauvegardez avant toute réduction. Et retenez la leçon pratique : créez les volumes petits, laissez de l'espace libre dans le groupe, et agrandissez au besoin. Agrandir est sans danger et se fait à chaud.

15.6 Les instantanés

Un **instantané** (en anglais *snapshot*) fige l'état d'un volume à un moment donné. Il ne copie rien sur le coup. À partir de sa création, chaque bloc que l'on modifie dans le volume d'origine est d'abord recopié dans l'instantané. Celui-ci ne grossit donc qu'au rythme des modifications.

Prenons un instantané de 500 Mo avant de toucher à un fichier :

```
gary@serveur:~$ echo "version 1" | sudo tee /srv/projets/rapport.txt
version 1
gary@serveur:~$ sudo lvcreate -s -n projets-avant -L 500M /dev/vg-donnees/projets
  Logical volume "projets-avant" created.
gary@serveur:~$ echo "version 2, ratée" | sudo tee /srv/projets/rapport.txt
version 2, ratée
```

L'instantané est un volume comme un autre. Monté en lecture seule, il montre le passé :

```
gary@serveur:~$ sudo mkdir /mnt/avant
gary@serveur:~$ sudo mount -o ro /dev/vg-donnees/projets-avant /mnt/avant
gary@serveur:~$ cat /mnt/avant/rapport.txt
version 1
gary@serveur:~$ cat /srv/projets/rapport.txt
version 2, ratée
```

Deux usages en découlent.

Le premier est la sauvegarde. Copier un volume pendant qu'on y écrit donne une copie incohérente. On prend donc un instantané, on sauvegarde l'instantané, figé, puis on le supprime avec `lvremove`. Le chapitre 16 s'en servira.

Le second est le retour en arrière. Avant une mise à jour risquée, un instantané. Si elle tourne mal, on fusionne l'instantané dans le volume d'origine, qui retrouve son état d'avant :

```
gary@serveur:~$ sudo umount /mnt/avant
gary@serveur:~$ sudo umount /srv/projets
gary@serveur:~$ sudo lvconvert --merge /dev/vg-donnees/projets-avant
  Merging of volume vg-donnees/projets-avant started.
  vg-donnees/projets: Merged: 100.00%
gary@serveur:~$ sudo mount /dev/vg-donnees/projets /srv/projets
gary@serveur:~$ cat /srv/projets/rapport.txt
version 1
```

La fusion consomme l'instantané : il n'existe plus ensuite.

> **ATTENTION · Un instantané se remplit**
>
> Les 500 Mo de l'exemple sont la quantité de modifications que l'instantané peut absorber. S'il se remplit, il devient inutilisable. La colonne `Data%` de `lvs` indique son remplissage. Un instantané n'est pas fait pour durer : quelques heures, le temps d'une sauvegarde ou d'une mise à jour.

15.7 Le RAID logiciel

> **REPÈRE HISTORIQUE · Des disques bon marché**
>
> Le mot RAID apparaît en 1987 à l'université de Berkeley, sous la plume de David Patterson, Garth Gibson et Randy Katz. Leur article proposait de remplacer les gros disques coûteux des grands systèmes par des grappes de petits disques de micro-ordinateur : *Redundant Arrays of Inexpensive Disks*. Les fabricants ont ensuite préféré lire *Independent* à la place d'*Inexpensive*.

Le **RAID** assemble plusieurs disques en une *grappe* qui se présente comme un seul. Selon le niveau choisi, on y gagne de la vitesse, de la sécurité, ou les deux.

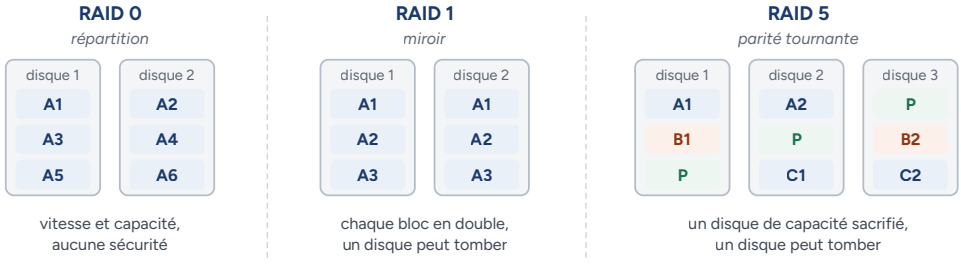


Figure 15.2 — Trois façons de répartir les données sur une grappe.

Niveau	Disques	Capacité utile	Pannes tolérées	Usage
RAID 0	2 ou plus	Tous les disques	Aucune	Données temporaires, jamais rien d'important
RAID 1	2	Un seul disque	1	Système, petits serveurs
RAID 5	3 ou plus	Tous sauf un	1	Stockage de volume moyen
RAID 6	4 ou plus	Tous sauf deux	2	Gros stockage
RAID 10	4 ou plus	La moitié	1 par miroir	Bases de données

Sous Linux, le RAID logiciel est assuré par le noyau et piloté par la commande `mdadm`.

> **À RETENIR · Le RAID n'est pas une sauvegarde**
>
> Un miroir recopie fidèlement, et instantanément, un fichier effacé par erreur, un fichier chiffré par un rançongiciel, une base corrompue par une mise à jour. Le RAID protège contre une seule chose : la panne d'un disque. Tout le reste relève de la sauvegarde, au chapitre 16.

Créer un miroir

Pour la suite du chapitre, rendez leur liberté aux trois disques : démontez `/srv/projets`, supprimez le groupe avec `sudo vgremove vg-donnees`, puis effacez les signatures avec `sudo wipefs -a /dev/sdc /dev/sdd /dev/sde`.

Assemblons `/dev/sdc` et `/dev/sdd` en miroir :

```
gary@serveur:~$ sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdc /dev/sdd
mdadm: Note: this array has metadata at the start and
    may not be suitable as a boot device.  If you plan to
    store '/boot' on this device please ensure that
    your boot-loader understands md/v1.x metadata, or use
    --metadata=0.90
Continue creating array? y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

La grappe s'appelle `/dev/md0` et s'utilise aussitôt. En arrière-plan, le noyau recopie le premier disque sur le second. Le fichier `/proc/mdstat` montre l'avancement :

```
gary@serveur:~$ cat /proc/mdstat
Personalities : [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdd[1] sdc[0]
      5237760 blocks super 1.2 [2/2] [UU]
      [>....................]  resync =  3.8% (201984/5237760) finish=0.4min
speed=201984K/sec

unused devices: <none>
```

Tout se lit dans `[2/2] [UU]` : deux disques attendus, deux présents, tous deux en bon état (*up*). Un disque défaillant serait signalé par un tiret bas : `[2/1] [_U]`.

Enregistrez ensuite la grappe dans la configuration de `mdadm`, puis reconstruisez l'initramfs. C'est la garantie qu'elle sera assemblée au démarrage sous le même nom :

```
gary@serveur:~$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
ARRAY /dev/md0 metadata=1.2 UUID=1f19aaca:87403589:fb7f5b24:df7293dc
gary@serveur:~$ sudo update-initramfs -u
update-initramfs: Generating /boot/initrd.img-6.8.0-142-generic
```

`/dev/md0` se comporte désormais comme un disque. On peut y créer un système de fichiers, ou mieux, en faire un volume physique LVM.

Surveiller

`sudo mdadm --detail /dev/md0` donne l'état complet de la grappe : son niveau, sa taille, l'état de chaque disque.

Un miroir dégradé fonctionne. C'est son mérite, et c'est le piège : rien ne vous avertit que la prochaine panne sera fatale. Ubuntu installe donc deux gardes avec le paquet `mdadm`. Le service `mdmonitor` surveille les grappes et envoie un courriel à l'adresse inscrite à la ligne `MAILADDR` de `mdadm.conf`. Encore faut-il que la machine

sache envoyer du courrier. Une minuterie, `mdcheck_start.timer`, relit toute la grappe le premier dimanche de chaque mois pour débusquer les secteurs devenus illisibles.

La panne d'un disque et son remplacement font l'objet de l'atelier.

15.8 Chiffrer un disque avec LUKS

Un disque non chiffré se lit sur n'importe quel ordinateur, quels que soient les droits des fichiers. Il suffit de le brancher ailleurs. **LUKS** chiffre tout un périphérique. Sans la phrase secrète, son contenu est indiscernable d'une suite d'octets au hasard.

Le chiffrement protège une machine éteinte : un portable volé, un disque retiré d'un serveur, un disque envoyé en réparation ou mis au rebut. Il ne protège pas d'un intrus connecté à une machine allumée, où les volumes sont ouverts.

Chiffrons le troisième disque :

```
gary@serveur:~$ sudo cryptsetup luksFormat /dev/sde

WARNING!
========
This will overwrite data on /dev/sde irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sde:
Verify passphrase:
```

Le disque chiffré ne se monte pas directement. On l'*ouvre* d'abord : `cryptsetup` demande la phrase secrète et crée un périphérique déchiffré dans `/dev/mapper/`, sous le nom de notre choix. C'est lui que l'on formate et que l'on monte.

```
gary@serveur:~$ sudo cryptsetup open /dev/sde coffre
Enter passphrase for /dev/sde:
gary@serveur:~$ lsblk /dev/sde
NAME     MAJ:MIN RM SIZE RO TYPE  MOUNTPOINTS
sde        8:64   0   5G  0 disk
└─coffre 252:3    0   5G  0 crypt
gary@serveur:~$ sudo mkfs.ext4 -q /dev/mapper/coffre
gary@serveur:~$ sudo mkdir /srv/coffre
gary@serveur:~$ sudo mount /dev/mapper/coffre /srv/coffre
```

Pour le refermer, on démonte puis on ferme :

```
gary@serveur:~$ sudo umount /srv/coffre
gary@serveur:~$ sudo cryptsetup close coffre
gary@serveur:~$ sudo mount /dev/sde /mnt
mount: /mnt: unknown filesystem type 'crypto_LUKS'.
```

Fermé, le disque ne livre plus rien.

> **DANGER · Pas de phrase secrète, pas de données**
>
> Il n'existe ni porte dérobée, ni procédure de récupération. Une phrase secrète oubliée, et les données sont perdues pour tout le monde, vous compris. Notez-la dans un gestionnaire de mots de passe ou sur un papier rangé en lieu sûr.

Plusieurs clés, et une copie de l'en-tête

Un volume LUKS accepte plusieurs clés, chacune dans un *emplacement* numéroté. On peut ainsi donner une phrase à chaque administrateur, ou ajouter un fichier de clé :

```
gary@serveur:~$ sudo dd if=/dev/urandom of=/root/coffre.cle bs=64 count=1
1+0 records in
1+0 records out
64 bytes copied, 9.887e-05 s, 647 kB/s
gary@serveur:~$ sudo chmod 400 /root/coffre.cle
gary@serveur:~$ sudo cryptsetup luksAddKey /dev/sde /root/coffre.cle
Enter any existing passphrase:
```

La clé qui chiffre réellement les données est rangée dans l'en-tête du disque, protégée par vos phrases secrètes. Si cet en-tête est endommagé, plus aucune phrase n'ouvre le volume. Sauvegardez-le, et gardez la copie ailleurs que sur le disque :

```
sudo cryptsetup luksHeaderBackup /dev/sde --header-backup-file /root/coffre-entete.img
```

Ouvrir au démarrage

Le fichier `/etc/crypttab` liste les volumes à ouvrir au démarrage. Il joue pour le chiffrement le rôle de `fstab` pour les montages, et passe avant lui.

```
gary@serveur:~$ sudo blkid /dev/sde
/dev/sde: UUID="7d2a8d4a-4168-44f0-8f56-15ba9eca9ead" TYPE="crypto_LUKS"
gary@serveur:~$ cat /etc/crypttab
# <target name> <source device>         <key file>      <options>
```

```
coffre UUID=7d2a8d4a-4168-44f0-8f56-15ba9eca9ead /root/coffre.cle luks,nofail
gary@serveur:~$ tail -n 1 /etc/fstab
/dev/mapper/coffre /srv/coffre ext4 defaults,nofail 0 2
```

Les quatre champs de `crypttab` : le nom à donner au volume ouvert, le disque chiffré, la clé, les options. Au démarrage suivant, le volume est ouvert et monté sans intervention.

> **SUR LE TERRAIN · La clé à côté de la serrure**
>
> Un fichier de clé rangé sur la même machine, sur un disque système non chiffré, ne protège que le disque de données pris isolément : celui qu'on retire, qu'on remplace ou qu'on jette. Contre le vol de la machine entière, il ne sert à rien. Deux solutions. Soit on écrit `none` à la place du fichier de clé, et la phrase secrète est demandée sur la console à chaque démarrage, ce qui suppose d'y avoir accès. Soit le système lui-même est chiffré, option à choisir à l'installation (chapitre 2), et le fichier de clé est alors à l'abri.

15.9 Empiler les couches

RAID, chiffrement et LVM se combinent. Chacun présente à l'étage du dessus un périphérique ordinaire.

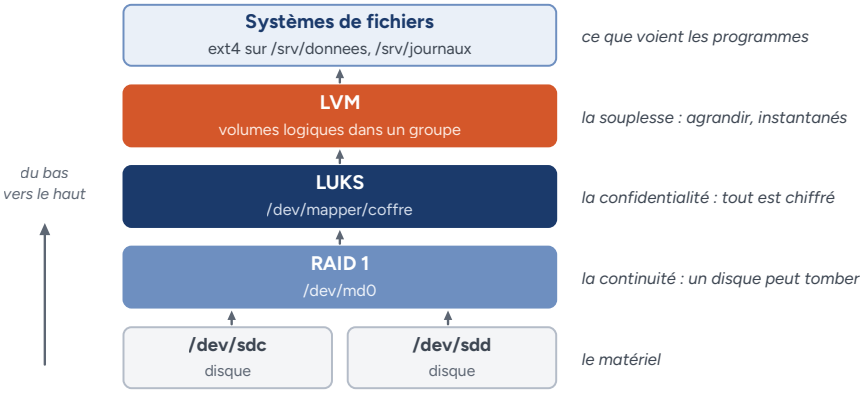


Figure 15.3 — L'ordre habituel des couches, du matériel aux fichiers.

L'ordre se justifie de bas en haut. Le RAID se place contre les disques : tout ce qui est au-dessus profite de la redondance. Le chiffrement vient ensuite : on chiffre une fois la grappe, et non chaque volume. LVM se place en haut, là où l'on a besoin de souplesse.

Tout n'est pas toujours nécessaire. Un serveur dans un centre de données protégé se contente souvent de RAID et de LVM. Un ordinateur portable n'a qu'un disque, et prend le chiffrement et LVM. L'atelier construit la pile la plus courante sur un serveur : un miroir, et LVM par-dessus.

En résumé

- LVM met les disques en commun dans un groupe de volumes, où l'on taille des volumes logiques.
- `pvs`, `vgs`, `lvs` montrent les trois étages.
- Sur un serveur fraîchement installé, la racine n'occupe que la moitié du groupe. `sudo lvextend -r -L +5G /dev/ubuntu-vg/ubuntu-lv` l'agrandit à chaud.
- Toujours `-r` avec `lvextend` et `lvreduce`.
- Gardez de l'espace libre dans chaque groupe.
- Un instantané fige un volume le temps d'une sauvegarde ou d'une opération risquée.
- Le RAID 1 survit à la panne d'un disque. `cat /proc/mdstat` : `[UU]` quand tout va bien.
- Le RAID ne remplace pas les sauvegardes.
- LUKS chiffre un disque entier. Sans phrase secrète, rien n'est récupérable. On sauvegarde l'en-tête.
- L'ordre des couches : disques, RAID, LUKS, LVM, système de fichiers.

Atelier — Un miroir qui survit à une panne

Objectif : monter un miroir de deux disques, y installer LVM et deux systèmes de fichiers, puis arracher un disque en fonctionnement, constater que rien n'est perdu, et reconstruire le miroir. Comptez une heure.

Préparer

1. Si vous avez créé la grappe `/dev/md0` en lisant le chapitre, passez directement à l'étape 5. Sinon, il vous faut `/dev/sdc` et `/dev/sdd`, vides. S'ils portent encore le groupe `vg-donnees`, démontez ce qui s'y trouve, supprimez le groupe et effacez les signatures :

```
sudo umount /srv/projets
sudo vgremove vg-donnees
sudo wipefs -a /dev/sdc /dev/sdd
```

Le miroir

2. Créez la grappe et répondez `y` à la question :

```
sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdc /dev/sdd
```

3. Suivez la synchronisation avec `cat /proc/mdstat` jusqu'à ce que la barre disparaisse et qu'il reste `[2/2] [UU]`.
4. Enregistrez la grappe :

```
sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
sudo update-initramfs -u
```

LVM par-dessus

5. Faites de la grappe un volume physique, créez un groupe et deux volumes, formatez-les :

```
gary@serveur:~$ sudo pvcreate /dev/md0
  Physical volume "/dev/md0" successfully created.
gary@serveur:~$ sudo vgcreate vg-miroir /dev/md0
  Volume group "vg-miroir" successfully created
gary@serveur:~$ sudo lvcreate -n donnees -L 2G vg-miroir
  Logical volume "donnees" created.
gary@serveur:~$ sudo lvcreate -n journaux -L 1G vg-miroir
  Logical volume "journaux" created.
gary@serveur:~$ sudo mkfs.ext4 -q /dev/vg-miroir/donnees
gary@serveur:~$ sudo mkfs.ext4 -q /dev/vg-miroir/journaux
```

6. Créez les points de montage, ajoutez deux lignes à `/etc/fstab`, puis vérifiez et montez comme au chapitre 14 :

```
/dev/vg-miroir/donnees /srv/miroir/donnees ext4 defaults,nofail 0 2
/dev/vg-miroir/journaux /srv/miroir/journaux ext4 defaults,nofail 0 2
```

```
gary@serveur:~$ sudo mkdir -p /srv/miroir/donnees /srv/miroir/journaux
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo mount -a
gary@serveur:~$ lsblk /dev/sdc /dev/sdd
NAME                    MAJ:MIN RM SIZE RO TYPE  MOUNTPOINTS
sdc                       8:32   0   5G  0 disk
```

```
└─md0                     9:0    0   5G  0 raid1
  ├─vg--miroir-donnees  252:1    0   2G  0 lvm   /srv/miroir/donnees
  └─vg--miroir-journaux 252:2    0   1G  0 lvm   /srv/miroir/journaux
sdd                       8:48   0   5G  0 disk
└─md0                     9:0    0   5G  0 raid1
  ├─vg--miroir-donnees  252:1    0   2G  0 lvm   /srv/miroir/donnees
  └─vg--miroir-journaux 252:2    0   1G  0 lvm   /srv/miroir/journaux
```

`lsblk` dessine toute la pile : deux disques, une grappe, deux volumes, deux montages.

Des fichiers témoins

7. Copiez quelques milliers de fichiers sur le volume et relevez leurs empreintes. Elles prouveront que rien n'a été altéré.

```
gary@serveur:~$ sudo cp -r /usr/share/doc /srv/miroir/donnees/
gary@serveur:~$ sudo sh -c 'cd /srv/miroir/donnees && find . -type f -exec
sha256sum {} + > /root/empreintes.txt'
gary@serveur:~$ sudo wc -l /root/empreintes.txt
3435 /root/empreintes.txt
```

La panne

8. Déclarez `/dev/sdc` en panne. C'est ce que ferait le noyau devant un disque qui ne répond plus.

```
gary@serveur:~$ sudo mdadm /dev/md0 --fail /dev/sdc
mdadm: set /dev/sdc faulty in /dev/md0
gary@serveur:~$ cat /proc/mdstat
Personalities : [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdd[1] sdc[0](F)
      5237760 blocks super 1.2 [2/1] [_U]

unused devices: <none>
```

Le `(F)` marque le disque fautif, et `[_U]` dit qu'un seul disque sur deux travaille encore.

9. Vérifiez les fichiers pendant que la grappe est dégradée :

```
gary@serveur:~$ sudo sh -c 'cd /srv/miroir/donnees && sha256sum --quiet -c /
root/empreintes.txt' && echo "tout est intact"
tout est intact
gary@serveur:~$ journalctl -k -n 2 -o cat
md/raid1:md0: Disk failure on sdc, disabling device.
md/raid1:md0: Operation continuing on 1 devices.
```

Les systèmes de fichiers n'ont rien remarqué. Le noyau, lui, a consigné l'événement.

Le remplacement

10. Sortez le disque de la grappe. Sur une vraie machine, on le remplacerait ici par un disque neuf. Dans la machine virtuelle, effacez-le pour faire comme s'il l'était, puis ajoutez-le :

```
gary@serveur:~$ sudo mdadm /dev/md0 --remove /dev/sdc
mdadm: hot removed /dev/sdc from /dev/md0
gary@serveur:~$ sudo wipefs -a /dev/sdc
/dev/sdc: 4 bytes were erased at offset 0x00001000 (linux_raid_member): fc 4e 2b a9
gary@serveur:~$ sudo mdadm /dev/md0 --add /dev/sdc
mdadm: added /dev/sdc
gary@serveur:~$ cat /proc/mdstat
Personalities : [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdc[2] sdd[1]
      5237760 blocks super 1.2 [2/1] [_U]
      [>....................]  recovery =  3.8% (201728/5237760) finish=0.4min
speed=201728K/sec

unused devices: <none>
```

La reconstruction démarre seule. Tant qu'elle n'est pas finie, la grappe reste vulnérable : sur des disques de plusieurs téraoctets, elle dure des heures.

11. Quand `/proc/mdstat` affiche de nouveau `[UU]`, relancez la vérification des empreintes. Elle doit répondre « tout est intact ».

Agrandir

12. Il reste de la place dans le groupe. Donnez 1 Go de plus au volume de données, à chaud :

```
gary@serveur:~$ sudo vgs vg-miroir
  VG        #PV #LV #SN Attr   VSize VFree
  vg-miroir   1   2   0 wz--n- 4.99g 1.99g
gary@serveur:~$ sudo lvextend -r -L +1G /dev/vg-miroir/donnees
[…]
gary@serveur:~$ df -h /srv/miroir/donnees
Filesystem                      Size  Used Avail Use% Mounted on
/dev/mapper/vg--miroir-donnees  2.9G   33M  2.7G   2% /srv/miroir/donnees
```

Vérifier

- Après `sudo reboot`, `cat /proc/mdstat` montre `md0` avec `[UU]`, et les deux volumes sont montés.
- Les empreintes sont toujours bonnes.

- Vous avez vu une grappe dégradée, et vous savez la reconnaître d'un coup d'œil.

Exercices

Exercice 1 — Lire une installation

Sur un serveur Ubuntu installé avec les choix par défaut :

1. Combien d'espace libre reste-t-il dans `ubuntu-vg` ?
2. Sur quelle partition se trouve ce groupe ?
3. Donnez à la racine la moitié de l'espace libre, sans redémarrer. Vérifiez avec `df`.
4. Pourquoi est-il prudent de ne pas tout donner ?

Exercice 2 — Choisir un niveau de RAID

Pour chaque besoin, quel niveau proposez-vous, avec combien de disques, et pour quelle capacité utile ?

1. Le disque système d'un serveur, avec deux disques de 500 Go.
2. Un stockage d'archives sur quatre disques de 4 To, qui doit survivre à la panne de deux disques.
3. Un espace de calcul temporaire, où seule la vitesse compte.
4. Un serveur de fichiers à trois disques de 2 To, avec le plus de capacité possible et la tolérance d'une panne.

Exercice 3 — Instantané

1. Créez un volume logique de 1 Go, formatez-le, montez-le et copiez-y `/etc`.
2. Prenez-en un instantané de 200 Mo.
3. Supprimez un dossier entier du volume. Observez la colonne `Data%` de `lvs`.
4. Récupérez un seul fichier en montant l'instantané en lecture seule.
5. Revenez à l'état initial par une fusion.

Exercice 4 — Diagnostic

1. `df -h` n'affiche aucun changement après un `lvextend`. Qu'avez-vous oublié, et comment rattraper ?
2. `/proc/mdstat` affiche `[U_]`. Que signifie cette ligne, et que faites-vous ?

3. `sudo vgs` annonce `VFree 0`. Quelles opérations ne sont plus possibles ?
4. Un collègue propose de se passer de sauvegardes « puisqu'on a un RAID 1 ». Que lui répondez-vous ?

Exercice 5 — Un disque chiffré

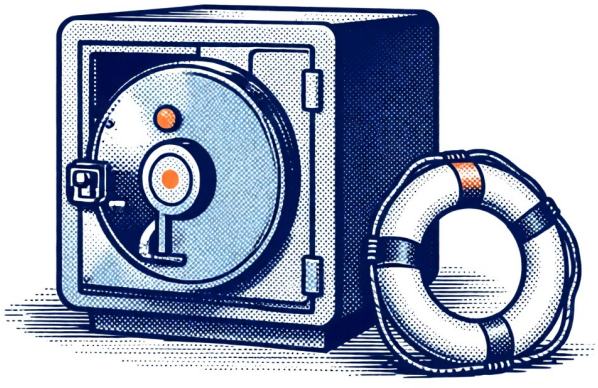
Sur `/dev/sde` :

1. Créez un volume LUKS avec une phrase secrète.
2. Ajoutez une seconde phrase, puis vérifiez avec `luksDump` que deux emplacements sont occupés.
3. Ouvrez le volume avec la seconde phrase, formatez-le, écrivez-y un fichier, refermez-le.
4. Supprimez la première phrase avec `luksRemoveKey`. Le volume s'ouvre-t-il encore ?
5. Sauvegardez l'en-tête. Où rangeriez-vous cette copie ?

Pour aller plus loin

Sur la machine. `man lvm` présente l'ensemble des commandes. `man lvmthin` décrit les volumes à allocation fine, qui rendent les instantanés bien plus souples. `man mdadm` et `man 4 md` détaillent chaque niveau de RAID. `man crypttab` liste les options d'ouverture au démarrage.

Prolongements. btrfs et zfs réunissent dans un seul outil ce que ce chapitre empile : plusieurs disques, la redondance, les instantanés. `systemd-cryptenroll` associe un volume LUKS à la puce TPM de la machine ou à une clé de sécurité FIDO2, pour une ouverture sans phrase à taper. Le chapitre 31 montrera comment retrouver ces couches depuis un système de secours : assembler une grappe, activer un groupe de volumes, ouvrir un volume chiffré.



CHAPITRE 16

Sauvegarde et restauration

Une sauvegarde qu'on n'a jamais restaurée n'est qu'un espoir. Ce chapitre construit une sauvegarde chiffrée, automatique et surveillée, puis vérifie qu'on sait en ressortir les fichiers.

DANS CE CHAPITRE

- Appliquer la règle des trois copies.
- Décider de ce qu'il faut sauvegarder, et de ce qu'il vaut mieux laisser.
- Copier et synchroniser avec `rsync`, sur place et par SSH.
- Archiver avec `tar` et extraire un fichier d'une archive.
- Sauvegarder avec restic : instantanés, chiffrement, restauration, tri.
- Figer un volume le temps de la sauvegarde.
- Automatiser la sauvegarde et être prévenu quand elle échoue.
- Tester une restauration.

> **AVANT DE COMMENCER**
>
> Les chapitres 13 à 15. Les exemples se déroulent sur le serveur des chapitres précédents, avec `/srv/donnees` et `/srv/archives` (chapitre 14) et le groupe de volumes `vg-miroir` (chapitre 15). `/srv/archives`, qui se trouve sur un autre disque que le système, reçoit les sauvegardes.

16.1 Trois copies, deux supports, un autre lieu

Un disque tombe en panne sans prévenir. Un `rm` part trop vite. Une mise à jour corrompt une base. Un rançongiciel chiffre tout ce qu'il atteint. Un cambriolage, un incendie, une inondation emportent la machine et ce qui se trouve à côté. Contre tout cela, la seule parade est d'avoir ailleurs une copie des données.

La règle dite « 3-2-1 » dit combien de copies, et où.

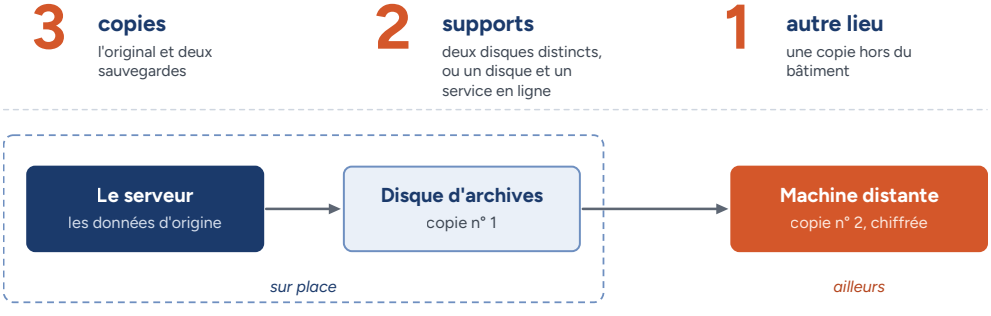


Figure 16.1 — La règle 3-2-1.

Trois copies : l'original et deux sauvegardes. Avec une seule sauvegarde, vous n'avez plus de filet le jour où vous vous en servez.

Deux supports : pas deux dossiers du même disque, ni deux disques du même boîtier. Ce qui détruit l'un ne doit pas pouvoir détruire l'autre.

Un autre lieu : une copie hors du bâtiment, chez un hébergeur, dans un autre bureau, chez un proche. C'est elle qui reste après un sinistre.

> **ATTENTION · Ce qui ressemble à une sauvegarde**
>
> Un miroir RAID recopie les erreurs à l'instant où elles se produisent. Un instantané LVM vit sur le même disque que l'original. Un dossier synchronisé avec un service en ligne propage les suppressions à toutes les copies. Aucun des trois ne permet de revenir à l'état de mardi dernier. Une sauvegarde, c'est une copie datée, rangée ailleurs, que les événements du jour n'atteignent pas.

16.2 Que sauvegarder ?

Tout sauvegarder coûte cher et ralentit les restaurations. La question utile est : qu'est-ce que je ne saurais pas refaire ?

À sauvegarder	Pourquoi
`/home`	Les données des utilisateurs
`/etc`	Toute la configuration du système
`/srv`, `/var/www`	Les données servies par la machine
`/root`	Les scripts et les clés de l'administrateur
`/var/lib` (en partie)	Les données des services : bases de données, conteneurs
`/usr/local`, `/opt`	Ce que vous avez installé hors des paquets

À laisser	Pourquoi
`/proc`, `/sys`, `/dev`, `/run`	Fabriqués par le noyau à chaque démarrage
`/tmp`, `/var/tmp`, `/var/cache`, `~/.cache`	Temporaires ou reconstruits tout seuls
`/usr`, `/bin`, `/lib`	Réinstallés par les paquets

Le système lui-même se réinstalle en vingt minutes. Ce qu'il faut garder, c'est la liste de ce qu'on y avait ajouté :

```
gary@serveur:~$ apt-mark showmanual > ~/paquets.txt
gary@serveur:~$ wc -l < ~/paquets.txt
42
```

Quarante-deux noms de paquets suffisent à reconstruire les 664 paquets de cette machine. Le fichier part avec le reste dans la sauvegarde.

> **DANGER · Les bases de données**
>
> Une base de données écrit dans ses fichiers en permanence. Copiés pendant qu'elle tourne, ils forment un ensemble incohérent, et la base restaurée refuse de démarrer. On sauvegarde une base en lui demandant d'exporter son contenu, avec `pg_dump` ou `mariadb-dump`, et c'est le fichier exporté que l'on sauvegarde. Le chapitre 24 y revient.

16.3 Complète, incrémentale, différentielle

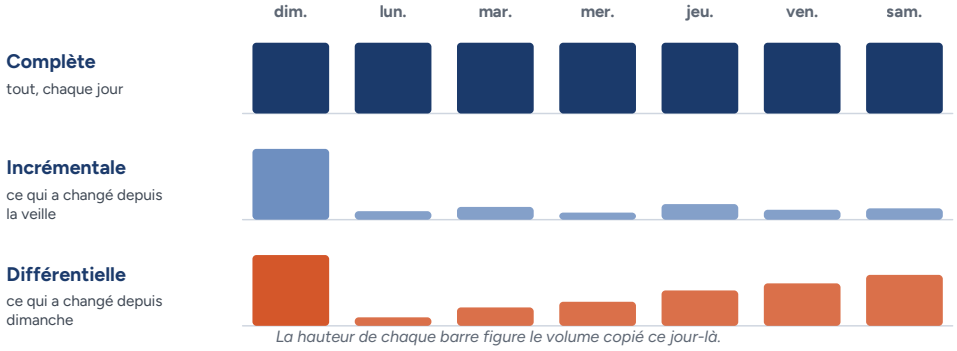


Figure 16.2 — Trois façons de sauvegarder la même semaine.

Type	Ce qui est copié	Pour restaurer
Complète	Tout, à chaque fois	La dernière sauvegarde suffit
Incrémentale	Ce qui a changé depuis la sauvegarde précédente	La complète, puis toutes les incrémentales dans l'ordre
Différentielle	Ce qui a changé depuis la dernière complète	La complète et la dernière différentielle

La complète est simple et volumineuse. L'incrémentale est légère, mais sa restauration dépend de chaque maillon de la chaîne : qu'un seul soit abîmé, et tout ce qui suit est perdu.

Les outils récents échappent à ce dilemme. Ils découpent les fichiers en blocs et ne rangent chaque bloc qu'une fois dans leur dépôt. Chaque sauvegarde se présente comme complète, et ne coûte que la place des blocs nouveaux. C'est le principe de restic, que nous utiliserons plus loin. Avant lui, deux outils que tout administrateur doit avoir en main.

16.4 rsync : copier ce qui a changé

`rsync` copie une arborescence vers une autre, en ne transférant que les différences. Il est installé d'origine sur Ubuntu Server. Lancé une seconde fois sur les mêmes dossiers, il n'a presque rien à faire :

```
gary@serveur:~$ rsync -avh ~/travail/ /srv/archives/copie/ | tail -n 2
sent 77.06K bytes  received 342 bytes  154.79K bytes/sec
total size is 75.94K  speedup is 0.98
gary@serveur:~$ rsync -avh ~/travail/ /srv/archives/copie/
sending incremental file list

sent 521 bytes  received 14 bytes  1.07K bytes/sec
total size is 75.94K  speedup is 141.93
```

Option	Effet
`-a`	Mode archive : sous-dossiers, droits, propriétaires, dates, liens
`-v`, `-h`	Lister les fichiers copiés, afficher des tailles lisibles
`-n`	Simuler : montrer ce qui serait fait, sans rien faire
`--delete`	Supprimer de la destination ce qui n'existe plus à la source
`--exclude='*.tmp'`	Écarter des fichiers
`-z`	Compresser pendant le transfert, utile sur une liaison lente

ATTENTION · La barre oblique finale

Avec une barre à la fin de la source, `rsync` copie le contenu du dossier. Sans elle, il copie le dossier lui-même :

```
gary@serveur:~$ rsync -a ~/travail /srv/archives/essai1
gary@serveur:~$ rsync -a ~/travail/ /srv/archives/essai2
gary@serveur:~$ ls /srv/archives/essai1 /srv/archives/essai2
/srv/archives/essai1:
travail

/srv/archives/essai2:
bash  notes
```

Prenez l'habitude de toujours écrire la barre finale, à la source comme à la destination.

`--delete` rend la copie fidèle, et dangereuse : une erreur sur la source vide la destination. On le précède toujours d'un essai avec `-n` :

```
gary@serveur:~$ rsync -avhn --delete ~/travail/ /srv/archives/copie/
sending incremental file list
deleting notes/liste.txt
notes/
notes/mardi.txt

sent 539 bytes  received 46 bytes  1.17K bytes/sec
total size is 75.95K  speedup is 129.83 (DRY RUN)
```

Un fichier serait supprimé, un autre copié. Si c'est bien ce que vous voulez, relancez sans le `n`.

Vers une autre machine

`rsync` passe par SSH dès que la destination est précédée d'un nom de machine. La syntaxe est celle de `scp` :

```
gary@serveur:~$ rsync -avh ~/travail/notes/ gary@localhost:/srv/archives/distant/
sending incremental file list
created directory /srv/archives/distant
./
lundi.txt
mardi.txt

sent 241 bytes  received 101 bytes  228.00 bytes/sec
total size is 44  speedup is 0.13
```

L'exemple se connecte à la machine elle-même pour l'essai. Remplacez `localhost` par le nom d'un autre serveur, et vous tenez la copie hors site de la règle 3-2-1. Le chapitre 19 expliquera comment ouvrir cette connexion par clé, sans mot de passe.

Une copie par jour, sans payer la place

Une copie synchronisée ne garde qu'un seul état : celui de la dernière exécution. Pour remonter dans le temps, il faut une copie par jour. L'option `--link-dest` rend cela abordable. Elle compare avec la copie de la veille, et remplace chaque fichier inchangé par un lien physique (chapitre 4) vers l'exemplaire déjà présent.

```
gary@serveur:~$ mkdir /srv/archives/jours
gary@serveur:~$ rsync -a ~/travail/ /srv/archives/jours/lundi/
gary@serveur:~$ echo "ajout du soir" >> ~/travail/notes/mardi.txt
gary@serveur:~$ rsync -a --link-dest=/srv/archives/jours/lundi \
    ~/travail/ /srv/archives/jours/mardi/
```

```
gary@serveur:~$ du -sh /srv/archives/jours/lundi /srv/archives/jours/mardi
108K    /srv/archives/jours/lundi
4.0K    /srv/archives/jours/mardi
```

Les deux dossiers contiennent tous les fichiers. Le second n'occupe pourtant que 4 Ko : seul le fichier modifié y a été copié. Les autres sont partagés, comme le prouvent leur numéro d'inode identique et leur compteur de liens à 2 :

```
gary@serveur:~$ ls -li /srv/archives/jours/*/notes/lundi.txt
4325572 -rw-rw-r-- 2 gary gary 22 Oct  4 22:04 /srv/archives/jours/lundi/notes/lundi.txt
4325572 -rw-rw-r-- 2 gary gary 22 Oct  4 22:04 /srv/archives/jours/mardi/notes/lundi.txt
```

On peut supprimer le dossier d'un jour sans abîmer les autres : un fichier ne disparaît que lorsque son dernier lien est effacé.

`rsync` crée le dossier de destination, mais pas ses parents. Sans le `mkdir` de la première ligne, il aurait répondu `mkdir "/srv/archives/jours/lundi" failed: No such file or directory`.

16.5 tar : tout dans un fichier

`tar` rassemble une arborescence dans un seul fichier, l'archive, que l'on compresse au passage. Vous l'avez utilisé dans le script du chapitre 10.

```
gary@serveur:~$ sudo tar -czf /srv/archives/etc.tar.gz -C / etc
gary@serveur:~$ ls -lh /srv/archives/etc.tar.gz
-rw-r--r-- 1 root root 568K Oct  4 22:04 /srv/archives/etc.tar.gz
gary@serveur:~$ tar -tzf /srv/archives/etc.tar.gz | head -n 3
etc/
etc/e2scrub.conf
etc/cron.weekly/
```

Option	Effet
`-c`, `-x`, `-t`	Créer une archive, en extraire le contenu, en lister le contenu
`-z`	Compresser avec gzip. `-J` pour xz, plus lent et plus compact
`-f fichier`	Le nom de l'archive
`-C dossier`	Se placer dans ce dossier avant d'agir
`--exclude='*.log'`	Écarter des fichiers

`-C / etc` mérite une explication. `tar` enregistre les chemins tels qu'on les lui donne. En se plaçant d'abord à la racine, on obtient des chemins relatifs, `etc/…`, que l'on pourra extraire où l'on veut. Si l'on écrit `/etc`, `tar` retire de lui-même la barre initiale et le signale : `Removing leading '/' from member names`.

L'extraction d'un seul fichier se fait en le nommant :

```
gary@serveur:~$ mkdir /tmp/restauration
gary@serveur:~$ tar -xzf /srv/archives/etc.tar.gz -C /tmp/restauration \
    etc/hostname etc/fstab
gary@serveur:~$ ls /tmp/restauration/etc
fstab  hostname
```

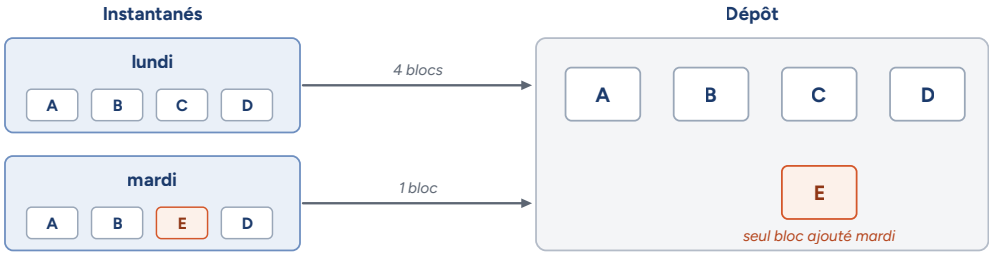
Extrayez toujours dans un dossier à part, puis recopiez ce dont vous avez besoin. Une extraction faite directement à la racine écrase les fichiers en place.

`tar` convient à une archive ponctuelle : un dossier avant une opération risquée, un transfert, un dépôt à long terme. Pour des sauvegardes quotidiennes, une archive complète par jour finit par coûter cher.

> **NOTE · dd, pour copier un disque entier**
>
> `dd` recopie un périphérique octet par octet, sans se soucier de ce qu'il contient : `sudo dd if=/dev/sdb of=/srv/archives/sdb.img bs=4M status=progress`. L'image a la taille du disque, pleine ou vide. On s'en sert pour cloner un disque ou préparer une clé d'installation, rarement pour sauvegarder. `if` désigne la source et `of` la destination : inversez-les, et vous écrasez l'original.

16.6 restic : la sauvegarde moderne

restic réunit ce qui manque aux outils précédents. Chaque sauvegarde est un *instantané* complet, mais le dépôt ne stocke qu'une fois chaque bloc de données. Tout est chiffré avant de quitter la machine. Le dépôt peut se trouver sur un disque, sur un serveur joint par SSH ou chez un hébergeur de stockage.



Chaque instantané voit tous ses fichiers. Le dépôt ne garde chaque bloc qu'une fois.

Figure 16.3 — Deux instantanés, un seul exemplaire de chaque bloc.

Il s'installe depuis le dépôt `universe` :

```
gary@serveur:~$ sudo apt install restic
gary@serveur:~$ restic version
restic 0.16.4 compiled with go1.22.2 on linux/amd64
```

Créer le dépôt

restic a besoin de deux informations : où se trouve le dépôt, et quel mot de passe le protège. On les lui donne par deux variables d'environnement. Comme la sauvegarde lira `/etc` et `/root`, on travaille sous `root`.

```
gary@serveur:~$ sudo mkdir /etc/restic
gary@serveur:~$ openssl rand -base64 24 | sudo tee /etc/restic/mot-de-passe > /dev/null
gary@serveur:~$ sudo chmod 600 /etc/restic/mot-de-passe
gary@serveur:~$ sudo -i
root@serveur:~# export RESTIC_REPOSITORY=/srv/archives/restic
root@serveur:~# export RESTIC_PASSWORD_FILE=/etc/restic/mot-de-passe
root@serveur:~# restic init
created restic repository 8dd46b3c37 at /srv/archives/restic

Please note that knowledge of your password is required to access
the repository. Losing your password means that your data is
irrecoverably lost.
```

DANGER · Le mot de passe du dépôt

L'avertissement de restic est à prendre au pied de la lettre. Le mot de passe se trouve dans `/etc/restic/mot-de-passe`, sur la machine même que l'on sauvegarde. Si elle brûle, il brûle avec elle, et la copie distante devient illisible. Recopiez-le dès maintenant dans votre gestionnaire de mots de passe.

Sauvegarder

```
root@serveur:~# restic backup /etc /home /root /srv/donnees
repository 8dd46b3c opened (version 2, compression level auto)
no parent snapshot found, will read all files
[…]
Files:         920 new,     0 changed,     0 unmodified
Dirs:          261 new,     0 changed,     0 unmodified
Added to the repository: 19.200 MiB (17.052 MiB stored)

processed 920 files, 18.687 MiB in 0:00
snapshot e663ebf9 saved
```

Ajoutons un fichier et recommençons :

```
root@serveur:~# echo "note ajoutée après coup" > /home/gary/travail/notes/mercredi.txt
root@serveur:~# restic backup /etc /home /root /srv/donnees
repository 8dd46b3c opened (version 2, compression level auto)
using parent snapshot e663ebf9
[…]
Files:           1 new,     0 changed,   920 unmodified
Dirs:            0 new,     4 changed,   257 unmodified
Added to the repository: 6.070 KiB (2.493 KiB stored)

processed 921 files, 18.687 MiB in 0:00
snapshot 2da0a268 saved
```

La seconde sauvegarde a examiné les 921 fichiers et n'a ajouté que 6 Kio au dépôt. Les deux instantanés sont pourtant complets :

```
root@serveur:~# restic snapshots
repository 8dd46b3c opened (version 2, compression level auto)
ID        Time                 Host        Tags        Paths
----------------------------------------------------------------------
e663ebf9  2026-10-04 22:14:25  serveur                 /etc
                                                       /home
                                                       /root
                                                       /srv/donnees

2da0a268  2026-10-04 22:14:26  serveur                 /etc
                                                       /home
                                                       /root
                                                       /srv/donnees
----------------------------------------------------------------------
2 snapshots
```

L'option `--exclude-file=fichier` donne à restic une liste de chemins à ignorer, un par ligne. L'atelier s'en servira pour écarter les caches.

Restaurer

Un fichier effacé par erreur se retrouve en deux commandes. `ls` montre ce que contient un instantané, `restore` en ressort ce qu'on lui désigne. Le mot `latest` désigne le plus récent.

```
root@serveur:~# rm /home/gary/travail/notes/lundi.txt
root@serveur:~# restic ls latest /home/gary/travail/notes
[…]
/home/gary/travail/notes
/home/gary/travail/notes/lundi.txt
/home/gary/travail/notes/mardi.txt
/home/gary/travail/notes/mercredi.txt
root@serveur:~# restic restore latest --target / \
    --include /home/gary/travail/notes/lundi.txt
repository 8dd46b3c opened (version 2, compression level auto)
[…]
root@serveur:~# cat /home/gary/travail/notes/lundi.txt
compte rendu du lundi
```

`--target /` remet le fichier à sa place d'origine. Pour une restauration plus large, visez plutôt un dossier vide, où vous pourrez examiner le résultat avant de recopier.

Trier et vérifier

Sans tri, les instantanés s'accumulent sans fin. `forget` applique une règle de conservation, et `--prune` libère la place des blocs que plus aucun instantané n'utilise :

```
root@serveur:~# restic forget --keep-daily 7 --keep-weekly 4 --keep-monthly 6 --prune
repository 8dd46b3c opened (version 2, compression level auto)
Applying Policy: keep 7 daily, 4 weekly, 6 monthly snapshots
[…]
```

On garde ici un instantané par jour sur sept jours, un par semaine sur quatre semaines, un par mois sur six mois. Moins d'une vingtaine d'instantanés couvrent ainsi une demi-année.

`restic check` relit la structure du dépôt et signale toute incohérence. Sa dernière ligne doit être `no errors were found`.

Mettre le dépôt ailleurs

Seule l'adresse du dépôt change :

Dépôt	Valeur de `RESTIC_REPOSITORY`
Un dossier, un disque externe	`/srv/archives/restic`
Un autre serveur, par SSH	`sftp:gary@autre-serveur:/srv/restic`
Un stockage compatible S3	`s3:https://adresse-du-service/nom-du-compartiment`
Backblaze B2	`b2:nom-du-compartiment:chemin`

Toutes les commandes restent les mêmes. Comme les données sont chiffrées sur votre machine, l'hébergeur ne voit que des blocs illisibles.

Rien n'empêche de tenir deux dépôts, l'un sur place pour restaurer vite, l'autre au loin pour le jour du sinistre. C'est la règle 3-2-1 mise en pratique.

> **NOTE · Et Borg ?**
>
> BorgBackup repose sur les mêmes principes : blocs dédupliqués, chiffrement, instantanés. Il travaille avec un dépôt local ou joint par SSH. Les deux outils sont de bons choix. restic l'emporte quand on vise un stockage en ligne.

16.7 Sauvegarder un volume en cours d'utilisation

Pendant qu'une sauvegarde parcourt un gros volume, les fichiers continuent de changer. Le début de la sauvegarde et sa fin ne montrent pas le même instant.

L'instantané LVM du chapitre 15 règle la question. On fige le volume, on sauvegarde l'image figée, on supprime l'instantané. Le dossier `/mnt/fige` a été créé au préalable.

```
root@serveur:~# lvcreate -s -n donnees-fige -L 500M /dev/vg-miroir/donnees
  Logical volume "donnees-fige" created.
root@serveur:~# mount -o ro /dev/vg-miroir/donnees-fige /mnt/fige
root@serveur:~# restic backup --tag miroir /mnt/fige
[…]
Files:        3435 new,     0 changed,     0 unmodified
Dirs:          806 new,     0 changed,     0 unmodified
Added to the repository: 19.357 MiB (12.629 MiB stored)

processed 3435 files, 20.821 MiB in 0:00
snapshot daec0604 saved
root@serveur:~# umount /mnt/fige
root@serveur:~# lvremove -y /dev/vg-miroir/donnees-fige
  Logical volume "donnees-fige" successfully removed.
```

L'instantané n'a vécu que le temps de la copie. L'option `--tag` pose une étiquette sur la sauvegarde, pour la retrouver avec `restic snapshots --tag miroir`.

16.8 Automatiser et surveiller

Une sauvegarde qu'il faut penser à lancer finit par ne plus l'être. Celle-ci sera confiée à une minuterie systemd, comme au chapitre 13.

Reste le vrai risque : la sauvegarde qui échoue en silence. Un disque d'archives plein, un mot de passe changé, un serveur distant injoignable, et plus rien n'est sauvegardé pendant des mois sans que personne le remarque. Deux protections se complètent.

La première est d'être prévenu de l'échec. La directive `OnFailure=` d'un service désigne une unité à lancer quand il échoue. L'atelier la met en place.

La seconde est de s'apercevoir de l'absence de succès. Une machine éteinte n'envoie pas d'alerte. La parade consiste à inverser la logique : la sauvegarde signale chaque réussite à un service de surveillance extérieur, et c'est ce service qui s'inquiète quand il n'a plus de nouvelles. Le chapitre 26 présente ce type d'outil.

16.9 Tester la restauration

On ne sait qu'une sauvegarde fonctionne qu'après avoir restauré. Tant que ce n'est pas fait, on ignore si elle contient les bons fichiers, si le mot de passe noté est le bon, et combien de temps il faudra le jour venu.

Le test le plus simple est une restauration complète dans un dossier vide, suivie d'une comparaison avec l'original :

```
root@serveur:~# restic restore latest --path /etc --target /tmp/restauration | tail -n 1
Summary: Restored 1782 files/dirs (19.489 MiB) in 0:00
root@serveur:~# diff -r /etc /tmp/restauration/etc && echo "identiques"
diff: /tmp/restauration/etc/mtab: No such file or directory
diff: /tmp/restauration/etc/os-release: No such file or directory
diff: /tmp/restauration/etc/resolv.conf: No such file or directory
Only in /etc/systemd/system/timers.target.wants: sauvegarde-restic.timer
```

Aucun fichier n'a un contenu différent. Les trois premières lignes concernent des liens symboliques qui visent des fichiers situés hors de `/etc`. Restaurés dans `/tmp/restauration`, ils pointent dans le vide, mais les liens eux-mêmes sont bien là. La

quatrième signale un fichier créé depuis la sauvegarde : la minuterie de l'atelier, activée entre-temps.

> **ATTENTION · latest n'est pas toujours celui qu'on croit**
>
> `latest` désigne l'instantané le plus récent du dépôt, tous contenus confondus. Ici, c'était celui du volume figé, et non celui du système. `--path /etc` restreint le choix aux instantanés qui contiennent `/etc`. En cas de doute, donnez l'identifiant lu dans `restic snapshots`.

Fréquence	Test
Chaque mois	Restaurer un fichier pris au hasard et le comparer à l'original
Chaque trimestre	Restaurer un dossier entier dans un emplacement vide
Chaque année	Reconstruire le service sur une machine vierge, à partir de la seule sauvegarde, montre en main

Le test annuel est le seul qui réponde à la vraie question : en combien de temps sommes-nous de nouveau en service ? Il révèle aussi ce qu'on avait oublié de sauvegarder.

En résumé

- Trois copies, deux supports, un autre lieu.
- Ni le RAID, ni un instantané, ni une synchronisation ne sont des sauvegardes.
- On sauvegarde ce qu'on ne saurait pas refaire : `/home`, `/etc`, `/srv`, les exports des bases de données.
- `rsync -a source/ destination/` copie les différences. `-n` avant tout `--delete`.
- `rsync --link-dest` garde une copie par jour au prix des seuls fichiers modifiés.
- `tar -czf` crée une archive, `-tzf` la liste, `-xzf` en extrait.
- restic chiffre, déduplique et garde des instantanés. Son mot de passe se conserve ailleurs que sur la machine.
- `restic forget --prune` applique la règle de conservation, `restic check` vérifie le dépôt.
- Un instantané LVM fige un volume le temps de la sauvegarde.

- Une sauvegarde automatique doit alerter quand elle échoue.
- Une sauvegarde n'existe qu'après une restauration réussie.

Atelier — Une sauvegarde de bout en bout

Objectif : mettre en place la sauvegarde quotidienne du serveur avec restic, la faire tourner par une minuterie, recevoir une alerte quand elle échoue, et prouver qu'on sait restaurer. Comptez une heure.

Le dépôt

1. Installez restic, créez `/etc/restic` et le fichier du mot de passe, puis initialisez le dépôt dans `/srv/archives/restic`, comme dans la section sur restic. Notez le mot de passe hors de la machine.
2. Créez `/etc/restic/env`, lisible par `root` seul (`sudo chmod 600`). Le service y lira l'adresse du dépôt et l'emplacement du mot de passe :

```
RESTIC_REPOSITORY=/srv/archives/restic
RESTIC_PASSWORD_FILE=/etc/restic/mot-de-passe
```

3. Créez `/etc/restic/exclusions` :

```
# Ce qui ne mérite pas d'être sauvegardé
/home/*/.cache
/var/cache
/var/tmp
*.tmp
```

Le script

4. Créez `/usr/local/bin/sauvegarde-restic` et rendez-le exécutable :

```
#!/bin/bash
#
# sauvegarde-restic.sh — sauvegarde quotidienne avec restic, puis tri des anciens
instantanés
# Lancé par sauvegarde-restic.service, qui fournit le dépôt et le mot de passe
(EnvironmentFile).

set -euo pipefail
```

```
restic backup --exclude-file=/etc/restic/exclusions /etc /home /root /srv/donnees
restic forget --keep-daily 7 --keep-weekly 4 --keep-monthly 6 --prune
restic check
```

Grâce à `set -e`, le script s'arrête à la première commande qui échoue, et systemd le saura.

L'alerte

5. Créez le script `/usr/local/bin/alerte`, exécutable lui aussi :

```
#!/bin/bash
#
# alerte.sh — signale l'échec d'une unité systemd
# Usage : alerte NOM-DE-L-UNITÉ

unite="$1"
logger -t alerte -p user.crit "ÉCHEC de $unite sur $(hostname)"
# Ajoutez ici l'envoi d'un courriel ou l'appel d'un service de messagerie.
```

6. Créez l'unité `/etc/systemd/system/alerte@.service`. L'arobase en fait un modèle : `%i` y sera remplacé par ce qui suit l'arobase quand on l'appellera.

```
[Unit]
Description=Alerte : échec de %i

[Service]
Type=oneshot
ExecStart=/usr/local/bin/alerte %i
```

Le service et la minuterie

7. Créez `/etc/systemd/system/sauvegarde-restic.service` :

```
[Unit]
Description=Sauvegarde restic
OnFailure=alerte@%n.service

[Service]
Type=oneshot
EnvironmentFile=/etc/restic/env
ExecStart=/usr/local/bin/sauvegarde-restic
Nice=19
IOSchedulingClass=idle
```

`OnFailure=alerte@%n.service` lance l'unité d'alerte en lui passant le nom du service en échec, que systemd met à la place de `%n`.

8. Créez `/etc/systemd/system/sauvegarde-restic.timer` :

```
[Unit]
Description=Sauvegarde restic, chaque nuit

[Timer]
OnCalendar=*-*-* 01:30:00
RandomizedDelaySec=20min
Persistent=true

[Install]
WantedBy=timers.target
```

9. Essayez le service à la main, puis activez la minuterie :

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl start sauvegarde-restic.service
gary@serveur:~$ journalctl -u sauvegarde-restic -o cat | tail -n 4
no errors were found
sauvegarde-restic.service: Deactivated successfully.
Finished sauvegarde-restic.service - Sauvegarde restic.
sauvegarde-restic.service: Consumed 1.660s CPU time.
gary@serveur:~$ sudo systemctl enable --now sauvegarde-restic.timer
```

Provoquer un échec

10. Rendez le dépôt introuvable, lancez la sauvegarde et cherchez l'alerte :

```
gary@serveur:~$ sudo mv /srv/archives/restic /srv/archives/restic.absent
gary@serveur:~$ sudo systemctl start sauvegarde-restic.service
Job for sauvegarde-restic.service failed because the control process exited with error code.
[…]
gary@serveur:~$ journalctl -t alerte -n 1
Oct 04 22:14:48 serveur alerte[1848]: ÉCHEC de sauvegarde-restic.service sur serveur
gary@serveur:~$ journalctl -u sauvegarde-restic -n 1 -o cat
sauvegarde-restic.service: Triggering OnFailure= dependencies.
```

L'alerte est partie. Elle n'est pour l'instant écrite que dans le journal, au niveau `crit`. Le script `alerte` est l'endroit où brancher un courriel ou un message, selon ce dont vous disposez.

11. Remettez le dépôt en place et effacez l'échec :

```
sudo mv /srv/archives/restic.absent /srv/archives/restic
sudo systemctl reset-failed
```

Restaurer

12. Supprimez un fichier de votre répertoire personnel, puis restaurez-le depuis le dernier instantané. Les commandes `restic` se lancent sous `root`, après avoir exporté les deux variables.
13. Restaurez dans `/tmp/restauration` le dernier instantané qui contient `/etc`, comparez avec `diff -r`, puis supprimez le dossier.

Vérifier

- `systemctl list-timers sauvegarde-restic.timer` annonce la prochaine sauvegarde.
- `sudo restic snapshots`, avec les deux variables, liste au moins trois instantanés.
- Le mot de passe du dépôt est noté ailleurs que sur le serveur.
- Demain matin, `journalctl -u sauvegarde-restic --since yesterday` montrera la sauvegarde de la nuit.

Il manque encore à ce serveur une copie hors site. Dès que vous disposerez d'une seconde machine, créez-y un second dépôt par SFTP et ajoutez-le au script.

Exercices

Exercice 1 — Une stratégie

Une association tient sa comptabilité et ses dossiers d'adhérents sur un petit serveur Ubuntu installé dans son local. Elle possède aussi un disque USB et un compte chez un hébergeur de stockage.

1. Proposez une organisation qui respecte la règle 3-2-1. Où se trouve chaque copie ?
2. Que sauvegardez-vous, et que laissez-vous ?
3. Le local est cambriolé, le serveur et le disque USB disparaissent. Que reste-t-il, et de quoi avez-vous besoin pour le relire ?

Exercice 2 — rsync

1. Copiez `/etc` dans `/srv/archives/etc-copie/`, en conservant droits et propriétaires.

2. Créez un fichier dans la copie, puis relancez la commande avec `--delete` et `-n`. Que ferait `rsync` ?
3. Excluez de la copie tous les fichiers dont le nom se termine par `.bak`.
4. Mettez en place trois copies datées avec `--link-dest`, en modifiant un fichier entre chacune. Quelle place occupe l'ensemble ?

Exercice 3 — tar

1. Archivez votre répertoire personnel, sans le dossier `.cache`.
2. Listez le contenu de l'archive et comptez les fichiers.
3. Extrayez un seul fichier dans `/tmp`.
4. Comparez la taille obtenue avec `-z` et avec `-J`.

Exercice 4 — restic

1. Créez un dépôt dans `/srv/archives/essai`, avec un mot de passe saisi au clavier.
2. Sauvegardez `~/travail` trois fois, en modifiant un fichier entre chaque passage.
3. Affichez ce qui distingue le premier instantané du dernier avec `restic diff`.
4. Ne conservez que les deux derniers instantanés.
5. Relevez la place occupée par le dépôt avant et après.

Exercice 5 — Diagnostic

1. La sauvegarde de la nuit n'a pas eu lieu. Quelles commandes lancez-vous, dans l'ordre ?
2. `restic` répond `wrong password or no key found`. Quelles sont les causes possibles ?
3. Le disque d'archives est plein. Comment faire de la place sans perdre l'historique récent ?
4. Un collègue sauvegarde `/var/lib/postgresql` par `rsync` chaque nuit, base en marche. Que lui dites-vous ?

Pour aller plus loin

Sur la machine. `man rsync` décrit plus de cent options, dont `--backup-dir`, qui met de côté les fichiers remplacés au lieu de les perdre. `restic help` suivi d'un nom de commande donne le détail de chacune.

Prolongements. `restic mount` présente le dépôt comme un dossier où l'on se promène d'instantané en instantané. `rclone` synchronise des fichiers avec plusieurs dizaines de services de stockage, et restic sait s'appuyer sur lui. Pour une machine entière, les hyperviseurs proposent leurs propres sauvegardes, qui complètent celle des fichiers sans la remplacer.

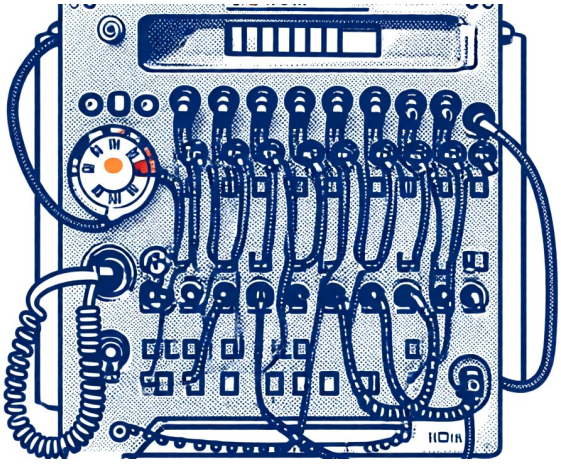
En ligne. La documentation de restic, sur `restic.readthedocs.io`, est courte et exemplaire. Son chapitre sur la préparation d'un dépôt passe en revue tous les types de stockage.

PARTIE VI

Réseau sous Ubuntu

Configurer le réseau, comprendre ce qui circule sur le câble et administrer une machine à distance.





CHAPITRE 17

Les fondamentaux du réseau

Une adresse, un masque, une passerelle, un serveur de noms. Quatre réglages relient une machine au reste du monde. Ce chapitre dit ce qu'ils signifient, et donne la méthode pour trouver lequel est en cause quand plus rien ne répond.

DANS CE CHAPITRE

- Situer les protocoles dans les couches du modèle TCP/IP.
- Lire une adresse IPv4 et son masque, délimiter un sous-réseau.
- Reconnaître les adresses privées et les adresses IPv6.
- Expliquer le rôle des ports, de TCP et d'UDP.
- Décrire le trajet d'une résolution de nom.
- Examiner interfaces, adresses et routes avec la commande `ip`.
- Tester une liaison, un nom, un port avec `ping`, `dig`, `ss`, `nc` et `curl`.
- Diagnostiquer une panne de réseau avec méthode.

> **AVANT DE COMMENCER**
>
> Le chapitre 12. Les exemples se déroulent sur le serveur des chapitres précédents, branché sur un réseau local ordinaire : il a reçu l'adresse 192.168.1.50 d'un routeur situé en 192.168.1.1. Vos adresses seront différentes, les commandes sont les mêmes.

17.1 Les couches

Envoyer une page web d'un continent à l'autre pose des problèmes de natures très différentes : faire passer un signal dans un câble, trouver un chemin entre des milliers de réseaux, remettre les données au bon programme, et dans le bon ordre. Le réseau les traite séparément, par couches. Chaque couche rend un service à celle du dessus sans s'occuper du reste.

Deux découpages coexistent. Le modèle OSI, normalisé en 1984, compte sept couches. On s'y réfère par leurs numéros : un « problème de couche 2 » concerne la liaison. Le modèle TCP/IP, celui d'Internet, en compte quatre. C'est lui qui décrit ce qui tourne réellement sur une machine.

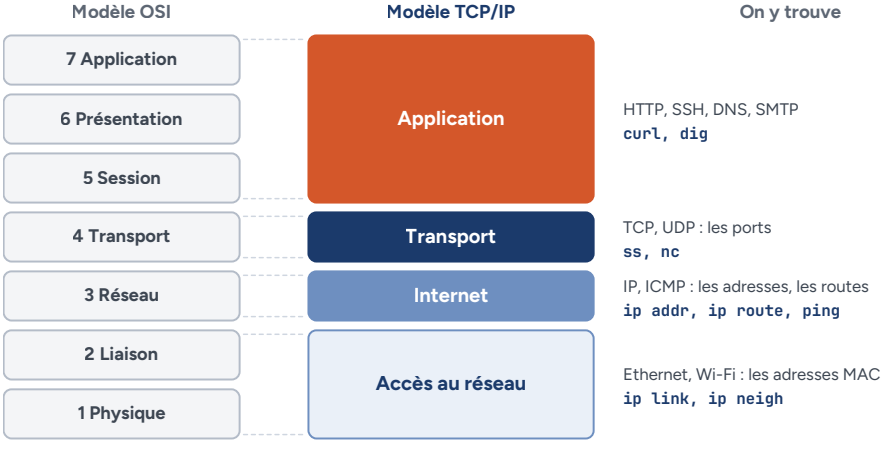


Figure 17.1 — Les deux modèles, et ce qu'on trouve à chaque étage.

Couche	Son travail	Ses adresses
Accès au réseau	Transmettre entre deux machines voisines, par câble ou par radio	Adresse MAC, fixée par le fabricant de la carte
Internet	Acheminer de réseau en réseau jusqu'à la destination	Adresse IP

Couche	Son travail	Ses adresses
Transport	Remettre les données au bon programme	Numéro de port
Application	Parler le langage du service : web, courrier, noms	Nom, adresse web

Quand un programme envoie des données, chaque couche les glisse dans une enveloppe portant ses propres adresses. À l'arrivée, chaque couche ouvre son enveloppe et passe le contenu à la suivante.

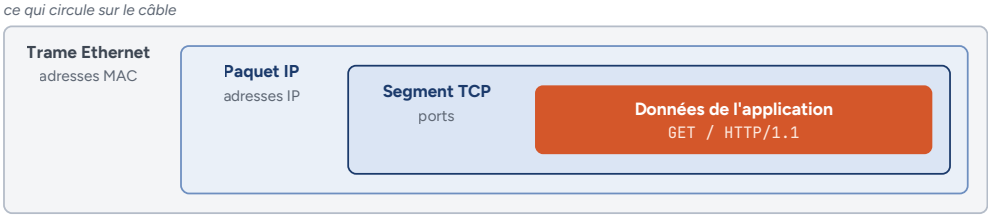


Figure 17.2 — L'emboîtement des enveloppes.

Ce découpage sert tous les jours. Devant une panne, on cherche d'abord à quel étage elle se trouve. Un nom qui ne se traduit pas en adresse n'a rien à voir avec un câble débranché, et ne se répare pas de la même façon.

17.2 Les adresses IPv4

Une adresse IPv4 tient sur 32 bits. On l'écrit sous la forme de quatre nombres de 0 à 255 séparés par des points : `192.168.1.50`.

Elle se lit en deux parties. Le début désigne un réseau, la fin désigne une machine dans ce réseau. La frontière entre les deux est donnée par le **masque**, que l'on note par le nombre de bits de la partie réseau, après une barre oblique. C'est la notation CIDR.



Figure 17.3 — L'adresse 192.168.1.50/24.

Dans `192.168.1.50/24`, les 24 premiers bits désignent le réseau `192.168.1.0`. Les 8 bits restants laissent 256 valeurs. Deux sont réservées : la première, `192.168.1.0`, nomme le réseau lui-même, et la dernière, `192.168.1.255`, est l'adresse de diffusion, qui s'adresse à toutes les machines à la fois. Il reste 254 adresses pour des machines.

L'ancienne écriture du masque, `255.255.255.0`, dit la même chose : les trois premiers octets sont ceux du réseau.

Deux machines du même réseau se parlent directement. Pour tout le reste, elles confient leurs paquets à un routeur, la **passerelle**. C'est le masque qui permet à chaque machine de savoir si son correspondant est un voisin ou non.

Notation	Masque	Machines	Usage courant
/8	255.0.0.0	16 777 214	Très grand réseau privé
/16	255.255.0.0	65 534	Réseau d'un site
/24	255.255.255.0	254	Réseau domestique, petit bureau
/26	255.255.255.192	62	Un service, un étage
/30	255.255.255.252	2	Liaison entre deux routeurs
/32	255.255.255.255	1	Une seule machine

La règle : un `/N` contient 2 puissance (32 − N) adresses, moins deux. Dès qu'on sort des octets entiers, le calcul de tête devient pénible. Python s'en charge :

```
gary@serveur:~$ python3 -c "import ipaddress as i; \
    print(i.ip_address('192.168.1.77') in i.ip_network('192.168.1.64/26'))"
True
```

Le paquet `ipcalc`, du dépôt `universe`, présente le même calcul de façon plus lisible.

Adresses privées, adresses publiques

Il n'y a que quatre milliards d'adresses IPv4, et la réserve mondiale a été entièrement distribuée en 2011. Les réseaux locaux emploient donc des adresses réservées à cet usage, que tout le monde réutilise chez soi :

Plage	Usage
`10.0.0.0/8`	Réseaux privés
`172.16.0.0/12`	Réseaux privés. Docker y prend ses adresses par défaut
`192.168.0.0/16`	Réseaux privés. C'est la plage des box domestiques
`127.0.0.0/8`	La machine elle-même. `127.0.0.1` s'appelle `localhost`
`169.254.0.0/16`	Adresses que se donnent des machines sans configuration
`100.64.0.0/10`	Adresses partagées, internes aux opérateurs

Ces adresses ne circulent pas sur Internet. À la sortie du réseau local, le routeur remplace l'adresse privée de la machine par sa propre adresse publique, et fait l'opération inverse au retour. C'est la traduction d'adresses, ou **NAT**. Conséquence pratique : de l'extérieur, toutes les machines d'une maison se présentent sous la même adresse, et aucune n'est joignable directement.

17.3 IPv6 en bref

IPv6 règle la pénurie en passant à 128 bits. L'adresse s'écrit en huit groupes de chiffres hexadécimaux séparés par des deux-points. Deux règles l'abrègent : les zéros en tête d'un groupe se suppriment, et une suite de groupes nuls se remplace, une seule fois, par `::`.

```
2620:002d:4000:0001:0000:0000:0000:0027
2620:2d:4000:1::27
```

Préfixe	Type d'adresse
`::1`	La machine elle-même
`fe80::/10`	Lien local : ne sort pas du réseau où la machine est branchée

Préfixe	Type d'adresse
`fd00::/8`	Adresses privées
`2000::/3`	Adresses publiques

Toute interface a une adresse de lien local, que le système se donne lui-même :

```
gary@serveur:~$ ip -6 addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP […]
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever
```

Si le routeur annonce un préfixe public, la machine se fabrique en plus une adresse publique, sans serveur DHCP. Ce réseau-ci n'en annonce pas : le serveur n'a que son adresse de lien local, et tout son trafic passe en IPv4. C'est encore la situation de beaucoup de réseaux. Tout ce qui suit vaut pour les deux protocoles. Les commandes acceptent `-4` ou `-6` pour n'en regarder qu'un.

17.4 Les ports, TCP et UDP

L'adresse IP mène à une machine. Or une machine fait tourner plusieurs services. Le **port**, un nombre de 1 à 65 535, désigne celui auquel on s'adresse.

Deux protocoles de transport se partagent les ports.

TCP établit une connexion, numérote ce qu'il envoie, attend les accusés de réception et renvoie ce qui s'est perdu. Ce qui arrive est complet et dans l'ordre. Le web, SSH et le courrier reposent sur lui.

UDP envoie sans connexion ni garantie. Il est plus léger, et convient quand un paquet perdu ne vaut pas la peine d'être renvoyé : résolution de noms, voix, vidéo en direct.

Port	Service
22	SSH
25, 587	Courrier sortant (SMTP)
53	DNS
80, 443	Web : HTTP et HTTPS
993	Courrier entrant (IMAP chiffré)

Port	Service
3306, 5432	Bases de données MariaDB et PostgreSQL

Le fichier `/etc/services` tient la liste complète. Les ports inférieurs à 1024 sont réservés : seul un programme lancé par `root` peut s'y installer.

Quand votre machine ouvre une connexion, elle utilise elle aussi un port, tiré au hasard dans une plage réservée aux connexions sortantes :

```
gary@serveur:~$ cat /proc/sys/net/ipv4/ip_local_port_range
32768   60999
```

Une connexion TCP est définie par quatre valeurs : deux adresses et deux ports. C'est ce qui permet à un serveur web de tenir des milliers de conversations sur son seul port 443.

17.5 Le DNS

Personne ne retient `185.125.190.21`. Le **DNS** (*Domain Name System*) est l'annuaire qui traduit les noms en adresses.

Cet annuaire n'est rangé nulle part en entier. Il est réparti. Les serveurs de la racine connaissent les serveurs de `.com`, de `.fr`, de `.cm`. Ceux de `.com` connaissent les serveurs de chaque domaine en `.com`. Et seuls les serveurs d'un domaine connaissent les adresses de ses machines.

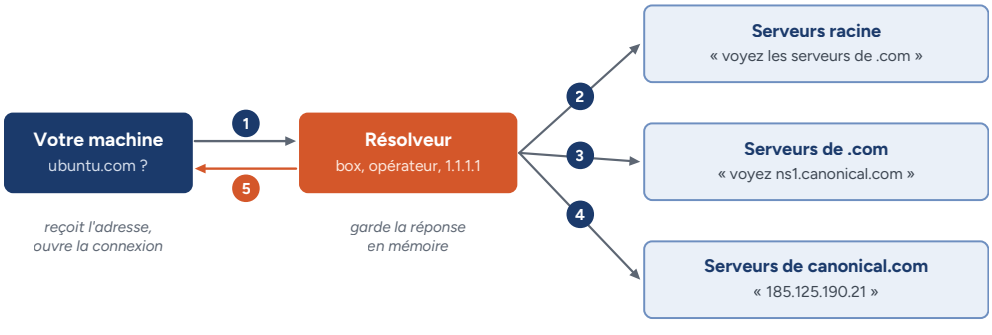


Figure 17.4 — La résolution du nom ubuntu.com.

Votre machine ne fait pas ce parcours. Elle pose la question à un **résolveur**, celui de la box ou de l'opérateur, qui interroge les serveurs successifs et retient la réponse

quelque temps. Chaque réponse porte une durée de validité en secondes, le **TTL**. Tant qu'elle court, le résolveur ne repose pas la question. Voilà pourquoi un changement d'adresse met parfois des heures à se propager.

L'annuaire contient plusieurs sortes de fiches :

Type	Contenu	Exemple pour `ubuntu.com`
`A`	Adresse IPv4	`185.125.190.21`
`AAAA`	Adresse IPv6	`2620:2d:4000:1::27`
`CNAME`	Renvoi vers un autre nom	
`MX`	Serveur qui reçoit le courrier du domaine	`10 mx.ubuntu.com.`
`NS`	Serveurs qui détiennent les fiches du domaine	`ns1.canonical.com.`
`TXT`	Texte libre : preuves de propriété, règles du courrier	
`PTR`	Nom associé à une adresse, pour la recherche inverse	

Avant d'interroger le DNS, la machine consulte un fichier local, `/etc/hosts`. Une ligne y associe une adresse à des noms. C'est le moyen de donner un nom à une machine du réseau local, ou d'essayer un site avant de modifier le vrai DNS.

17.6 Examiner la machine avec ip

La commande `ip` montre et règle tout ce qui concerne le réseau. Son option `-br` donne une vue résumée :

```
gary@serveur:~$ ip -br addr
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             192.168.1.50/24 metric 100 fe80::5054:ff:fe12:3456/64
```

La machine a deux interfaces. `lo` est la boucle locale, par laquelle elle se parle à elle-même. `enp0s3` est sa carte réseau. Le nom décrit l'emplacement de la carte : `en` pour Ethernet, `p0s3` pour le bus et le connecteur. Une carte Wi-Fi s'appellerait `wlp…`.

La forme complète en dit davantage :

```
gary@serveur:~$ ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP […]
    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.50/24 metric 100 brd 192.168.1.255 scope global dynamic enp0s3
```

```
       valid_lft 43056sec preferred_lft 43056sec
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever
```

- `UP` signifie que l'interface est activée, `LOWER_UP` que le câble est branché. Sans `LOWER_UP`, cherchez du côté du câble ou du commutateur.
- `link/ether` donne l'adresse MAC de la carte.
- `inet` donne l'adresse IPv4 et son masque. Le mot `dynamic` indique qu'elle vient d'un serveur DHCP, pour la durée affichée à la ligne suivante.

Les routes

```
gary@serveur:~$ ip route
default via 192.168.1.1 dev enp0s3 proto dhcp src 192.168.1.50 metric 100
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.50 metric 100
```

La table de routage se lit de la ligne la plus précise à la plus générale. La seconde ligne dit que le réseau `192.168.1.0/24` est joignable directement par `enp0s3`. La première, `default`, envoie tout le reste à la passerelle `192.168.1.1`. Sans ligne `default`, la machine ne sort pas de son réseau local.

`ip route get` indique le chemin que prendrait un paquet :

```
gary@serveur:~$ ip route get 1.1.1.1
1.1.1.1 via 192.168.1.1 dev enp0s3 src 192.168.1.50 uid 1000
gary@serveur:~$ ip route get 192.168.1.20
192.168.1.20 dev enp0s3 src 192.168.1.50 uid 1000
```

Le premier passe par la passerelle, le second va droit au voisin.

`ip neigh` liste les voisins avec lesquels la machine a échangé récemment, et leur adresse MAC :

```
gary@serveur:~$ ip neigh
192.168.1.1 dev enp0s3 lladdr 8a:f9:03:8d:6c:0a DELAY
```

Commande	Ce qu'elle montre
`ip -br link`	Les interfaces et leur état
`ip -br addr`	Les adresses
`ip route`	La table de routage
`ip neigh`	Les voisins du réseau local

Commande	Ce qu'elle montre
`ip -s link show enp0s3`	Les compteurs de paquets et d'erreurs d'une interface

> **NOTE · Les anciennes commandes**
>
> `ifconfig`, `route`, `netstat` et `arp` remplissent d'innombrables tutoriels. Elles appartiennent au paquet `net-tools`, qu'Ubuntu n'installe plus. `ip addr`, `ip route`, `ss` et `ip neigh` les remplacent.

`ip` sait aussi modifier : `sudo ip addr add`, `sudo ip route add`. Ces changements disparaissent au redémarrage. Ils servent à faire un essai. La configuration durable est l'affaire de Netplan, au chapitre 18.

17.7 Tester la liaison

`ping` envoie de petits paquets à une machine et mesure le temps de l'aller-retour. Sous Linux, il continue jusqu'à Ctrl+C, à moins qu'on ne fixe un nombre avec `-c`.

```
gary@serveur:~$ ping -c 3 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=54 time=22.5 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=54 time=20.5 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=54 time=116 ms

--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 20.500/52.953/115.818/44.459 ms
```

Deux chiffres comptent : le taux de perte, et le temps de réponse. Une réponse de la passerelle prend moins d'une milliseconde, une réponse d'un autre continent entre 100 et 300.

Un `ping` sans réponse ne prouve pas que la machine est éteinte. Beaucoup de pare-feu rejettent ces paquets. En revanche, un `ping` qui répond prouve que la route fonctionne dans les deux sens.

`mtr` montre le chemin, routeur après routeur :

```
gary@serveur:~$ mtr -rwnc 3 1.1.1.1
Start: 2026-10-04T22:23:54+0000
HOST: serveur        Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 192.168.1.1     0.0%     3    0.6   0.5   0.3   0.6   0.2
[…]
```

```
  5.|-- ???              100.0     3    0.0   0.0   0.0   0.0   0.0
  6.|-- 100.64.0.1       0.0%     3   25.9  54.6  25.9 109.5  47.6
[…]
 10.|-- 172.69.12.2      0.0%     3   24.8  48.9  19.4 102.4  46.5
 11.|-- 1.1.1.1          0.0%     3  132.5  59.5  21.5 132.5  63.2
```

Onze étapes séparent ce serveur de sa destination. La première est la passerelle. La sixième porte une adresse en `100.64` : nous sommes chez l'opérateur. La cinquième ne répond pas, et ce n'est pas une panne, puisque les suivantes répondent : ce routeur transmet les paquets mais refuse de se nommer. Une vraie coupure se reconnaît à ce que plus rien ne répond au-delà d'une étape.

Sans option, `mtr` s'affiche en continu, ce qui permet de voir une perte apparaître. `tracepath` rend le même service en plus rustique.

17.8 Interroger le DNS

`dig` pose une question au DNS. Avec `+short`, il ne donne que la réponse :

```
gary@serveur:~$ dig +short ubuntu.com
185.125.190.20
185.125.190.21
185.125.190.29
gary@serveur:~$ dig +short MX ubuntu.com
10 mx.ubuntu.com.
gary@serveur:~$ dig +short NS ubuntu.com
ns2.canonical.com.
ns1.canonical.com.
ns3.canonical.com.
gary@serveur:~$ dig +short -x 1.1.1.1
one.one.one.one.
```

Un nom peut avoir plusieurs adresses : la charge se répartit entre plusieurs serveurs. `-x` fait la recherche inverse, de l'adresse vers le nom.

Sans `+short`, `dig` détaille sa réponse. Trois lignes méritent l'attention :

```
gary@serveur:~$ dig ubuntu.com
[…]
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12098
[…]
;; ANSWER SECTION:
ubuntu.com.             39      IN      A       185.125.190.21
[…]
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
```

`status: NOERROR` signale une réponse normale. `NXDOMAIN` voudrait dire que le nom n'existe pas, `SERVFAIL` que le résolveur n'a pas pu obtenir de réponse. Le `39` est le TTL restant, en secondes. La ligne `SERVER` nomme celui qui a répondu.

Qui répond ?

L'adresse `127.0.0.53` n'est ni la box ni l'opérateur. C'est `systemd-resolved`, un relais installé sur la machine. Les programmes s'adressent à lui, parce que `/etc/resolv.conf` le leur désigne. Il transmet aux vrais serveurs et garde les réponses en mémoire.

```
gary@serveur:~$ resolvectl status
Global
         Protocols: -LLMNR -mDNS -DNSOverTLS DNSSEC=no/unsupported
  resolv.conf mode: stub

Link 2 (enp0s3)
    Current Scopes: DNS
         Protocols: +DefaultRoute -LLMNR -mDNS -DNSOverTLS
                    DNSSEC=no/unsupported
Current DNS Server: 192.168.1.1
       DNS Servers: 192.168.1.1
        DNS Domain: maison.lan
```

Le serveur réellement interrogé est `192.168.1.1`, reçu par DHCP en même temps que l'adresse. Ne modifiez pas `/etc/resolv.conf` à la main : ce fichier est tenu par `systemd-resolved`. Le chapitre 18 montrera où déclarer d'autres serveurs.

Pour savoir si un problème vient de votre résolveur, posez la question à un autre en le nommant après une arobase :

```
gary@serveur:~$ dig +short @1.1.1.1 ubuntu.com
185.125.190.29
185.125.190.21
185.125.190.20
```

`dig +trace ubuntu.com` refait enfin tout le parcours de la figure, depuis les treize serveurs de la racine. C'est le meilleur moyen de voir le DNS fonctionner.

17.9 Ports et services

Ce qui écoute sur la machine

`ss` liste les connexions et les ports ouverts. La combinaison à retenir est `-tlnp` : TCP, en écoute, en chiffres, avec le programme.

```
gary@serveur:~$ sudo ss -tlnp
State   Recv-Q  Send-Q  Local Address:Port  Peer Address:Port  Process
LISTEN  0       4096          0.0.0.0:22         0.0.0.0:*     users:(("sshd",pid=1022,…
LISTEN  0       4096      127.0.0.54:53          0.0.0.0:*     users:(("systemd-resolve",…
LISTEN  0       4096    127.0.0.53%lo:53         0.0.0.0:*     users:(("systemd-resolve",…
LISTEN  0       4096             [::]:22            [::]:*     users:(("sshd",pid=1022,…
```

La colonne `Local Address` dit sur quelle adresse le service écoute, et c'est une information de sécurité. `0.0.0.0:22` : SSH accepte les connexions sur toutes les interfaces, donc depuis le réseau. `127.0.0.53:53` : le relais DNS n'écoute que la boucle locale, et personne ne peut l'atteindre du dehors.

Avec `-u` à la place de `-t`, `ss` montre les ports UDP. Sans `sudo`, la colonne des programmes reste vide pour les services des autres comptes.

Ce qui répond ailleurs

`nc` essaie d'ouvrir une connexion vers un port. Avec `-z`, il se contente de vérifier qu'elle s'établit :

```
gary@serveur:~$ nc -zv 192.168.1.1 53
Connection to 192.168.1.1 53 port [tcp/domain] succeeded!
gary@serveur:~$ nc -zv -w 2 192.168.1.1 81
nc: connect to 192.168.1.1 port 81 (tcp) failed: Connection refused
```

Trois réponses sont possibles, et chacune désigne une cause. `succeeded` : un service écoute. `Connection refused` : la machine a répondu qu'aucun service n'écoute sur ce port. Pas de réponse jusqu'à l'expiration du délai : un pare-feu jette les paquets en silence, ou la machine est injoignable.

`curl` va un cran plus haut et parle HTTP. `-I` ne demande que les en-têtes de la réponse :

```
gary@serveur:~$ curl -sI https://ubuntu.com | head -n 4
HTTP/2 200
server: nginx/1.14.0 (Ubuntu)
date: Sun, 04 Oct 2026 22:24:11 GMT
content-type: text/html; charset=utf-8
```

Un code `200` : le site répond. `301` ou `302` : il renvoie ailleurs. `404` : la page n'existe pas. `500` et au-delà : le serveur a un problème. Avec `-v`, `curl` montre chaque étape, de la connexion au certificat. C'est l'outil de base pour tout ce qui touche au web.

17.10 Regarder passer les paquets

Quand les tests ne suffisent plus, `tcpdump` affiche les paquets eux-mêmes. `-n` laisse les adresses en chiffres, `-i` choisit l'interface, `-c` s'arrête après un nombre de paquets. Un filtre, à la fin, dit lesquels retenir.

Voici ce que montre `sudo tcpdump -ni enp0s3 -c 2 port 53` pendant un `dig` :

```
22:24:20.314840 IP 192.168.1.50.48961 > 192.168.1.1.53: 44727+ [1au] A? ubuntu.com. (51)
22:24:20.315190 IP 192.168.1.1.53 > 192.168.1.50.48961: 44727 3/0/1 A 185.125.190.21, […]
```

Une question part du port 48961 du serveur vers le port 53 du résolveur : `A? ubuntu.com`. La réponse revient par le chemin inverse, trois dixièmes de milliseconde plus tard.

Et voici l'ouverture d'une connexion TCP vers un serveur web, avec le filtre `tcp port 80` :

```
22:24:23.536035 IP 192.168.1.50.33312 > 185.125.190.20.80: Flags [S], seq 1263418262, […]
22:24:23.704399 IP 185.125.190.20.80 > 192.168.1.50.33312: Flags [S.], […] ack 1263418263,
[…]
22:24:23.704453 IP 192.168.1.50.33312 > 185.125.190.20.80: Flags [.], ack 1, […]
```

Trois paquets : c'est la poignée de main de TCP. `[S]`, le client demande la connexion. `[S.]`, le serveur accepte. `[.]`, le client confirme. La conversation peut commencer. Un `[S]` qui reste sans réponse désigne un pare-feu ou une machine absente. Un `[R]` en retour est un refus.

Filtre	Paquets retenus
`host 192.168.1.1`	Ceux qui viennent de cette machine ou qui y vont
`port 443`	Ceux du port 443, TCP ou UDP
`icmp`	Ceux de `ping`
`tcp port 80 and host 192.168.1.20`	Les deux conditions réunies

Filtre	Paquets retenus
`not port 22`	Tout sauf SSH. Utile quand on est soi-même connecté par SSH

Avec `-w fichier.pcap`, `tcpdump` enregistre la capture au lieu de l'afficher. On l'ouvre ensuite avec Wireshark sur un poste graphique.

En résumé

- Le réseau s'organise en couches : accès, Internet, transport, application. Une panne se trouve à un étage.
- `192.168.1.50/24` : 24 bits de réseau, 8 bits de machine, 254 adresses.
- Les plages `10.`, `172.16.` à `172.31.` et `192.168.` sont privées. Le routeur traduit les adresses à la sortie.
- Un port désigne un service. TCP garantit la livraison, UDP non.
- Le DNS traduit les noms. Sur Ubuntu, `systemd-resolved` fait le relais en `127.0.0.53`.
- `ip -br addr`, `ip route` : les adresses et les routes.
- `ping` teste la route, `mtr` montre le chemin, `dig` interroge le DNS.
- `ss -tlnp` liste ce qui écoute, `nc -zv` teste un port, `curl -I` un site.
- `tcpdump` montre les paquets quand tout le reste a échoué.

Atelier — Diagnostiquer une panne, couche par couche

Objectif : acquérir une méthode qui trouve n'importe quelle panne de réseau en six questions, puis l'exercer sur deux pannes provoquées. Comptez quarante minutes.

La méthode

On remonte les couches, du câble au service. Chaque question suppose que les précédentes ont reçu une réponse favorable.

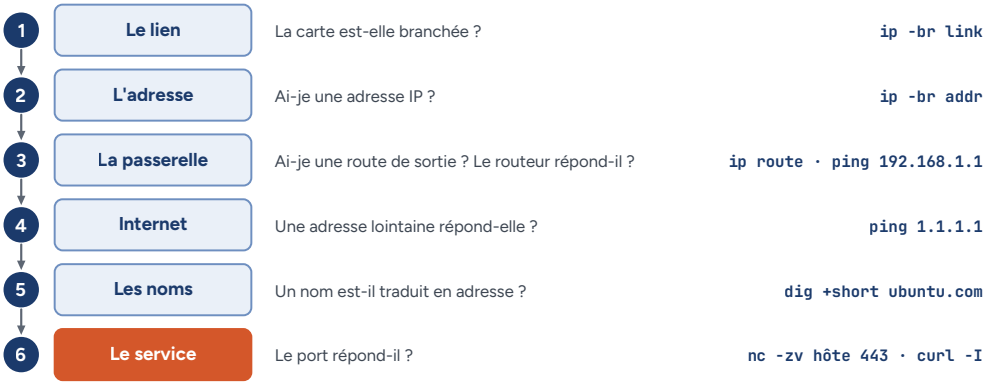


Figure 17.5 — Six questions, dans cet ordre.

1. Déroulez les six étapes sur votre machine en bon état, et notez les réponses normales : le nom de l'interface, votre adresse, celle de la passerelle, celle du serveur DNS. Le jour de la panne, vous saurez à quoi comparer.

Première panne

2. Travaillez sur la console de la machine virtuelle, ou par SSH depuis une machine du même réseau local. Supprimez la route par défaut :

```
gary@serveur:~$ sudo ip route del default
```

3. Appliquez la méthode. Le lien et l'adresse sont bons. À la troisième étape :

```
gary@serveur:~$ ip route
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.50 metric 100
gary@serveur:~$ ping -c 1 192.168.1.1
[…]
1 packets transmitted, 1 received, 0% packet loss, time 0ms
gary@serveur:~$ ping -c 1 1.1.1.1
ping: connect: Network is unreachable
```

La passerelle répond, mais il n'y a plus de ligne `default`. Le message `Network is unreachable` vient de la machine elle-même : elle ne sait pas par où envoyer le paquet, et ne l'envoie pas.

4. Réparez en demandant à Netplan de réappliquer la configuration, et vérifiez :

```
gary@serveur:~$ sudo netplan apply
gary@serveur:~$ ip route | head -n 1
default via 192.168.1.1 dev enp0s3 proto dhcp src 192.168.1.50 metric 100
```

Seconde panne

5. Désignez à `systemd-resolved` un serveur DNS qui n'existe pas :

```
gary@serveur:~$ sudo resolvectl dns enp0s3 192.168.1.99
```

6. Reprenez la méthode. Cette fois, les quatre premières étapes passent. La cinquième échoue :

```
gary@serveur:~$ ping -c 1 -q 1.1.1.1 | tail -n 2
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 45.689/45.689/45.689/0.000 ms
gary@serveur:~$ ping -c 1 ubuntu.com
ping: ubuntu.com: Temporary failure in name resolution
gary@serveur:~$ dig +short ubuntu.com @1.1.1.1
185.125.190.21
185.125.190.29
185.125.190.20
```

Internet est joignable par son adresse, et un autre résolveur répond. Le coupable est donc le serveur DNS configuré, ce que confirme `resolvectl dns enp0s3`. C'est la panne la plus fréquente, et la plus trompeuse : pour l'utilisateur, « Internet ne marche plus », alors que seul l'annuaire est en cause.

7. Annulez le réglage :

```
gary@serveur:~$ sudo resolvectl revert enp0s3
gary@serveur:~$ resolvectl dns enp0s3
Link 2 (enp0s3): 192.168.1.1
```

Vérifier

- `ping -c 1 ubuntu.com` répond de nouveau.
- Vous savez dire, devant chacun de ces messages, à quelle étape de la méthode il correspond : `Network is unreachable`, `Temporary failure in name resolution`, `Connection refused`.

Exercices

Exercice 1 — Adresses

1. Combien de machines peut contenir le réseau `10.20.0.0/22` ? Quelle est son adresse de diffusion ?
2. Les adresses `192.168.1.70` et `192.168.1.130` sont-elles dans le même réseau avec un masque `/24` ? Avec un masque `/26` ?
3. Parmi ces adresses, lesquelles sont privées : `172.20.4.1`, `172.32.4.1`, `8.8.8.8`, `10.255.0.1`, `192.169.1.1` ?
4. Écrivez sous forme abrégée l'adresse `2001:0db8:0000:0000:0000:0000:0000:0001`.

Exercice 2 — Votre machine

1. Quel est le nom de votre interface, son adresse MAC, son adresse IP et son masque ?
2. Quelle est votre passerelle ? Quelle est son adresse MAC ?
3. Votre adresse vient-elle d'un serveur DHCP ? Pour combien de temps encore ?
4. Quels ports TCP écoutent sur votre machine, et sur quelles adresses ? Lesquels sont joignables depuis le réseau ?

Exercice 3 — DNS

1. Quelles sont les adresses IPv4 et IPv6 de `debian.org` ?
2. Quels serveurs reçoivent le courrier de ce domaine ?
3. Quel est le TTL de sa fiche `A` ? Reposez la question dix secondes plus tard : qu'observez-vous ?
4. À quel nom correspond l'adresse `9.9.9.9` ?
5. Quel serveur DNS votre machine interroge-t-elle réellement ?

Exercice 4 — Ports

1. Le port 443 de `ubuntu.com` répond-il ? Et le port 22 ?
2. Que répond `nc` quand vous testez le port 8080 de votre propre machine ? Pourquoi ce message plutôt qu'un délai dépassé ?
3. Affichez les en-têtes renvoyés par `http://ubuntu.com`. Quel code obtenez-vous, et que signifie-t-il ?

Exercice 5 — Capture

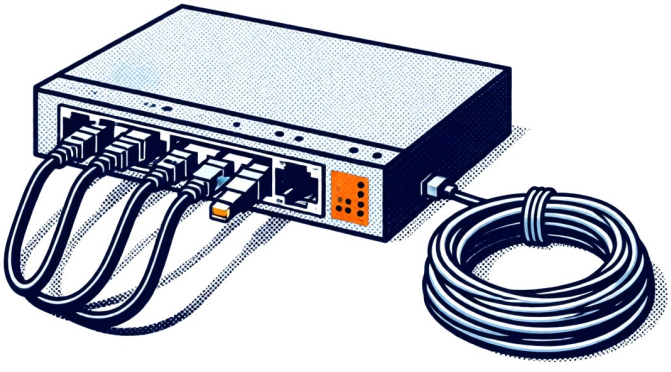
1. Lancez `tcpdump` sur le trafic ICMP, puis un `ping` dans un second terminal. Identifiez les questions et les réponses.
2. Capturez l'ouverture d'une connexion vers le port 80 d'un site. Retrouvez les trois paquets de la poignée de main.
3. Testez avec `nc` un port fermé de votre passerelle pendant une capture. Quel paquet la passerelle renvoie-t-elle ?

Pour aller plus loin

Sur la machine. `man ip` et ses pages dérivées, `man ip-address`, `man ip-route`. `man 7 tcp` et `man 7 udp` décrivent les protocoles vus par le noyau. `man pcap-filter` détaille la syntaxe des filtres de `tcpdump`.

Outils. `nmap`, du dépôt `universe`, examine les ports ouverts d'une machine ou d'un réseau entier. N'en usez que sur des machines dont vous avez la charge. `iperf3` mesure le débit réel entre deux machines. Wireshark lit les captures de `tcpdump` avec une interface graphique.

Lecture. *TCP/IP Illustrated, Volume 1*, de W. Richard Stevens, reste le livre de référence sur les protocoles. Chaque notion y est montrée par une capture.



CHAPITRE 18

Configurer le réseau sous Ubuntu

Le chapitre précédent a montré comment lire l'état du réseau. Celui-ci montre comment l'écrire, dans un fichier de quelques lignes, et comment l'appliquer à un serveur distant sans risquer de s'en couper.

DANS CE CHAPITRE

- Dire qui configure le réseau sur Ubuntu Server et sur Ubuntu Desktop.
- Lire un fichier Netplan et ce qu'il produit.
- Donner une adresse fixe à un serveur.
- Appliquer un changement avec `netplan try`, qui revient en arrière si la connexion est perdue.
- Choisir les serveurs DNS d'une machine.
- Nommer une machine avec `hostnamectl`.
- Régler le réseau d'un poste avec `nmcli`.

AVANT DE COMMENCER

Le chapitre 17. Les manipulations se font sur le serveur, connecté par SSH depuis une autre machine du réseau local : c'est précisément la situation où une erreur coupe l'accès. Prenez un instantané de la machine virtuelle. Gardez sa console à portée de main.

18.1 Qui fait quoi

Sur Ubuntu, la configuration du réseau passe par trois logiciels.

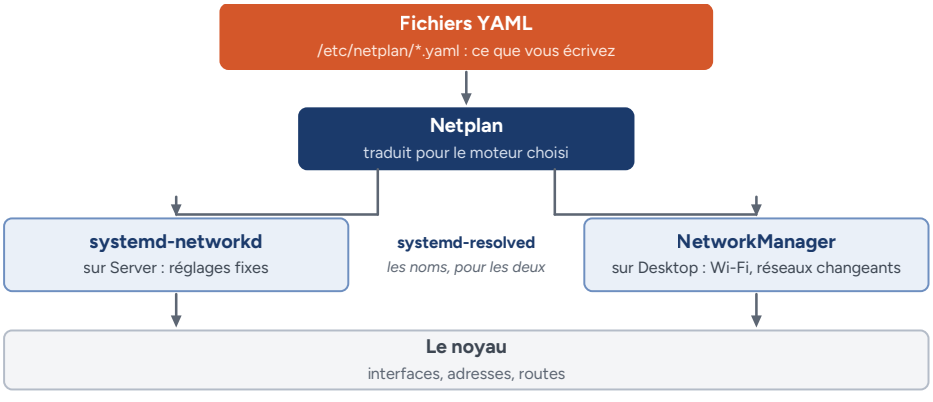


Figure 18.1 — De vos fichiers au noyau.

Netplan est le point d'entrée. Vous décrivez le réseau voulu dans un fichier YAML, sous `/etc/netplan/`. Netplan ne configure rien lui-même. Il traduit votre description pour un moteur, qui fait le travail.

Deux moteurs existent. **systemd-networkd** équipe Ubuntu Server : il applique des réglages fixes, sans interface. **NetworkManager** équipe Ubuntu Desktop : il gère le Wi-Fi, les réseaux qu'on change en se déplaçant, les VPN, et s'accompagne des réglages graphiques de GNOME.

systemd-resolved, rencontré au chapitre 17, s'occupe des noms dans les deux cas.

```
gary@serveur:~$ systemctl is-active systemd-networkd NetworkManager
active
inactive
```

Netplan a remplacé l'ancien fichier `/etc/network/interfaces` depuis Ubuntu 17.10. Les tutoriels qui vous font modifier ce fichier sont périmés.

18.2 Lire la configuration en place

Voici ce que l'installateur a écrit sur le serveur :

```
gary@serveur:~$ ls -l /etc/netplan/
total 4
-rw------- 1 root root 391 Oct  4 22:32 50-cloud-init.yaml
gary@serveur:~$ sudo netplan get
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
```

Quatre lignes utiles : une carte Ethernet, `enp0s3`, qui demande son adresse à un serveur DHCP. `netplan get` affiche la configuration telle que Netplan la comprend, tous fichiers réunis.

`netplan status` montre le résultat : ce que la machine a effectivement obtenu.

```
gary@serveur:~$ sudo netplan status
     Online state: online
    DNS Addresses: 127.0.0.53 (stub)
       DNS Search: maison.lan

●  2: enp0s3 ethernet UP (networkd: enp0s3)
      MAC Address: 52:54:00:12:34:56 (Red Hat, Inc.)
        Addresses: 192.168.1.50/24 (dynamic, dhcp)
                   fe80::5054:ff:fe12:3456/64 (link)
    DNS Addresses: 192.168.1.1
       DNS Search: maison.lan
           Routes: default via 192.168.1.1 from 192.168.1.50 metric 100 (dhcp)
                   192.168.1.0/24 from 192.168.1.50 metric 100 (link)
[…]
```

L'adresse, la passerelle et le serveur DNS viennent tous du DHCP. C'est confortable, et gênant pour un serveur : on veut le trouver toujours à la même adresse.

18.3 Une adresse fixe

Deux façons d'y parvenir. La première se règle sur le routeur : on y réserve une adresse pour la carte du serveur, qui continue de la demander par DHCP. C'est la plus simple quand on a la main sur le routeur. La seconde se règle sur le serveur : on y écrit l'adresse en dur. C'est celle des serveurs hébergés, et celle que nous allons suivre.

On ne touche pas au fichier de l'installateur. Netplan lit tous les fichiers de `/etc/netplan/` dans l'ordre alphabétique, et les suivants complètent ou remplacent les précédents. Nous créons donc `/etc/netplan/60-adresse-fixe.yaml` :

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: false
      addresses:
        - 192.168.1.50/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses: [192.168.1.1, 1.1.1.1]
        search: [maison.lan]
```

Clé	Rôle
`dhcp4: false`	Ne plus demander d'adresse au DHCP
`addresses`	L'adresse de la machine, avec son masque
`routes`	La route par défaut : `to: default`, puis la passerelle après `via`
`nameservers, addresses`	Les serveurs DNS à interroger
`nameservers, search`	Le domaine ajouté aux noms courts : `vostro` devient `vostro.maison.lan`

Choisissez une adresse hors de la plage que distribue votre routeur, sinon il finira par la donner à une autre machine.

> **ATTENTION · Un serveur DNS local, ou des serveurs publics, pas les deux**
>
> Ce fichier déclare le routeur de la maison, qui connaît les noms en `maison.lan`, puis 1.1.1.1, qui ne les connaît pas. systemd-resolved tient les deux pour interchangeables : après une seule absence de réponse du premier, il passe au second et y reste. Les noms locaux cessent alors de se résoudre, sans le moindre message. C'est arrivé au serveur de ce livre, et le chapitre 30 raconte le diagnostic. Si vous utilisez des noms locaux, ne déclarez que le serveur local : il transmet lui-même les autres demandes vers Internet.

ATTENTION · YAML ne pardonne pas l'indentation

Dans un fichier YAML, la structure vient des retraits. Chaque niveau se décale de deux espaces, jamais d'une tabulation. Une ligne mal alignée change le sens du fichier ou le rend illisible. Netplan signale l'endroit :

```
/etc/netplan/65-essai.yaml:5:1: Invalid YAML: tabs are not allowed for indent:
/etc/netplan/65-essai.yaml:5:7: Error in network definition: unknown key
'adresses'
```

La première erreur est une tabulation, la seconde un mot-clé mal orthographié. Les deux nombres donnent la ligne et la colonne.

Le fichier doit n'être lisible que par `root` : il peut contenir un mot de passe Wi-Fi. `sudo chmod 600` s'en charge. À défaut, Netplan proteste à chaque commande :

```
WARNING **: Permissions for /etc/netplan/60-adresse-fixe.yaml are too open.
Netplan configuration should NOT be accessible by others.
```

18.4 Appliquer sans se couper du serveur

Vous êtes connecté par SSH à un serveur situé à l'autre bout du pays. Vous modifiez son adresse, vous appliquez, et une faute de frappe dans la passerelle le rend injoignable. Il ne reste qu'à téléphoner à quelqu'un qui soit sur place.

`netplan try` existe pour cette situation. Il applique la nouvelle configuration, puis attend une confirmation. Si elle ne vient pas, il remet l'ancienne.

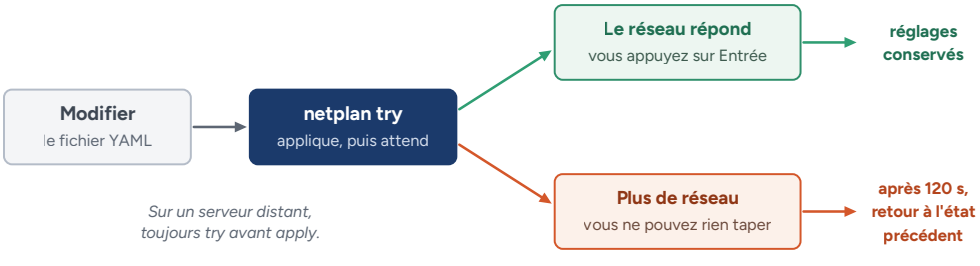


Figure 18.2 — Le filet de sécurité de netplan try.

Essayons d'abord avec le bon fichier :

```
gary@serveur:~$ sudo netplan try
Do you want to keep these settings?


Press ENTER before the timeout to accept the new configuration


Changes will revert in 120 seconds
Configuration accepted.
```

La connexion SSH a tenu, on a pu appuyer sur Entrée : les réglages sont gardés. Vérifions :

```
gary@serveur:~$ ip addr show enp0s3 | grep 'inet '
    inet 192.168.1.50/24 brd 192.168.1.255 scope global enp0s3
gary@serveur:~$ ip route
default via 192.168.1.1 dev enp0s3 proto static
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.50
gary@serveur:~$ resolvectl dns enp0s3
Link 2 (enp0s3): 192.168.1.1 1.1.1.1
```

Le mot `dynamic` a disparu de la ligne `inet`, et la route par défaut est marquée `static`.

Maintenant, l'erreur. Remplaçons dans le fichier `192.168.1` par `192.168.2`, pour l'adresse comme pour la passerelle, et relançons avec un délai raccourci :

```
gary@serveur:~$ sudo netplan try --timeout 15
Do you want to keep these settings?


Press ENTER before the timeout to accept the new configuration


Changes will revert in 15 seconds
Reverting.
gary@serveur:~$ ip addr show enp0s3 | grep 'inet '
    inet 192.168.1.50/24 brd 192.168.1.255 scope global enp0s3
```

Pendant quinze secondes, le terminal est resté figé : le serveur avait changé de réseau et ne nous entendait plus. Puis Netplan a annulé, et la session a repris comme si de rien n'était. Le fichier fautif, lui, est toujours sur le disque. Il faut le corriger avant le prochain redémarrage.

Commande	Effet
`sudo netplan generate`	Vérifier la syntaxe, sans rien appliquer
`sudo netplan try`	Appliquer, avec retour en arrière faute de confirmation
`sudo netplan apply`	Appliquer pour de bon
`sudo netplan get`	Afficher la configuration réunie
`sudo netplan status`	Afficher l'état obtenu

À RETENIR · try, toujours

Sur une machine que vous ne pouvez pas atteindre physiquement, n'utilisez jamais `netplan apply` pour un changement d'adresse, de passerelle ou d'interface. `netplan try` coûte une pression sur Entrée, et évite un déplacement.

NOTE · Dans les coulisses

Netplan écrit sa traduction dans `/run/systemd/network/`. Ce dossier est recréé à chaque démarrage, à partir de `/etc/netplan/`. On y lit ce que systemd-networkd a réellement reçu :

```
[Match]
Name=enp0s3

[Network]
Address=192.168.1.50/24
DNS=192.168.1.1
DNS=1.1.1.1
Domains=maison.lan

[Route]
Destination=0.0.0.0/0
Gateway=192.168.1.1
```

Ne modifiez jamais ces fichiers. Tout changement se fait dans `/etc/netplan/`.

18.5 Les serveurs DNS

La clé `nameservers` du fichier Netplan fixe les serveurs DNS d'une interface. Avec le DHCP, ils viennent du routeur, mais on peut imposer les siens :

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
      dhcp4-overrides:
        use-dns: false
      nameservers:
        addresses: [1.1.1.1, 9.9.9.9]
```

On garde ici l'adresse fournie par le DHCP, en refusant ses serveurs DNS.

Vérifiez toujours avec `resolvectl status` ou `resolvectl dns enp0s3`. Le fichier `/etc/resolv.conf` ne vous apprendra rien : il ne contient que l'adresse du relais, `127.0.0.53`.

ATTENTION · Un vieux mot-clé

Beaucoup d'exemples en ligne écrivent la passerelle avec `gateway4`. Netplan l'accepte encore, en avertissant :

```
`gateway4` has been deprecated, use default routes instead.
```

Écrivez la route par défaut sous la forme `routes`, comme dans notre fichier.

18.6 Le nom de la machine

`hostnamectl` affiche et modifie le nom de la machine :

```
gary@serveur:~$ hostnamectl
 Static hostname: serveur
       Icon name: computer-vm
         Chassis: vm
      Machine ID: 2df357adc3184a56b82e1e62d04f5472
         Boot ID: 0af9a591d96843a8878a6252d8967d15
  Virtualization: kvm
Operating System: Ubuntu 24.04.5 LTS
          Kernel: Linux 6.8.0-142-generic
    Architecture: x86-64
[…]
```

`sudo hostnamectl set-hostname web1` change le nom sur-le-champ et l'inscrit dans `/etc/hostname`. Il reste à corriger à la main la ligne de `/etc/hosts` qui com-

mence par `127.0.1.1`. Elle associe le nom de la machine à une adresse locale, et des programmes s'y fient pour se nommer eux-mêmes.

Donnez aux serveurs des noms qui disent leur rôle, courts, en minuscules et sans accent : `web1`, `sauvegarde`, `base-compta`. Ce nom apparaît dans l'invite de chaque terminal. Il vous évitera un jour de taper une commande sur la mauvaise machine.

18.7 Sur un poste de travail : NetworkManager

Sur Ubuntu Desktop, le seul fichier que vous trouverez d'abord dans `/etc/netplan/` tient en trois lignes :

```
gary@vostro:~$ cat /etc/netplan/01-network-manager-all.yaml
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: NetworkManager
```

Tout est délégué à NetworkManager. On le pilote par les réglages de GNOME, ou par la commande `nmcli` :

```
gary@vostro:~$ nmcli device status
DEVICE           TYPE      STATE
wlp2s0           wifi      connecté
[…]
```

Commande	Effet
`nmcli device status`	L'état des interfaces
`nmcli connection show`	Les connexions enregistrées
`nmcli device wifi list`	Les réseaux Wi-Fi à portée
`nmcli device wifi connect NOM --ask`	Se connecter à un réseau Wi-Fi, en saisissant le mot de passe
`nmcli connection up NOM`, `down NOM`	Activer, désactiver une connexion
`nmcli connection modify NOM ipv4.dns "1.1.1.1"`	Modifier un réglage
`nmtui`	Les mêmes opérations dans des menus en mode texte

Une *connexion*, pour NetworkManager, est un jeu de réglages enregistré : un par réseau Wi-Fi fréquenté, un pour le câble. Une adresse fixe se déclare ainsi :

```
nmcli connection add type ethernet con-name bureau ifname enp3s0 \
    ipv4.method manual ipv4.addresses 192.168.1.60/24 \
    ipv4.gateway 192.168.1.1 ipv4.dns 192.168.1.1
```

Sur Ubuntu 24.04, NetworkManager range ses connexions dans `/etc/netplan/`, sous des noms en `90-NM-….yaml`. Netplan reste ainsi le lieu unique de la configuration, sur un poste comme sur un serveur.

`nmtui` rend service dans un cas précis : un poste ou un serveur équipé de NetworkManager, dont l'interface graphique ne démarre plus.

18.8 Au-delà d'une carte

Netplan décrit aussi des montages plus élaborés. Les trois plus courants :

Un **pont** (*bridge*) relie des machines virtuelles ou des conteneurs au réseau local, comme s'ils y étaient branchés. La carte physique devient un port du pont, et c'est le pont qui porte l'adresse :

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: false
  bridges:
    br0:
      interfaces: [enp0s3]
      dhcp4: true
```

Un **agrégat** (*bond*) réunit deux cartes en une seule interface. Si un câble ou un commutateur lâche, l'autre carte prend le relais :

```
network:
  version: 2
  ethernets:
    enp1s0: {}
    enp2s0: {}
  bonds:
    bond0:
      interfaces: [enp1s0, enp2s0]
```

```
      parameters:
        mode: active-backup
      dhcp4: true
```

Le **Wi-Fi** se déclare sous `wifis`. C'est utile sur une machine sans écran, un Raspberry Pi par exemple :

```
network:
  version: 2
  wifis:
    wlan0:
      dhcp4: true
      access-points:
        "NomDuReseau":
          password: "phrase-secrete-du-wifi"
```

Ces trois fichiers ont été validés par `netplan generate`. Avant d'en appliquer un, relisez les noms d'interfaces, qui sont ceux de votre machine, et passez par `netplan try`.

En résumé

- Netplan lit des fichiers YAML dans `/etc/netplan/` et les traduit pour systemd-networkd sur Server, pour NetworkManager sur Desktop.
- `netplan get` montre la configuration, `netplan status` le résultat.
- On ajoute son propre fichier plutôt que de modifier celui de l'installateur.
- Une adresse fixe : `dhcp4: false`, `addresses`, `routes`, `nameservers`.
- Des espaces, jamais de tabulations. Droits `600`.
- `netplan try` annule tout seul si la connexion est perdue.
- `resolvectl status` dit quels serveurs DNS sont utilisés.
- `hostnamectl set-hostname` nomme la machine. `/etc/hosts` se corrige à la main.
- Sur Desktop, `nmcli` et `nmtui` pilotent NetworkManager.

Atelier — Passer le serveur en adresse fixe

Objectif : remplacer l'adresse reçue par DHCP par une adresse fixe, en travaillant par SSH comme sur un serveur distant, et constater que `netplan try` vous protège d'une erreur. Comptez trente minutes.

Relever l'existant

1. Connectez-vous au serveur par SSH. Notez ce que donnent `ip -br addr`, `ip route` et `resolvectl dns` : le nom de l'interface, l'adresse et son masque, la passerelle, le serveur DNS.
2. Copiez le dossier de configuration, par précaution :

```
sudo cp -a /etc/netplan /etc/netplan.avant
```

Écrire

3. Créez `/etc/netplan/60-adresse-fixe.yaml` sur le modèle du chapitre, avec vos propres valeurs : le nom de votre interface, l'adresse relevée, votre passerelle, votre serveur DNS. Puis protégez-le et vérifiez-le :

```
sudo chmod 600 /etc/netplan/60-adresse-fixe.yaml
sudo netplan generate
```

Aucun message : la syntaxe est bonne.

Appliquer

4. Lancez `sudo netplan try`. Si le compte à rebours s'affiche et que votre terminal répond, appuyez sur Entrée.
5. Vérifiez : `ip addr show` ne porte plus le mot `dynamic`, `ip route` montre une route `static`, et `ping -c 1 ubuntu.com` répond.

Se tromper exprès

6. Modifiez la passerelle dans le fichier pour une adresse qui n'existe pas sur votre réseau, puis lancez `sudo netplan try --timeout 20`. N'appuyez sur rien. Dans un second terminal, sur le serveur, constatez pendant ce temps que `ping -c 1 1.1.1.1` échoue. Après vingt secondes, Netplan affiche `Reverting.` et le réseau revient.
7. Remettez la bonne passerelle dans le fichier, et confirmez avec `sudo netplan generate`.

Nommer

8. Donnez à la machine un nom qui dise son rôle, corrigez `/etc/hosts`, et ouvrez une nouvelle session pour voir l'invite changer :

```
sudo hostnamectl set-hostname web1
sudo nano /etc/hosts
```

Vérifier

- Après `sudo reboot`, le serveur revient à la même adresse.
- `sudo netplan status` n'affiche plus `dhcp` à côté de l'adresse.
- `hostnamectl hostname`, `cat /etc/hostname` et l'invite de votre terminal donnent le même nom.

Exercices

Exercice 1 — Lire

Sur votre serveur :

1. Quel moteur configure le réseau ? Comment le vérifiez-vous ?
2. Quels fichiers Netplan lit-il, et dans quel ordre ?
3. Affichez la configuration réunie, puis l'état obtenu. Quelle information ne figure que dans le second ?

Exercice 2 — Écrire

Écrivez le fichier Netplan d'un serveur qui a :

1. l'interface `ens18` ;
2. l'adresse `10.0.5.20` dans un réseau de 254 machines ;
3. la passerelle `10.0.5.1` ;
4. les serveurs DNS `10.0.5.2` et `9.9.9.9`, et le domaine de recherche `bureau.lan`.

Vérifiez-le sans l'appliquer, dans un dossier à part, avec `sudo netplan generate --root-dir`.

Exercice 3 — Corriger

Ce fichier contient trois erreurs. Trouvez-les sans l'aide de Netplan, puis vérifiez.

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: false
      adresses: [192.168.1.50]
      routes:
        - to: default
            via: 192.168.1.1
```

Exercice 4 — DNS

1. Faites en sorte que le serveur garde son adresse DHCP, mais interroge `9.9.9.9` au lieu du serveur DNS annoncé par le routeur.
2. Appliquez avec `netplan try` et vérifiez avec `resolvectl`.
3. Que contient `/etc/resolv.conf` avant et après ? Pourquoi ?

Exercice 5 — Poste de travail

Sur une machine Ubuntu Desktop :

1. Listez les connexions enregistrées par NetworkManager.
2. Affichez l'adresse, la passerelle et les serveurs DNS de la connexion active avec `nmcli device show`.
3. Où NetworkManager a-t-il rangé le mot de passe de votre réseau Wi-Fi ? Qui peut lire ce fichier ?

Pour aller plus loin

Sur la machine. `man netplan` décrit toutes les clés du format YAML. Le dossier `/usr/share/doc/netplan/examples/` contient une trentaine de fichiers d'exemple : adresses multiples, VLAN, ponts, agrégats, WireGuard. `man networkctl` et `man nmcli` documentent les deux moteurs.

En ligne. `netplan.readthedocs.io` reprend ces exemples avec des explications.

La suite. Le serveur a une adresse stable et un nom. Il reste à s'y connecter proprement : c'est l'objet du chapitre 19, consacré à SSH.



CHAPITRE 19

SSH

SSH est la porte par laquelle on administre un serveur. Bien réglée, elle ne s'ouvre qu'à vos clés. Ce chapitre l'installe, la verrouille, et montre ce qu'on y fait passer : des commandes, des fichiers, et d'autres connexions.

DANS CE CHAPITRE

- Se connecter à un serveur et vérifier qu'il est bien celui qu'on croit.
- Créer une paire de clés et l'installer sur un serveur.
- Utiliser un agent pour ne saisir sa phrase secrète qu'une fois.
- Simplifier ses connexions avec `~/.ssh/config`.
- Durcir le serveur SSH : clés seulement, pas de `root`, accès réservé à un groupe.
- Copier des fichiers avec `scp`, `sftp` et `rsync`.
- Atteindre un service interne par un tunnel, et une machine cachée par un rebond.
- Ajouter un second facteur d'authentification.

AVANT DE COMMENCER

Les chapitres 6, 17 et 18. Il faut deux machines : le serveur, à l'adresse 192.168.1.50, et un poste d'où l'on se connecte, ici `vostro`. OpenSSH est installé d'origine sur Ubuntu Server si vous avez coché la case à l'installation. Sinon : `sudo apt install openssh-server`. Gardez la console du serveur à portée de main pendant tout le chapitre : on y règle la porte d'entrée.

19.1 Ce que fait SSH

REPÈRE HISTORIQUE · Un mot de passe intercepté

En 1995, à l'université de technologie d'Helsinki, un programme espion capture des mots de passe qui circulent sur le réseau. On administrait alors les machines avec `telnet` et `rlogin`, qui transmettent tout en clair. Un chercheur de l'université, Tatu Ylönen, écrit en réponse un remplaçant chiffré : SSH, *Secure Shell*. La version libre que nous utilisons tous, OpenSSH, est née en 1999 au sein du projet OpenBSD.

SSH ouvre un terminal sur une machine distante, à travers un réseau qu'on suppose hostile. Il garantit trois choses. Personne ne peut lire ce qui passe. Personne ne peut le modifier en route. Et chacun des deux bouts sait à qui il parle.

Deux programmes se répondent : le client, `ssh`, sur votre poste, et le serveur, `sshd`, qui écoute sur le port 22 de la machine distante.

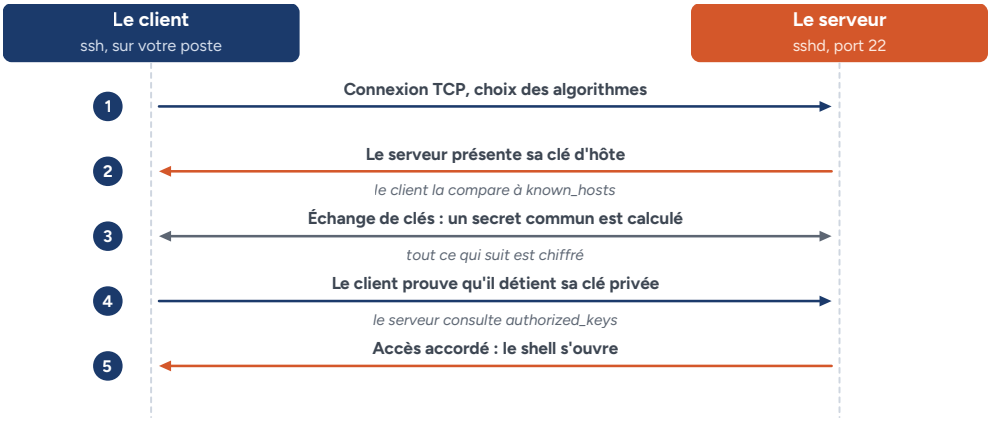


Figure 19.1 — Les étapes d'une connexion.

Deux vérifications d'identité ont lieu, une dans chaque sens. Le serveur prouve la sienne avec sa **clé d'hôte**. L'utilisateur prouve la sienne avec un mot de passe ou, mieux, avec sa propre clé. Tout le chapitre tourne autour de ces deux preuves.

19.2 La première connexion

```
gary@vostro:~$ ssh gary@192.168.1.50
The authenticity of host '192.168.1.50 (192.168.1.50)' can't be established.
ED25519 key fingerprint is SHA256:woXoUDeWvs/+OIB0C2rbhmFkywQTzX94Fz2CkPrxkEs.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.50' (ED25519) to the list of known hosts.
gary@192.168.1.50's password:
Welcome to Ubuntu 24.04.5 LTS (GNU/Linux 6.8.0-142-generic x86_64)
[…]
gary@serveur:~$
```

La syntaxe est `ssh utilisateur@machine`. Si le nom d'utilisateur est le même des deux côtés, on peut l'omettre.

La question posée n'est pas une formalité. Votre poste ne connaît pas encore ce serveur, et rien ne prouve que la machine qui répond est la bonne. SSH affiche l'**empreinte** de sa clé d'hôte et vous demande de la valider. Tout le monde tape `yes` sans regarder. La bonne pratique consiste à comparer cette empreinte avec celle qu'on lit sur le serveur lui-même, depuis sa console :

```
gary@serveur:~$ ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub
256 SHA256:woXoUDeWvs/+OIB0C2rbhmFkywQTzX94Fz2CkPrxkEs root@serveur (ED25519)
```

Les deux empreintes sont identiques : c'est bien notre serveur. La clé est alors enregistrée dans `~/.ssh/known_hosts`, et la question ne sera plus posée.

Si un jour la clé présentée ne correspond plus à celle qui est enregistrée, SSH refuse de se connecter :

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
[…]
Host key verification failed.
```

Deux explications sont possibles. Le serveur a été réinstallé, ou une autre machine a pris son adresse : c'est le cas habituel. Ou bien quelqu'un s'est placé entre vous et lui. Ne passez outre que si vous connaissez la raison du changement. `ssh-keygen -R 192.168.1.50` retire alors l'ancienne clé, et la connexion suivante reposera la question.

19.3 Les clés

Un mot de passe se devine, se réutilise et s'intercepte par ruse. Les robots qui parcourent Internet en essaient des milliers par jour sur chaque serveur SSH qu'ils trouvent. La parade est de ne plus accepter de mots de passe du tout, et d'ouvrir la porte avec des clés.

Une **paire de clés** se compose de deux fichiers liés par les mathématiques. La **clé privée** reste sur votre poste et n'en sort jamais. La **clé publique** se distribue sans précaution : on la dépose sur chaque serveur où l'on veut entrer. À la connexion, le serveur soumet au client une épreuve que seule la clé privée permet de réussir. La clé privée ne voyage pas, et il n'y a rien à intercepter.

Créer sa paire de clés

```
gary@vostro:~$ ssh-keygen -t ed25519 -C "gary@vostro"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/gary/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/gary/.ssh/id_ed25519
Your public key has been saved in /home/gary/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:+q6P8FjARTH006Xk11XHQ8MVH/ihlQG8B7vXrKby594 gary@vostro
[…]
```

`-t ed25519` choisit l'algorithme. Ed25519 donne des clés courtes, rapides et sûres, et tous les serveurs l'acceptent depuis la version 6.5 d'OpenSSH. `-C` ajoute un commentaire, qui servira à reconnaître la clé parmi d'autres.

À la première question, Entrée accepte l'emplacement proposé. La seconde demande une **phrase secrète**. Elle chiffre la clé privée sur le disque : celui qui vole le fichier ne peut rien en faire sans elle. Donnez-en toujours une.

```
gary@vostro:~$ ls -l ~/.ssh
total 16
-rw------- 1 gary gary 444 Oct  4 22:41 id_ed25519
```

```
-rw-r--r-- 1 gary gary  93 Oct  4 22:41 id_ed25519.pub
-rw------- 1 gary gary 978 Oct  4 22:41 known_hosts
gary@vostro:~$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPgNVDIjKQsmt8dDpBT49TVl5I8/33ZXLd3lbTojso7Y gary@vostro
```

Fichier	Contenu	À protéger ?
`~/.ssh/id_ed25519`	Votre clé privée	Oui. Ne la copiez nulle part, ne l'envoyez à personne
`~/.ssh/id_ed25519.pub`	Votre clé publique	Non. C'est elle qu'on distribue
`~/.ssh/known_hosts`	Les clés d'hôte des serveurs déjà rencontrés	Non
`~/.ssh/config`	Vos raccourcis de connexion	Non

Installer la clé sur le serveur

`ssh-copy-id` dépose la clé publique au bon endroit. Il demande le mot de passe, pour la dernière fois :

```
gary@vostro:~$ ssh-copy-id gary@192.168.1.50
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/
gary/.ssh/id_ed25519.pub"
[…]
gary@192.168.1.50's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'gary@192.168.1.50'"
and check to make sure that only the key(s) you wanted were added.
```

Sur le serveur, la clé a été ajoutée à `~/.ssh/authorized_keys`. Ce fichier contient une clé publique par ligne. Qui détient l'une des clés privées correspondantes peut ouvrir une session sous ce compte.

```
gary@vostro:~$ ssh gary@192.168.1.50
Enter passphrase for key '/home/gary/.ssh/id_ed25519':
[…]
gary@serveur:~$ cat ~/.ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPgNVDIjKQsmt8dDpBT49TVl5I8/33ZXLd3lbTojso7Y gary@vostro
```

Le serveur ne demande plus le mot de passe du compte. C'est le client qui demande la phrase secrète, pour déverrouiller la clé privée. Elle ne quitte pas votre poste.

ATTENTION · Des droits trop larges, et la clé est refusée

`sshd` ignore un fichier `authorized_keys` que d'autres utilisateurs pourraient modifier. La connexion retombe alors sur le mot de passe, sans explication côté client. Le journal du serveur, lui, le dit :

```
Authentication refused: bad ownership or modes for file /home/gary/.ssh/
authorized_keys
```

Les bons droits : `700` pour le dossier `~/.ssh`, `600` pour `authorized_keys`, et tous deux à votre nom. Devant une clé qui « ne marche pas », `journalctl -u ssh` sur le serveur est le premier endroit où regarder.

L'agent

Taper la phrase secrète à chaque connexion lasse vite. L'**agent** garde la clé déverrouillée en mémoire pour la durée de la session :

```
gary@vostro:~$ eval "$(ssh-agent -s)"
Agent pid 413
gary@vostro:~$ ssh-add
Enter passphrase for /home/gary/.ssh/id_ed25519:
Identity added: /home/gary/.ssh/id_ed25519 (gary@vostro)
gary@vostro:~$ ssh gary@192.168.1.50 hostname
serveur
```

La dernière commande n'a rien demandé. Elle montre aussi une autre façon d'employer `ssh` : suivi d'une commande, il l'exécute sur le serveur, affiche le résultat et referme la connexion.

Sur Ubuntu Desktop, il n'y a rien à lancer. La session GNOME fournit un agent, et propose de retenir la phrase secrète à la première utilisation.

19.4 Le fichier de configuration du client

Adresses, noms de comptes, ports : dès qu'on gère trois serveurs, la mémoire ne suffit plus. Le fichier `~/.ssh/config` donne un nom court à chaque destination.

```
Host serveur
    HostName 192.168.1.50
    User gary
    IdentityFile ~/.ssh/id_ed25519
```

```
Host *
    ServerAliveInterval 60
```

```
gary@vostro:~$ ssh serveur
[…]
gary@serveur:~$
```

`scp`, `sftp`, `rsync` et tous les outils qui s'appuient sur SSH lisent le même fichier. Le bloc `Host *` s'applique à toutes les connexions. Ici, `ServerAliveInterval 60` envoie un signe de vie chaque minute, ce qui empêche les routeurs de couper une session restée muette.

Directive	Rôle
`HostName`	L'adresse ou le nom réel de la machine
`User`	Le compte à utiliser
`Port`	Le port, s'il n'est pas 22
`IdentityFile`	La clé privée à présenter
`ProxyJump`	La machine par laquelle passer (voir plus bas)
`LocalForward`	Un tunnel à ouvrir à chaque connexion

`ssh -G serveur` affiche les réglages qui seront réellement appliqués à une destination, et `ssh -v serveur` détaille la connexion étape par étape :

```
debug1: Connecting to 192.168.1.50 [192.168.1.50] port 22.
debug1: Host '192.168.1.50' is known and matches the ED25519 host key.
debug1: Authentications that can continue: publickey
debug1: Offering public key: /home/gary/.ssh/id_ed25519 ED25519 SHA256:+q6P8Fj…
debug1: Server accepts key: /home/gary/.ssh/id_ed25519 ED25519 SHA256:+q6P8Fj…
```

19.5 Régler le serveur

Maintenant que la clé fonctionne, on peut fermer les autres entrées.

La configuration de `sshd` se trouve dans `/etc/ssh/sshd_config`. Ce fichier commence par inclure tous ceux du dossier `/etc/ssh/sshd_config.d/`. On laisse le fichier principal intact, et on écrit ses réglages dans ce dossier.

`sshd -T` affiche la configuration effective, tous fichiers réunis. Avant de rien changer :

```
gary@serveur:~$ sudo sshd -T | \
    grep -E '^(permitrootlogin|passwordauthentication|maxauthtries|x11forwarding) '
maxauthtries 6
permitrootlogin without-password
passwordauthentication yes
x11forwarding yes
```

Le premier qui parle a raison

Pour chaque directive, `sshd` retient la première valeur qu'il lit et ignore les suivantes. Or l'installateur a pu déposer un fichier dans le dossier :

```
gary@serveur:~$ ls /etc/ssh/sshd_config.d/
50-cloud-init.conf
gary@serveur:~$ sudo cat /etc/ssh/sshd_config.d/50-cloud-init.conf
PasswordAuthentication yes
```

Créons notre fichier sous un nom qui se classe après lui, et regardons :

```
gary@serveur:~$ echo 'PasswordAuthentication no' | \
    sudo tee /etc/ssh/sshd_config.d/99-durcissement.conf
PasswordAuthentication no
gary@serveur:~$ sudo sshd -T | grep ^passwordauthentication
passwordauthentication yes
```

Notre réglage est ignoré : `50-cloud-init.conf` a été lu avant `99-durcissement.conf`. C'est l'inverse de ce que font Netplan et systemd, où le dernier fichier l'emporte. Beaucoup d'administrateurs croient avoir interdit les mots de passe et ne l'ont pas fait. Deux leçons : donnez à votre fichier un nom qui passe en premier, et vérifiez toujours avec `sshd -T`.

Le fichier de durcissement

Voici `/etc/ssh/sshd_config.d/10-durcissement.conf` :

```
# Connexion par clé uniquement, jamais en root, réservée à un groupe
PasswordAuthentication no
PermitRootLogin no
AllowGroups acces-ssh
MaxAuthTries 3
X11Forwarding no
```

Directive	Effet
`PasswordAuthentication no`	Refuser les mots de passe : seules les clés ouvrent la porte
`PermitRootLogin no`	Interdire toute connexion directe sous `root`
`AllowGroups acces-ssh`	N'accepter que les membres de ce groupe
`MaxAuthTries 3`	Couper après trois essais manqués
`X11Forwarding no`	Ne pas transporter d'applications graphiques

Le groupe se crée d'abord, et on y inscrit les comptes qui ont le droit d'entrer :

```
gary@serveur:~$ sudo groupadd acces-ssh
gary@serveur:~$ sudo usermod -aG acces-ssh gary
```

Avec `AllowGroups`, donner ou retirer l'accès SSH à quelqu'un devient une simple affaire de groupe, sans toucher à la configuration.

Appliquer sans s'enfermer dehors

Une erreur dans ce fichier peut vous interdire l'accès à votre propre serveur. Trois précautions rendent l'opération sûre.

Vérifier la syntaxe. `sshd -t` relit la configuration et ne dit rien si elle est correcte. Avec une faute de frappe :

```
gary@serveur:~$ sudo sshd -t
/etc/ssh/sshd_config.d/15-faute.conf: line 1: Bad configuration option:
PasswordAuthentification
/etc/ssh/sshd_config.d/15-faute.conf: terminating, 1 bad configuration options
```

Garder une session ouverte. Recharger `sshd` ne coupe pas les sessions en cours. Tant que vous en tenez une, vous pouvez corriger.

Essayer depuis un autre terminal avant de fermer la première.

```
gary@serveur:~$ sudo sshd -t
gary@serveur:~$ sudo systemctl reload ssh
gary@serveur:~$ sudo sshd -T | \
    grep -E '^(permitrootlogin|passwordauthentication|maxauthtries|allowgroups) '
maxauthtries 3
permitrootlogin no
passwordauthentication no
allowgroups acces-ssh
```

Puis, depuis le poste, dans un second terminal :

```
gary@vostro:~$ ssh serveur hostname
serveur
gary@vostro:~$ ssh -o PubkeyAuthentication=no serveur
gary@192.168.1.50: Permission denied (publickey).
gary@vostro:~$ ssh root@192.168.1.50
root@192.168.1.50: Permission denied (publickey).
```

La clé ouvre toujours. Sans clé, le serveur ne propose même plus de saisir un mot de passe : `(publickey)` est la seule méthode qu'il annonce.

À RETENIR · La clé d'abord, le verrou ensuite

N'interdisez les mots de passe qu'après avoir vérifié que votre clé fonctionne, depuis chaque poste dont vous aurez besoin. Et conservez un autre chemin vers la machine : sa console, ou celle que fournit l'hébergeur.

SUR LE TERRAIN · Changer de port sur Ubuntu 24.04

Déplacer SSH sur un autre port ne le protège pas, mais allège les journaux des tentatives des robots. Sur Ubuntu 24.04, la directive `Port` ne suffit plus. Ce n'est pas `sshd` qui écoute, mais systemd, par l'unité `ssh.socket` du chapitre 12. Après avoir écrit `Port 2222` dans un fichier du dossier, `sudo systemctl restart ssh` ne change rien :

```
gary@serveur:~$ sudo ss -tln | grep -E ':(22|2222) '
LISTEN 0      4096         0.0.0.0:22        0.0.0.0:*
```

Il faut faire régénérer l'unité, puis la relancer :

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl restart ssh.socket
gary@serveur:~$ sudo ss -tln | grep -E ':(22|2222) '
LISTEN 0      4096         0.0.0.0:2222      0.0.0.0:*
```

Les sessions ouvertes survivent à l'opération. Pensez au pare-feu avant de changer de port (chapitre 21).

19.6 Copier des fichiers

Trois outils transportent des fichiers par SSH. Tous acceptent les noms définis dans `~/.ssh/config`.

`scp` copie comme `cp`, avec une destination de la forme `machine:chemin` :

```
gary@vostro:~$ scp rapport.txt serveur:/tmp/
rapport.txt                                   100%   16    65.9KB/s   00:00
gary@vostro:~$ scp serveur:/etc/hostname ./nom-du-serveur.txt
hostname                                      100%    8     6.3KB/s   00:00
```

`-r` copie un dossier entier.

`sftp` ouvre une session interactive, avec ses propres commandes : `ls`, `cd`, `get` pour rapatrier, `put` pour envoyer, `bye` pour sortir. Les logiciels graphiques de transfert, comme FileZilla, parlent le même protocole. Il n'y a rien à installer sur le serveur.

```
gary@vostro:~$ sftp serveur
Connected to serveur.
sftp> get /etc/os-release
Fetching /etc/os-release to os-release
sftp> bye
```

`rsync`, vu au chapitre 16, reste le meilleur choix dès qu'il y a beaucoup de fichiers ou qu'on recommence souvent : il ne transfère que les différences.

19.7 Les tunnels

Une connexion SSH peut transporter autre chose qu'un terminal. Elle sait relayer des connexions réseau d'un bout à l'autre, à l'abri du chiffrement. On appelle cela un **tunnel**.

Atteindre un service interne

Sur un serveur bien tenu, la base de données et les outils d'administration n'écoutent que sur `127.0.0.1` : ils sont inaccessibles du dehors. Supposons un service web de ce genre, sur le port 8000 du serveur. Depuis le poste, il ne répond pas. L'option `-L` crée le passage :

```
gary@vostro:~$ ssh -f -N -L 8080:localhost:8000 serveur
gary@vostro:~$ curl -s -o /dev/null -w '%{http_code}\n' http://localhost:8080/
200
```

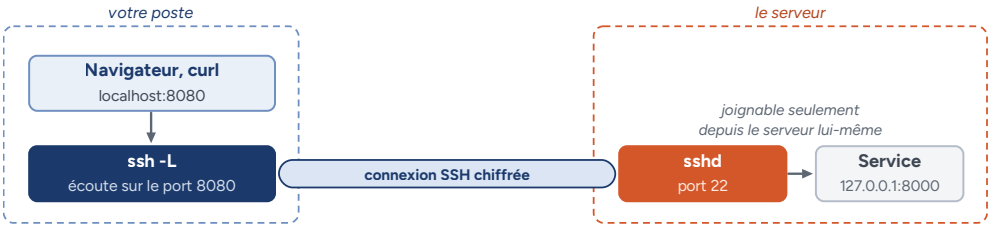


Figure 19.2 — Un tunnel local : le port 8080 du poste mène au port 8000 du serveur.

La formule `8080:localhost:8000` se lit en deux temps. `8080` est le port que `ssh` ouvre sur votre poste. `localhost:8000` est la destination, vue depuis le serveur. Tout ce qui entre par le premier ressort vers la seconde. `-N` demande de n'ouvrir aucun terminal, `-f` de passer en arrière-plan.

C'est la bonne manière d'atteindre une base de données ou une interface d'administration : le service reste fermé au réseau, et seuls les détenteurs d'une clé SSH y accèdent.

Option	Sens du tunnel	Usage
`-L`	Un port du poste mène à un service vu du serveur	Atteindre un service interne
`-R`	Un port du serveur mène à un service du poste	Montrer à l'extérieur une application en cours de développement
`-D`	Un port du poste relaie toutes les connexions qu'on lui confie	Naviguer comme si l'on était sur le serveur

`-R` s'écrit comme `-L`, par exemple `-R 9090:localhost:3000`. `-D` n'attend qu'un numéro de port.

Avec `-D`, `ssh` se comporte en relais SOCKS, que les navigateurs et `curl` savent utiliser :

```
gary@vostro:~$ ssh -f -N -D 1080 serveur
gary@vostro:~$ curl -s --socks5-hostname localhost:1080 -o /dev/null \
    -w '%{http_code}\n' https://ubuntu.com
200
```

La requête est partie du serveur, pas du poste.

19.8 Le rebond

Dans un réseau bien conçu, une seule machine est exposée à Internet. Les autres n'ont que des adresses privées. On les atteint en passant par la première, qu'on appelle un **rebond** ou un bastion.

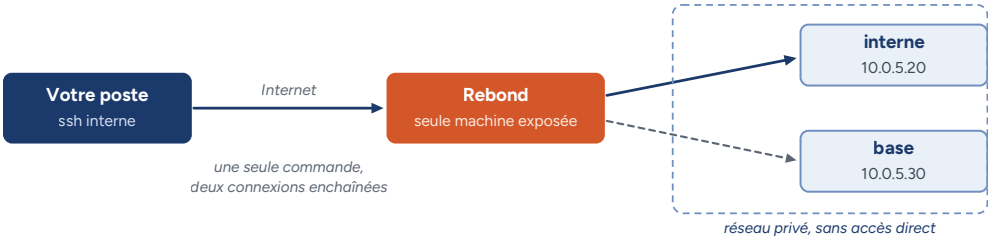


Figure 19.3 — Le rebond : une machine exposée, les autres derrière elle.

L'option `-J` enchaîne les deux connexions en une commande. L'essai ci-dessous passe par notre serveur pour revenir sur lui-même : la variable `SSH_CONNECTION` montre que la seconde connexion part bien du serveur, et non du poste.

```
gary@vostro:~$ ssh -J serveur gary@127.0.0.1 'echo $SSH_CONNECTION'
127.0.0.1 59106 127.0.0.1 22
```

En pratique, on l'écrit une fois pour toutes dans `~/.ssh/config` :

```
Host rebond
    HostName rebond.exemple.net
    User gary

Host interne
    HostName 10.0.5.20
    User gary
    ProxyJump rebond
```

`ssh interne` suffit ensuite, et `scp` ou `rsync` suivent le même chemin. Votre clé privée ne quitte jamais votre poste : le rebond ne fait que relayer une connexion chiffrée de bout en bout.

19.9 Un second facteur

Une clé volée avec sa phrase secrète ouvre toutes les portes. Pour les accès les plus sensibles, on exige en plus un code à six chiffres, renouvelé toutes les trente se-

condes par une application du téléphone. C'est le même mécanisme que celui des banques et des messageries.

Sur le serveur, on installe le module, qui vient du dépôt `universe` : `sudo apt install libpam-google-authenticator`. Malgré son nom, il suit une norme ouverte et fonctionne avec toutes les applications de ce type.

Chaque utilisateur s'inscrit ensuite avec son propre compte :

```
gary@serveur:~$ google-authenticator -t -d -f -r 3 -R 30 -w 3
[…]
Your new secret key is: YRFLEVVP…
Your emergency scratch codes are:
  66611339
  88347019
[…]
```

La commande affiche un code QR à photographier avec l'application, demande un premier code pour s'assurer que tout est bien réglé, puis donne des codes de secours. Notez-les : ils serviront le jour où le téléphone sera perdu. Les options demandent des codes fondés sur l'heure (`-t`), utilisables une seule fois (`-d`), limitent les essais à trois par demi-minute (`-r 3 -R 30`) et tolèrent un léger décalage d'horloge (`-w 3`).

Reste à demander à `sshd` d'exiger ce code. Dans `/etc/pam.d/sshd` , on ajoute deux lignes en tête et on neutralise la demande de mot de passe :

```
auth required pam_google_authenticator.so nullok
auth required pam_permit.so
#@include common-auth
```

Le mot `nullok` laisse entrer sans code les comptes qui ne se sont pas encore inscrits. On le retire quand tout le monde l'a fait.

Puis on crée `/etc/ssh/sshd_config.d/30-second-facteur.conf` :

```
KbdInteractiveAuthentication yes
AuthenticationMethods publickey,keyboard-interactive
```

La seconde ligne exige les deux preuves, dans cet ordre : la clé, puis le code. Après `sudo sshd -t` et `sudo systemctl reload ssh` , en gardant une session ouverte :

```
gary@vostro:~$ ssh serveur
(gary@192.168.1.50) Verification code:
[…]
```

```
gary@serveur:~$ journalctl -u ssh -n 2 -o cat
Accepted google_authenticator for gary
Accepted keyboard-interactive/pam for gary from 192.168.1.1 port 39284 ssh2
```

La clé a été présentée par l'agent sans rien demander, puis le serveur a réclamé le code. L'horloge du serveur doit être à l'heure : un décalage de plus d'une minute, et tous les codes sont refusés.

En résumé

- `ssh utilisateur@machine`. À la première connexion, on compare l'empreinte de la clé d'hôte.
- `ssh-keygen -t ed25519` crée la paire de clés, toujours avec une phrase secrète. `ssh-copy-id` installe la clé publique.
- La clé privée ne quitte jamais le poste. L'agent évite de retaper la phrase.
- `~/.ssh/config` donne un nom court à chaque serveur.
- Sur le serveur, on écrit dans `/etc/ssh/sshd_config.d/`. La première valeur lue l'emporte : `sshd -T` dit ce qui s'applique vraiment.
- `PasswordAuthentication no`, `PermitRootLogin no`, `AllowGroups`.
- `sshd -t`, une session gardée ouverte, un essai depuis un autre terminal.
- `scp` et `sftp` copient des fichiers. `-L` ouvre un tunnel vers un service interne. `-J` passe par un rebond.
- Un code à usage unique s'ajoute à la clé pour les accès sensibles.

Atelier — Verrouiller l'accès au serveur

Objectif : passer d'un serveur accessible par mot de passe à un serveur qui n'accepte que votre clé, sans jamais risquer de vous enfermer dehors. Puis atteindre par un tunnel un service invisible du réseau. Comptez quarante-cinq minutes.

La clé

1. Sur le poste, créez une paire de clés Ed25519 protégée par une phrase secrète. Si vous en avez déjà une, gardez-la.
2. Avant d'aller plus loin, relevez sur la console du serveur l'empreinte de sa clé d'hôte, et comparez-la à celle qu'affiche votre première connexion.

3. Installez votre clé publique avec `ssh-copy-id`, puis vérifiez que la connexion ne demande plus le mot de passe du compte.
4. Créez `~/.ssh/config` avec un bloc pour le serveur, et vérifiez que `ssh serveur` suffit.

Le verrou

5. Ouvrez une session sur le serveur, et ne la fermez plus avant la fin de l'atelier.
6. Relevez la configuration effective avec `sudo sshd -T`, pour les quatre directives du chapitre. Regardez ce que contient déjà `/etc/ssh/sshd_config.d/`.
7. Créez le groupe `acces-ssh`, inscrivez-y votre compte, puis créez `10-durcissement.conf` sur le modèle du chapitre.
8. Vérifiez la syntaxe, rechargez, et contrôlez le résultat avec `sudo sshd -T`. Si `passwordauthentication` vaut encore `yes`, cherchez quel fichier est lu avant le vôtre.

Les preuves

9. Depuis un second terminal du poste, vérifiez les trois comportements attendus :

```
ssh serveur hostname                          # doit répondre
ssh -o PubkeyAuthentication=no serveur        # doit être refusé
ssh root@192.168.1.50                         # doit être refusé
```

10. Lisez sur le serveur ce que le journal a retenu de ces trois essais : `journalctl -u ssh -n 20`.

Le tunnel

11. Sur le serveur, lancez un petit service web qui n'écoute que sur la boucle locale :

```
python3 -m http.server 8000 --bind 127.0.0.1 --directory /usr/share/doc
```

12. Depuis le poste, constatez que `curl http://192.168.1.50:8000/` échoue. Ouvrez un tunnel avec `ssh -N -L 8080:localhost:8000 serveur`, puis, dans un autre terminal, demandez `http://localhost:8080/`.

Vérifier

- `sudo sshd -T` affiche `passwordauthentication no` et `permitrootlogin no`.
- Vous vous connectez par `ssh serveur`, sans mot de passe de compte.
- Une connexion sans clé est refusée avec la mention `(publickey)`.
- Le service du port 8000 répond par le tunnel, et par lui seulement.

Exercices

Exercice 1 — Clés

1. Quelle est l'empreinte de votre clé publique ? Et celle de la clé d'hôte Ed25519 de votre serveur ?
2. Changez la phrase secrète de votre clé privée sans recréer la clé. Cherchez l'option dans `man ssh-keygen`.
3. Ajoutez sur le serveur la clé publique d'un second poste, à la main, dans `authorized_keys`. Quels droits doit avoir ce fichier ?
4. Que se passe-t-il si vous donnez à `authorized_keys` les droits `666` ? Où lisez-vous l'explication ?

Exercice 2 — Configuration du client

Écrivez un fichier `~/.ssh/config` pour trois machines :

1. `web`, à l'adresse `203.0.113.10`, compte `deploy`, port `2222` ;
2. `base`, à l'adresse privée `10.0.5.30`, compte `gary`, joignable seulement en passant par `web` ;
3. toutes les machines : un signe de vie toutes les trente secondes.

Vérifiez avec `ssh -G base` que le rebond est bien pris en compte.

Exercice 3 — Configuration du serveur

1. Affichez les valeurs effectives de `port`, `permitrootlogin` et `passwordauthentication`.
2. Autorisez un nouveau compte, `alice`, à se connecter par SSH, sans toucher à la configuration de `sshd`.
3. Un collègue a écrit `PasswordAuthentication no` dans `/etc/ssh/sshd_config.d/99-securite.conf`, et les mots de passe sont toujours acceptés. Expliquez pourquoi, et corrigez.
4. Introduisez volontairement une faute dans votre fichier, et lancez `sudo sshd -t`. Que se passerait-il si vous rechargiez le service sans cette vérification ?

Exercice 4 — Tunnels

1. Le serveur fait tourner une base PostgreSQL qui écoute sur `127.0.0.1:5432`. Quelle commande vous permet de vous y connecter depuis votre poste, sur le port 15432 ?
2. Rendez ce tunnel automatique à chaque `ssh serveur`, par une directive du fichier de configuration.
3. Faites sortir `curl` par le serveur à l'aide d'un relais SOCKS, et comparez l'adresse publique vue par un site avec et sans le relais.

Exercice 5 — Diagnostic

Que vérifiez-vous dans chacune de ces situations ?

1. `ssh serveur` répond `Connection refused`.
2. `ssh serveur` ne répond rien, puis `Connection timed out`.
3. `ssh serveur` demande un mot de passe alors que vous avez installé votre clé.
4. `ssh serveur` affiche le grand avertissement `REMOTE HOST IDENTIFICATION HAS CHANGED`.
5. Après une modification de `sshd_config`, plus personne ne peut se connecter.

Pour aller plus loin

Sur la machine. `man ssh_config` décrit toutes les directives du client, `man sshd_config` celles du serveur. `man ssh-keygen` couvre la gestion des clés, y compris les certificats, qui évitent de distribuer des clés publiques une à une sur un grand parc.

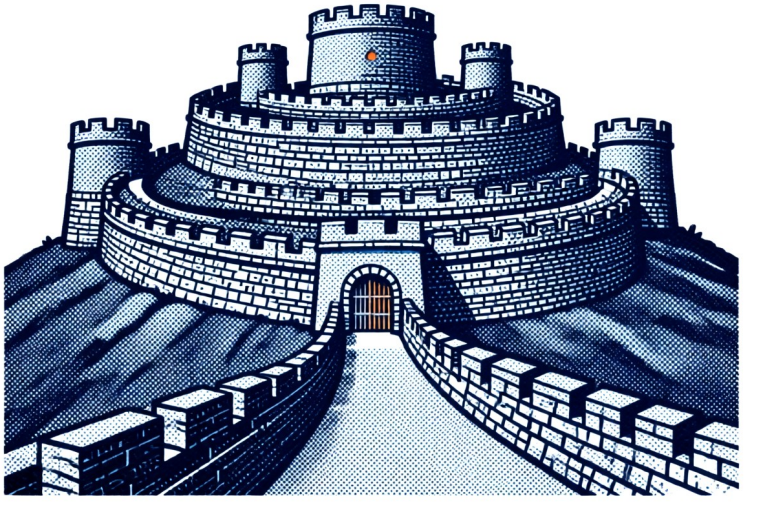
Outils. `mosh` remplace `ssh` sur les liaisons instables : la session survit à un changement de réseau ou à une mise en veille. `autossh` relance un tunnel qui tombe. `ssh-audit`, du dépôt `universe`, examine les algorithmes proposés par un serveur.

La suite. Le serveur n'accepte plus que des clés, mais les robots continuent de frapper à la porte. Le chapitre 21 leur opposera un pare-feu et Fail2ban, qui bannit les adresses trop insistantes. Avant cela, le chapitre 20 passe en revue le durcissement du système tout entier.

PARTIE VII

Sécurité du système

Préparer un serveur Ubuntu à vivre sur Internet : durcissement, pare-feu, confinement des services.





CHAPITRE 20

Durcir le système

Un serveur fraîchement installé n'est pas une passoire, mais il est réglé pour convenir à tout le monde. Le durcir, c'est le régler pour un seul usage, retirer ce qui ne sert pas et resserrer ce qui reste. On mesure, on corrige, on mesure de nouveau.

DANS CE CHAPITRE

- Appliquer les trois principes du durcissement à une machine.
- Contrôler les mises à jour automatiques de sécurité.
- Imposer une politique de mots de passe et gérer la durée de vie des comptes.
- Accorder par `sudo` le strict nécessaire.
- Réduire les services en marche et les ports ouverts.
- Régler le noyau et les options de montage.
- Garder la trace des modifications sensibles.
- Mesurer le résultat avec Lynis.

> **AVANT DE COMMENCER**
>
> Les chapitres 6, 12, 14 et 19. Tout se passe sur le serveur des chapitres précédents, déjà accessible par clé SSH uniquement. Prenez un instantané de la machine virtuelle avant de commencer : plusieurs réglages de ce chapitre peuvent gêner un service s'ils sont mal dosés.

20.1 Trois principes

Le **moindre privilège** : chaque compte et chaque programme ne reçoit que les droits dont il a besoin pour sa tâche. Le service `veilleur` du chapitre 12 en est l'exemple : un compte à lui, un seul dossier où écrire.

La **surface d'attaque réduite** : ce qui n'est pas installé ne peut pas être attaqué. Un port fermé, un service arrêté, un paquet retiré, ce sont autant de failles futures qui ne vous concernent plus.

La **défense en profondeur** : aucune protection n'est parfaite, on en superpose donc plusieurs. Celui qui franchit la première doit en trouver une autre derrière.

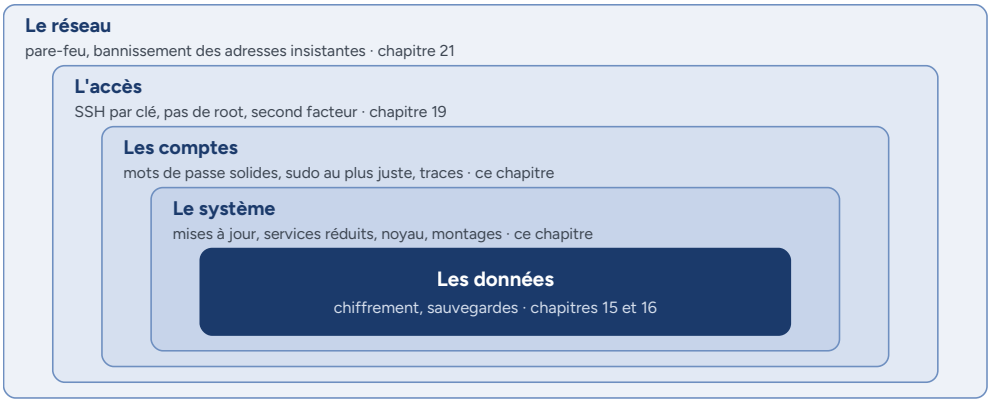


Figure 20.1 — Les protections se superposent, du réseau jusqu'aux données.

Ce chapitre s'occupe des deux couches du milieu. Ubuntu fait déjà une bonne part du travail, et nous commencerons chaque fois par regarder ce qui est en place.

> **ATTENTION · Durcir, c'est aussi casser**
>
> Chaque restriction peut empêcher un programme légitime de fonctionner. Appliquez un réglage à la fois, vérifiez que vos services marchent toujours, et notez ce que vous avez changé. Un serveur parfaitement verrouillé qui ne rend plus son service n'a aucun intérêt.

20.2 Les mises à jour de sécurité

La première cause d'intrusion n'est pas une attaque savante. C'est une faille connue, corrigée depuis des mois, sur une machine qui n'a pas été mise à jour.

Ubuntu installe seul les correctifs de sécurité, grâce au paquet `unattended-upgrades` :

```
gary@serveur:~$ systemctl is-enabled unattended-upgrades
enabled
gary@serveur:~$ cat /etc/apt/apt.conf.d/20auto-upgrades
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
```

Chaque jour, la liste des paquets est rafraîchie et les correctifs de sécurité sont installés. Seuls ceux-là : les autres mises à jour attendent votre `sudo apt upgrade`. L'option `--dry-run` montre ce qui serait fait :

```
gary@serveur:~$ sudo unattended-upgrade --dry-run -v 2>&1 | head -n 6
Starting unattended upgrades script
Allowed origins are: o=Ubuntu,a=noble, o=Ubuntu,a=noble-security, […]
[…]
Option --dry-run given, *not* performing real actions
Packages that will be upgraded: dracut-install […] libssl3t64 libxpm4 openssl
```

Le compte rendu de chaque passage se trouve dans `/var/log/unattended-upgrades/`.

Reste le redémarrage. Un noyau corrigé ne protège qu'une fois chargé. Quand un redémarrage est nécessaire, le fichier `/var/run/reboot-required` apparaît, et le message d'accueil le signale. Sur un serveur qu'on ne consulte pas tous les jours, on peut laisser la machine redémarrer seule à une heure creuse, par un fichier `/etc/apt/apt.conf.d/52unattended-upgrades-local` :

```
// Redémarrer seul, à 4 h 30, quand une mise à jour l'exige
Unattended-Upgrade::Automatic-Reboot "true";
Unattended-Upgrade::Automatic-Reboot-Time "04:30";
```

C'est un choix. Un serveur dont l'arrêt de deux minutes pose problème se redémarre à la main, à une date convenue. Mais il se redémarre.

> **NOTE · Ubuntu Pro**
>
> L'abonnement Ubuntu Pro étend les correctifs de sécurité aux paquets du dépôt `universe`, porte la durée de maintenance à dix ans et donne accès à Livepatch, qui corrige le noyau sans redémarrage. Il est gratuit pour un usage personnel, jusqu'à cinq machines. `pro status` indique si la machine y est rattachée.

20.3 Les comptes et les mots de passe

Une politique de mots de passe

SSH n'accepte plus que des clés, mais les mots de passe servent encore à `sudo` et à la console. Le module `pwquality` refuse ceux qui sont trop faibles :

```
gary@serveur:~$ sudo apt install libpam-pwquality
gary@serveur:~$ grep pwquality /etc/pam.d/common-password
password        requisite                       pam_pwquality.so retry=3
```

L'installation l'a branché d'elle-même sur le changement de mot de passe. Ses réglages se posent dans un fichier du dossier `/etc/security/pwquality.conf.d/`, que l'installation ne crée pas : `sudo mkdir /etc/security/pwquality.conf.d`, puis on y écrit `50-politique.conf` :

```
# Politique de mots de passe
minlen = 12
enforce_for_root
```

`minlen` fixe la longueur minimale. `enforce_for_root` applique la règle même quand c'est l'administrateur qui choisit le mot de passe d'un autre. L'effet est immédiat :

```
gary@serveur:~$ sudo passwd alice
New password:
BAD PASSWORD: The password is shorter than 12 characters
New password:
```

La longueur protège mieux que la complication. Une phrase de quatre ou cinq mots ordinaires est plus solide, et plus facile à retenir, que huit caractères hérissés de symboles.

La vie d'un compte

Commande	Effet
`sudo chage -l alice`	Afficher les dates d'expiration du compte et du mot de passe
`sudo chage -M 365 alice`	Faire expirer le mot de passe au bout d'un an
`sudo chage -E 2026-12-31 alice`	Faire expirer le compte à une date donnée
`sudo passwd -l alice`	Verrouiller le mot de passe
`sudo passwd -u alice`	Le déverrouiller
`sudo usermod -s /usr/sbin/nologin alice`	Interdire toute ouverture de session

Une date d'expiration convient aux comptes temporaires : un stagiaire, un prestataire. Quand quelqu'un quitte l'équipe, on verrouille son compte le jour même, on retire sa clé de `authorized_keys`, et on ne supprime le compte qu'après avoir décidé du sort de ses fichiers.

> **ATTENTION · Verrouiller un mot de passe ne ferme pas SSH**
>
> `passwd -l` rend le mot de passe inutilisable. Une clé SSH installée continue d'ouvrir la session. Pour fermer vraiment la porte, retirez le compte du groupe `acces-ssh` du chapitre 19, ou donnez-lui le shell `nologin`.

20.4 sudo, au plus juste

Le premier compte créé à l'installation appartient au groupe `sudo` : il peut tout faire, après avoir donné son mot de passe. C'est ce qu'il faut à l'administrateur. Pour les autres besoins, `sudo` sait accorder une seule commande.

Un compte `deploiement` doit pouvoir relancer un service après une mise en ligne, sans plus. On l'écrit dans un fichier du dossier `/etc/sudoers.d/` :

```
# Le compte de déploiement peut relancer cron, et rien d'autre
deploiement ALL=(root) NOPASSWD: /usr/bin/systemctl restart cron
```

La ligne se lit ainsi : le compte `deploiement`, sur toutes les machines, peut exécuter sous l'identité de `root`, sans mot de passe, cette commande précise.

```
gary@serveur:~$ sudo -l -U deploiement | tail -n 2
User deploiement may run the following commands on serveur:
    (root) NOPASSWD: /usr/bin/systemctl restart cron
gary@serveur:~$ sudo -u deploiement sudo -n systemctl restart cron && echo "autorisé"
autorisé
gary@serveur:~$ sudo -u deploiement sudo -n cat /etc/shadow
sudo: a password is required
```

La commande prévue passe. Toute autre est refusée. `sudo -l -U compte` liste ce qu'un compte a le droit de faire : c'est la commande d'audit à connaître.

Trois règles pour ces fichiers.

Modifiez-les toujours avec `sudo visudo -f /etc/sudoers.d/nom`. `visudo` vérifie la syntaxe avant d'enregistrer. Une erreur dans `sudoers` rend `sudo` inutilisable pour tout le monde, vous compris. `visudo -c` contrôle un fichier sans l'ouvrir :

```
gary@serveur:~$ sudo visudo -cf /tmp/faute-sudoers
/tmp/faute-sudoers:1:46: syntax error
deploiement ALL=(root) NOPASSWD /usr/bin/true
                                            ^
```

Écrivez le chemin complet de chaque commande, avec ses arguments. `systemctl restart cron` et non `systemctl` tout court, qui permettrait d'arrêter n'importe quel service.

Réservez `NOPASSWD` aux comptes de service qui ne peuvent pas saisir de mot de passe, et à des commandes sans danger. Méfiez-vous des commandes qui en lancent d'autres : autoriser `vim`, `less` ou `find` par `sudo`, c'est donner un shell `root`.

Chaque appel à `sudo` laisse dans le journal une trace, avec le compte, le dossier et la commande. `journalctl -t sudo` les affiche :

```
gary : TTY=pts/0 ; PWD=/home/gary ; USER=root ; COMMAND=/usr/sbin/useradd essai-audit
```

C'est l'une des raisons de ne jamais travailler dans un shell `root` permanent : on y perd cette trace.

20.5 Les fichiers sensibles

Les droits des fichiers critiques sont bien réglés d'origine. Il faut seulement savoir les contrôler, pour repérer ce qui aurait été modifié.

```
gary@serveur:~$ ls -l /etc/shadow /etc/sudoers
-rw-r----- 1 root shadow 1062 Oct  4 22:58 /etc/shadow
-r--r----- 1 root root   1800 Jan 29  2024 /etc/sudoers
gary@serveur:~$ ls -ld /home/gary
drwxr-x--- 5 gary  gary  4096 Oct  4 22:56 /home/gary
```

Depuis Ubuntu 21.04, les répertoires personnels sont fermés aux autres utilisateurs.

Deux recherches font partie de tout contrôle :

```
gary@serveur:~$ sudo find / -xdev -perm -4000 -type f | wc -l
13
gary@serveur:~$ sudo find / -xdev -type f -perm -0002 | wc -l
0
```

La première compte les programmes setuid, qui s'exécutent avec les droits de leur propriétaire (chapitre 6). Il y en a treize sur ce serveur : `sudo`, `passwd`, `mount` et quelques autres. Gardez cette liste. Un quatorzième qui apparaîtrait sans que vous ayez rien installé mériterait une enquête. La seconde cherche les fichiers que tout le monde peut modifier : il ne doit pas y en avoir.

20.6 Moins de services, moins de ports

Chaque service en marche est du code qui tourne, souvent en `root`, parfois à l'écoute du réseau. Deux listes disent tout :

```
gary@serveur:~$ systemctl list-units --type=service --state=running --no-legend | wc -l
20
gary@serveur:~$ sudo ss -tuln | awk 'NR>1 {print $1, $5}'
udp 127.0.0.54:53
udp 127.0.0.53%lo:53
tcp 0.0.0.0:22
```

```
tcp 127.0.0.54:53
tcp 127.0.0.53%lo:53
tcp [::]:22
```

Côté réseau, ce serveur est déjà sobre : seul SSH écoute au-dehors. Les autres lignes sont le relais DNS, joignable de la machine seule.

Parmi les vingt services, deux ne servent à rien ici. `ModemManager` gère les modems cellulaires. `multipathd` gère les baies de stockage reliées par plusieurs chemins. Cette machine n'a ni l'un ni l'autre :

```
gary@serveur:~$ sudo systemctl disable --now ModemManager.service \
    multipathd.service multipathd.socket
[…]
gary@serveur:~$ systemctl is-active ModemManager multipathd
inactive
inactive
```

Avant de désactiver un service que vous ne connaissez pas, lisez sa description avec `systemctl status` et la page de manuel qu'elle cite. Dans le doute, laissez-le. La bonne question n'est pas « puis-je l'arrêter ? », mais « à quoi sert-il sur cette machine ? ».

20.7 Les réglages du noyau

Le noyau expose des centaines de paramètres sous `/proc/sys`. `sysctl` les lit et les modifie. Ubuntu en a déjà durci plusieurs, dans les fichiers de `/etc/sysctl.d/` :

Paramètre	Valeur d'origine	Effet
`kernel.dmesg_restrict`	1	Seul `root` lit les messages du noyau
`kernel.kptr_restrict`	1	Les adresses internes du noyau sont masquées
`kernel.yama.ptrace_scope`	1	Un processus ne peut espionner que ses propres enfants
`kernel.randomize_va_space`	2	La disposition de la mémoire change à chaque exécution
`net.ipv4.tcp_syncookies`	1	Protection contre les rafales de demandes de connexion
`fs.protected_symlinks`	1	Protection contre des détournements par liens dans `/tmp`

Quelques réglages du réseau restent permissifs. On les resserre dans un fichier à soi, `/etc/sysctl.d/60-durcissement.conf` :

```
# Ni accepter ni émettre de redirections ICMP
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0

# Vérifier strictement l'adresse source des paquets
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# Journaliser les paquets aux adresses impossibles
net.ipv4.conf.all.log_martians = 1
```

Une redirection ICMP est un message par lequel un routeur conseille un autre chemin. Un serveur n'en a pas l'usage, et un attaquant placé sur le même réseau peut s'en servir pour détourner le trafic. Le filtrage de l'adresse source rejette les paquets qui prétendent venir d'un réseau d'où ils ne peuvent pas arriver.

```
gary@serveur:~$ sudo sysctl --system
[…]
* Applying /etc/sysctl.d/60-durcissement.conf ...
[…]
gary@serveur:~$ sysctl net.ipv4.conf.all.accept_redirects net.ipv4.conf.all.rp_filter
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.rp_filter = 1
```

`sysctl --system` relit tous les fichiers. Les réglages sont aussi appliqués à chaque démarrage.

> **ATTENTION · Pas sur un routeur**
>
> Ces réglages conviennent à un serveur qui a une seule carte réseau. Une machine qui relaie du trafic, qui héberge des conteneurs avec un réseau élaboré ou qui a plusieurs sorties vers Internet peut avoir besoin d'un filtrage plus souple. Testez.

20.8 Les options de montage

Un dossier de données n'a aucune raison de contenir des programmes. Les options de montage du chapitre 14 permettent de l'interdire. Dans `/etc/fstab`, la ligne de `/srv/donnees` devient :

```
UUID=b438851b-… /srv/donnees ext4 defaults,nofail,nodev,nosuid,noexec 0 2
```

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo mount -o remount /srv/donnees
gary@serveur:~$ findmnt -no OPTIONS /srv/donnees
rw,nosuid,nodev,noexec,relatime
gary@serveur:~$ /srv/donnees/essai.sh
-bash: /srv/donnees/essai.sh: Permission denied
gary@serveur:~$ sh /srv/donnees/essai.sh
bonjour
```

Le script, pourtant exécutable, est refusé. La dernière commande montre la limite de `noexec` : rien n'empêche de donner le script à lire à un interpréteur. L'option arrête un programme déposé là par une faille d'un site web, pas un utilisateur décidé. C'est un obstacle de plus, pas une garantie.

Les candidats habituels sont les partitions de données, `/home` sur un serveur partagé, et `/dev/shm`, qu'Ubuntu monte déjà avec `nosuid` et `nodev`. Pour `/tmp`, la prudence s'impose : des installations de paquets y exécutent des scripts, et `noexec` les fait échouer.

20.9 Les périphériques USB

Sur un serveur physique, une clé USB branchée par un visiteur est un moyen d'apporter un programme ou d'emporter des données. On peut interdire au noyau de charger le pilote des supports de stockage USB, par un fichier `/etc/modprobe.d/usb-storage.conf` :

```
# Interdire le chargement du pilote des clés et disques USB
install usb-storage /bin/false
```

```
gary@serveur:~$ sudo modprobe usb-storage
modprobe: ERROR: […] Error running install command '/bin/false' for module usb_storage […]
modprobe: ERROR: could not insert 'usb_storage': Invalid argument
```

La ligne `install` remplace le chargement du module par la commande `/bin/false`, qui échoue toujours. Claviers et souris ne sont pas concernés. Ce réglage n'empêche pas de démarrer la machine sur une clé de secours, car il n'agit qu'une fois Ubuntu lancé. Il empêche en revanche d'y brancher un disque USB de sauvegarde. À vous de peser l'un contre l'autre.

20.10 Un avertissement à l'entrée

Une bannière affichée avant la connexion n'arrête personne. Elle sert à établir, en cas de litige, que l'accès était explicitement réservé et que les connexions sont enregistrées. Elle ne doit rien révéler de la machine.

Le texte se place dans `/etc/issue.net`, et une ligne dans la configuration de `sshd` le fait afficher :

```
gary@serveur:~$ cat /etc/ssh/sshd_config.d/40-banniere.conf
Banner /etc/issue.net
gary@serveur:~$ sudo sshd -t && sudo systemctl reload ssh
```

```
gary@vostro:~$ ssh serveur
Serveur de travail, usage restreint.
Les connexions sont inscrites au journal.
```

> **SUR LE TERRAIN · Pas d'accents dans la bannière**
>
> Le client SSH n'affiche pas les lettres accentuées d'une bannière. Il les remplace par leur code : « Accès réservé » devient `Acc\303\250s r\303\251serv\303\251`. Tournez le texte pour vous en passer.

Par défaut, `/etc/issue.net` contient « Ubuntu 24.04.5 LTS ». Annoncer sa version exacte avant même l'authentification n'est pas nécessaire.

20.11 Garder la trace

Le journal et `sudo` disent qui s'est connecté et quelles commandes ont été lancées. **auditd** va plus loin : il enregistre, au niveau du noyau, tout accès à ce qu'on lui demande de surveiller.

```
gary@serveur:~$ sudo apt install auditd
gary@serveur:~$ sudo auditctl -w /etc/passwd -p wa -k comptes
gary@serveur:~$ sudo auditctl -l
-w /etc/passwd -p wa -k comptes
```

La règle surveille (`-w`) le fichier `/etc/passwd`, pour les écritures et les changements d'attributs (`-p wa`), et marque les événements du mot-clé `comptes` (`-k`). Créons un compte, puis interrogeons :

```
gary@serveur:~$ sudo useradd essai-audit
gary@serveur:~$ sudo ausearch -k comptes -i | \
    grep -oE 'comm=[^ ]+|exe=[^ ]+|auid=[^ ]+' | sort | uniq -c
      4 auid=gary
      2 comm=useradd
      2 exe=/usr/sbin/useradd
[…]
```

Le champ le plus précieux est `auid`. Il désigne le compte qui a ouvert la session à l'origine, même si la commande a été lancée par `sudo` et s'est donc exécutée en `root`. C'est Gary qui a modifié `/etc/passwd`, avec `useradd`.

Une règle posée par `auditctl` disparaît au redémarrage. Pour la conserver, on l'écrit dans un fichier de `/etc/audit/rules.d/`, puis on recharge avec `sudo augenrules --load` :

```
# Surveiller toute modification des comptes
-w /etc/passwd -p wa -k comptes
-w /etc/shadow -p wa -k comptes
-w /etc/sudoers.d/ -p wa -k comptes
```

20.12 Mesurer : Lynis

Lynis passe la machine en revue : plus de deux cent cinquante contrôles, du chargeur de démarrage aux droits des fichiers. Il ne modifie rien. Il s'installe depuis le dépôt `universe`.

```
gary@serveur:~$ sudo apt install lynis
gary@serveur:~$ sudo lynis audit system --quick
[…]
  Hardening index : 61 [############        ]
  Tests performed : 256
```

Avant ce chapitre, le serveur obtenait 61. Après les mises à jour, la politique de mots de passe, les réglages du noyau, la bannière et auditd :

```
gary@serveur:~$ sudo lynis audit system --quick | grep -E 'Hardening index|Tests performed'
  Hardening index : 70 [##############      ]
  Tests performed : 259
```

Le détail se trouve dans `/var/log/lynis-report.dat`. Chaque ligne qui commence par `warning` ou `suggestion` porte un code, que `sudo lynis show details CODE` explique :

```
gary@serveur:~$ sudo grep '^warning' /var/log/lynis-report.dat | cut -d'|' -f1,2
warning[]=FIRE-4512|iptables module(s) loaded, but no rules active
gary@serveur:~$ sudo grep -c '^suggestion' /var/log/lynis-report.dat
42
```

Il reste un avertissement : la machine n'a pas de pare-feu. Ce sera l'objet du chapitre 21.

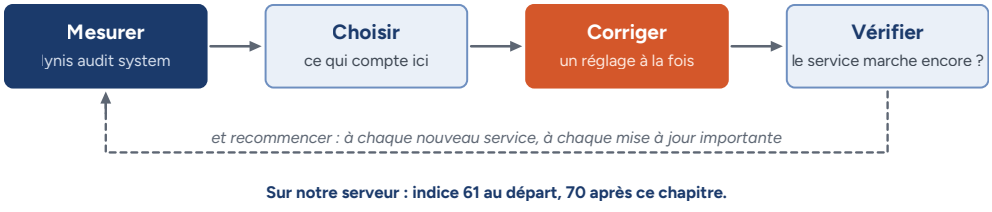


Figure 20.2 — La boucle du durcissement.

> **À RETENIR · Un indice n'est pas un but**
>
> Ne courez pas après le chiffre. Certaines des quarante-deux suggestions n'ont pas de sens sur votre machine, d'autres coûteraient plus qu'elles ne rapportent. Lisez-les, retenez celles qui répondent à un risque réel pour ce serveur, et notez pourquoi vous écartez les autres. Un indice de 70 compris vaut mieux qu'un 85 obtenu en appliquant une liste sans la lire.

Les référentiels

Lynis donne un avis général. Les organisations soumises à des obligations s'appuient sur un référentiel écrit, dont le plus répandu est le *CIS Benchmark*. Celui d'Ubuntu 24.04 énumère plusieurs centaines de réglages, répartis en deux niveaux : le premier applicable presque partout, le second plus strict. Presque tout ce chapitre en est tiré.

Canonical fournit un outil qui audite et applique ce référentiel, l'*Ubuntu Security Guide*, réservé aux machines rattachées à Ubuntu Pro. `sudo usg audit cis_level1_server` produit un rapport détaillé.

En résumé

- Moindre privilège, surface d'attaque réduite, défense en profondeur.

- Les correctifs de sécurité s'installent seuls. Il faut encore redémarrer quand `/var/run/reboot-required` existe.
- `libpam-pwquality` impose une longueur minimale. `chage` et `passwd -l` règlent la vie des comptes.
- `sudo` accorde une commande précise par un fichier de `/etc/sudoers.d/`, écrit avec `visudo`. `sudo -l -U compte` liste les droits.
- Ce qui ne sert pas s'arrête : `systemctl disable --now`. `ss -tuln` montre ce qui écoute.
- Les réglages du noyau se posent dans `/etc/sysctl.d/`, les options `nodev,nosuid,noexec` dans `fstab`.
- auditd enregistre qui a touché aux fichiers surveillés.
- Lynis mesure. On corrige un réglage à la fois, et on vérifie que le service marche encore.

Atelier — Durcir et mesurer

Objectif : relever l'indice de durcissement du serveur, appliquer les réglages du chapitre un par un, et mesurer le gain. Comptez une heure.

Mesurer

1. Prenez un instantané de la machine virtuelle.
2. Installez Lynis et lancez un premier audit. Notez l'indice, le nombre d'avertissements et le nombre de suggestions :

```
sudo apt install lynis
sudo lynis audit system --quick
sudo grep -c '^suggestion' /var/log/lynis-report.dat
```

3. Relevez l'état de départ : le nombre de services en marche, les ports à l'écoute, le nombre de programmes setuid. Gardez ces trois listes dans un fichier.

Corriger

4. Appliquez les mises à jour en attente avec `sudo apt update && sudo apt upgrade`. Redémarrez si `/var/run/reboot-required` existe.
5. Installez `libpam-pwquality` et fixez la longueur minimale à douze caractères. Vérifiez en essayant de vous donner un mot de passe trop court.

6. Désactivez les services qui ne servent pas sur votre machine. Pour chacun, écrivez en une phrase à quoi il servait.
7. Créez `/etc/sysctl.d/60-durcissement.conf`, appliquez-le, et vérifiez deux valeurs avec `sysctl`.
8. Ajoutez `nodev,nosuid,noexec` au montage de `/srv/donnees`, remontez, et constatez qu'un script n'y est plus exécutable.
9. Rédigez une bannière sans accents et faites-la afficher par `sshd`. Vérifiez depuis le poste.
10. Installez auditd et surveillez `/etc/passwd`, `/etc/shadow` et `/etc/sudoers.d/`, de façon permanente.

Après chaque étape, vérifiez que vous pouvez toujours ouvrir une session SSH et utiliser `sudo`.

Mesurer de nouveau

11. Relancez Lynis. De combien l'indice a-t-il monté ? Quels avertissements restent ?
12. Choisissez trois suggestions dans le rapport. Pour chacune, lisez son explication avec `sudo lynis show details`, et décidez : l'appliquer, ou l'écarter en sachant pourquoi.

Vérifier

- L'indice a progressé d'une dizaine de points. Sur le serveur du livre, il est passé de 61 à 70.
- `sudo auditctl -l` liste vos trois règles après un redémarrage.
- Vous tenez un fichier qui énumère chaque réglage appliqué et sa raison. C'est ce fichier que le chapitre 29 transformera en automatisation.

Exercices

Exercice 1 — État des lieux

1. Les mises à jour automatiques sont-elles actives sur votre machine ? Quand ont-elles tourné pour la dernière fois ?
2. Combien de paquets attendent une mise à jour ? Un redémarrage est-il nécessaire ?
3. Quels comptes ont un shell de connexion ? Lesquels appartiennent au groupe `sudo` ?

4. Quel compte a un mot de passe qui n'expire jamais ?

Exercice 2 — sudo

1. Autorisez le compte `sauvegarde` à lancer `/usr/local/bin/sauvegarde-restic` en `root`, sans mot de passe, et rien d'autre.
2. Vérifiez la syntaxe du fichier, puis les droits accordés.
3. Un collègue propose la ligne `sauvegarde ALL=(root) NOPASSWD: /usr/bin/rsync`. Quel est le danger ?
4. Retrouvez dans le journal les trois dernières commandes lancées par `sudo`.

Exercice 3 — Services et ports

1. Listez les services en marche. Pour trois d'entre eux, trouvez à quoi ils servent.
2. Quels ports écoutent sur une adresse autre que la boucle locale ?
3. Installez un service de votre choix, constatez le port qu'il ouvre, puis retirez-le. Le port est-il refermé ?

Exercice 4 — Noyau et montages

1. Quelle est la valeur de `net.ipv4.ip_forward` ? Que signifierait la valeur `1` ?
2. Dans quel fichier Ubuntu fixe-t-il `kernel.yama.ptrace_scope` ?
3. Avec quelles options `/dev/shm` est-il monté ? Et `/boot` ?
4. Pourquoi `noexec` n'empêche-t-il pas `sh script.sh` ?

Exercice 5 — Audit

1. Surveillez avec auditd les modifications du dossier `/etc/ssh/sshd_config.d/`.
2. Modifiez un fichier de ce dossier, puis retrouvez l'événement. Quel compte en est à l'origine ?
3. Rendez la règle permanente.
4. Lynis suggère `BOOT-5122`. Que recommande-t-il ? Rapprochez cette suggestion de ce que le chapitre 11 a montré du mode de secours.

Pour aller plus loin

Sur la machine. `man sudoers` pour toute la syntaxe des règles, `man pwquality.conf` pour la politique de mots de passe, `man auditctl` et `man audit.rules` pour auditd. Le fichier `/etc/sysctl.d/README.sysctl` explique l'ordre de lecture des réglages du noyau.

Outils. `aide` photographie l'état des fichiers du système et signale ensuite tout changement. `rkhunter` et `chkrootkit` cherchent les traces de programmes malveillants connus : leurs alertes demandent à être vérifiées, car les faux positifs sont fréquents. `needrestart`, installé d'origine, dit après chaque mise à jour quels services tournent encore avec d'anciennes bibliothèques.

Référentiels. Le *CIS Benchmark* d'Ubuntu 24.04 se télécharge gratuitement sur le site du Center for Internet Security, après inscription. Chaque réglage y est accompagné de sa justification : c'est une excellente lecture, même si l'on n'applique pas tout.



CHAPITRE 21

Pare-feu et protection du réseau

Un serveur relié à Internet reçoit ses premières tentatives d'intrusion dans l'heure. Le pare-feu décide de ce qui entre. Fail2ban écarte ceux qui insistent. Deux outils, une demi-heure de réglage.

DANS CE CHAPITRE

- Décrire le trajet d'un paquet dans le noyau et l'endroit où le pare-feu intervient.
- Mettre en place UFW sans se couper de son serveur.
- Écrire des règles par port, par service et par adresse d'origine.
- Lire les blocages dans le journal.
- Reconnaître les règles nftables qu'UFW produit, et en écrire soi-même.
- Bannir automatiquement les adresses trop insistantes avec Fail2ban.
- Vérifier le résultat de l'extérieur.

> **AVANT DE COMMENCER**
>
> Les chapitres 17, 19 et 20. Le serveur écoute en SSH et fait tourner nginx (`sudo apt install nginx`), que le chapitre 23 étudiera. Les vérifications se font depuis le poste `vostro` . Gardez la console du serveur à portée de main : une règle mal placée ferme la porte par laquelle vous êtes entré.

21.1 Ce que fait un pare-feu

Le chapitre 20 a réduit les services à l'écoute. Le pare-feu ajoute une seconde barrière, indépendante de la première. Il examine chaque paquet à son arrivée et décide de l'accepter ou de le jeter, d'après son adresse d'origine, son port de destination, son protocole.

Sous Linux, ce tri est fait par le noyau lui-même, dans un mécanisme appelé **Netfilter**. Le noyau présente chaque paquet à des listes de règles, les *chaînes*, placées à des points précis de son trajet.

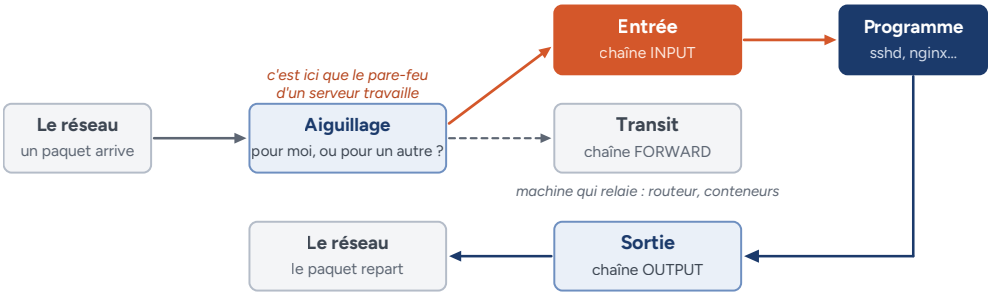


Figure 21.1 — Le trajet d'un paquet dans le noyau.

Sur un serveur, tout se joue dans la chaîne d'entrée. La chaîne de transit ne sert qu'aux machines qui relaient du trafic, et l'on filtre rarement la sortie.

REPÈRE HISTORIQUE · D'iptables à nftables

Pendant vingt ans, on a écrit les règles de Netfilter avec la commande `iptables`, apparue en 2001. Son successeur, nftables, est entré dans le noyau 3.13 en 2014. Il remplace quatre outils par un seul, `nft`, avec une syntaxe plus régulière. Ubuntu a fait la bascule en douceur : la commande `iptables` existe toujours, mais elle traduit désormais ses règles pour nftables.

```
gary@serveur:~$ sudo iptables -V
iptables v1.8.10 (nf_tables)
```

Personne ne vous demande d'écrire ces règles à la main. Ubuntu fournit depuis 2008 un outil qui s'en charge : **UFW**, *Uncomplicated Firewall*, le pare-feu sans complication.

21.2 UFW

UFW est installé d'origine, et inactif :

```
gary@serveur:~$ sudo ufw status verbose
Status: inactive
```

L'ordre des opérations

Activer un pare-feu sur un serveur distant demande de procéder dans le bon ordre : poser la politique, autoriser SSH, et seulement ensuite activer.

```
gary@serveur:~$ sudo ufw default deny incoming
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)
gary@serveur:~$ sudo ufw default allow outgoing
Default outgoing policy changed to 'allow'
(be sure to update your rules accordingly)
gary@serveur:~$ sudo ufw allow OpenSSH
Rules updated
Rules updated (v6)
gary@serveur:~$ sudo ufw allow 'Nginx Full'
Rules updated
Rules updated (v6)
```

La politique par défaut tient en deux lignes : tout ce qui entre est refusé, tout ce qui sort est permis. Les règles `allow` ouvrent ensuite les exceptions. Chacune vaut pour IPv4 et pour IPv6.

Tant qu'UFW est inactif, `ufw status` n'affiche rien. `ufw show added` relit ce qui a été saisi :

```
gary@serveur:~$ sudo ufw show added
Added user rules (see 'ufw status' for running firewall):
ufw allow OpenSSH
ufw allow 'Nginx Full'
```

SSH y figure : on peut activer.

```
gary@serveur:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
gary@serveur:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp (OpenSSH)           ALLOW IN    Anywhere
80,443/tcp (Nginx Full)    ALLOW IN    Anywhere
22/tcp (OpenSSH (v6))      ALLOW IN    Anywhere (v6)
80,443/tcp (Nginx Full (v6)) ALLOW IN    Anywhere (v6)
```

Le pare-feu est en service, et il le restera après un redémarrage.

> **DANGER · SSH d'abord**
>
> `sudo ufw enable` sans règle pour SSH, et la session en cours est la dernière. Vérifiez avec `ufw show added` avant d'activer. Si vous avez déplacé SSH sur un autre port, c'est ce port qu'il faut ouvrir : `sudo ufw allow 2222/tcp`.

> **ATTENTION · UFW a ses propres réglages du noyau**
>
> À chaque démarrage et à chaque rechargement, UFW applique le fichier `/etc/ufw/sysctl.conf`, qui l'emporte sur `/etc/sysctl.d/`. Il remet notamment `log_martians` à 0, ce qui annule l'un des réglages du chapitre 20. Pour le conserver, changez aussi les deux lignes `log_martians` de ce fichier en `=1`. Le chapitre 29 raconte comment ce conflit a été découvert.

Les profils d'applications

`OpenSSH` et `Nginx Full` ne sont pas des mots magiques. Ce sont des **profils**, de petits fichiers déposés par les paquets dans `/etc/ufw/applications.d/`. Ils donnent un nom à un jeu de ports.

```
gary@serveur:~$ sudo ufw app list
Available applications:
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
gary@serveur:~$ sudo ufw app info 'Nginx Full'
Profile: Nginx Full
Title: Web Server (Nginx, HTTP + HTTPS)
Description: Small, but very powerful and efficient web server

Ports:
  80,443/tcp
```

Écrire des règles

Commande	Effet
`sudo ufw allow 8080/tcp`	Ouvrir un port
`sudo ufw allow 'Nginx Full'`	Ouvrir les ports d'un profil
`sudo ufw allow from 192.168.1.0/24`	Tout accepter d'un réseau
`sudo ufw allow from 192.168.1.0/24 to any port 5432 proto tcp`	Ouvrir un port à un seul réseau
`sudo ufw deny from 203.0.113.5`	Refuser tout ce qui vient d'une adresse
`sudo ufw limit OpenSSH`	Accepter, mais freiner les rafales de connexions
`sudo ufw status numbered`	Lister les règles avec leur numéro
`sudo ufw delete 4`	Supprimer la règle numéro 4

La quatrième forme est la plus utile. Une base de données n'a pas à être joignable du monde entier. On n'ouvre son port qu'au réseau, ou à la machine, qui en a besoin.

`limit` est fait pour SSH. La règle accepte les connexions, mais rejette une adresse qui en ouvre six ou davantage en trente secondes : c'est le rythme d'un robot, pas celui d'un administrateur.

L'ordre des règles

UFW lit les règles de haut en bas et s'arrête à la première qui correspond.

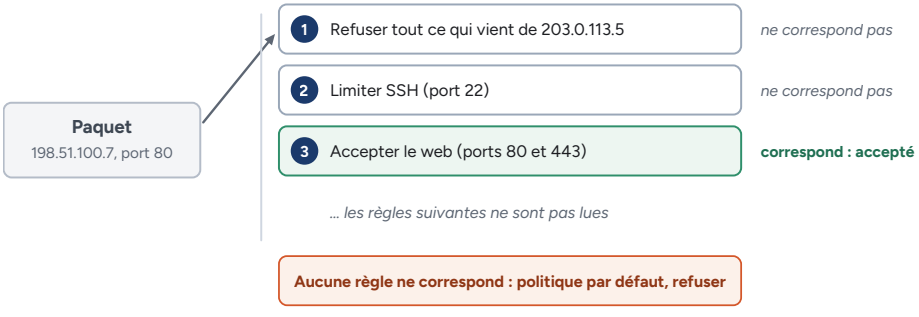


Figure 21.2 — La première règle qui correspond décide.

La conséquence est pratique. Pour refuser une adresse précise, un `deny` ajouté en fin de liste ne sert à rien si une règle plus haut accepte déjà le port visé. Il faut l'insérer en tête :

```
gary@serveur:~$ sudo ufw insert 1 deny from 203.0.113.5
Rule inserted
gary@serveur:~$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] Anywhere                   DENY IN     203.0.113.5
[ 2] OpenSSH                    LIMIT IN    Anywhere
[ 3] Nginx Full                 ALLOW IN    Anywhere
[ 4] OpenSSH (v6)               LIMIT IN    Anywhere (v6)
[ 5] Nginx Full (v6)            ALLOW IN    Anywhere (v6)
```

C'est le sort réservé à l'adresse `203.0.113.5`, repérée dans les journaux du serveur web au chapitre 9.

Lire les blocages

UFW note dans le journal du noyau une partie de ce qu'il refuse. Depuis le poste, tentons d'atteindre un port fermé, puis lisons sur le serveur :

```
gary@serveur:~$ journalctl -k -g 'UFW BLOCK' -n 1 -o cat
[UFW BLOCK] IN=enp0s3 […] SRC=192.168.1.1 DST=192.168.1.50 […] PROTO=TCP
SPT=42896 DPT=8000 […]
```

Trois champs suffisent à comprendre : `SRC`, l'adresse d'origine, `DPT`, le port visé, et `PROTO`. Sur un serveur exposé à Internet, ces lignes défilent sans arrêt. C'est le bruit de fond du réseau, et la preuve que le pare-feu travaille.

Vu de l'extérieur, un port refusé ne répond plus du tout. `nmap` le signale par le mot `filtered` :

```
gary@vostro:~$ nmap -Pn -p 22,80,443,8000 192.168.1.50
[…]
22/tcp   open     ssh
80/tcp   open     http
443/tcp  closed   https
8000/tcp filtered http-alt
```

Le port 443 est ouvert dans le pare-feu, mais aucun service n'y écoute encore : la machine répond qu'il est fermé. Le port 8000 est bloqué : la machine se tait.

21.3 Sous le capot : nftables

UFW ne filtre rien lui-même. Il traduit vos commandes en règles pour le noyau. `nft list ruleset` les montre toutes :

```
gary@serveur:~$ sudo nft list ruleset | grep -c .
327
gary@serveur:~$ sudo nft list chain ip filter ufw-user-input
# Warning: table ip filter is managed by iptables-nft, do not touch!
table ip filter {
        chain ufw-user-input {
                ip saddr 203.0.113.5 counter packets 0 bytes 0 drop
                […]
                ip protocol tcp tcp dport { 80, 443 }  counter packets 0
bytes 0 accept
        }
}
```

Cinq commandes UFW ont produit plus de trois cents lignes. La plupart forment un socle qu'on n'écrirait pas de mémoire : laisser passer les réponses aux connexions déjà ouvertes, la boucle locale, les messages ICMP utiles, rejeter les paquets mal formés. C'est tout l'intérêt de l'outil.

Vos propres règles se retrouvent dans la chaîne `ufw-user-input`. On y lit la structure de nftables : une **table** contient des **chaînes**, une chaîne contient des **règles**, et chaque règle se termine par un verdict, `accept` ou `drop`. L'avertissement de la première ligne est à prendre au sérieux : ces tables appartiennent à UFW, on ne les modifie pas à la main.

Une table à soi

Rien n'empêche d'ajouter sa propre table à côté de celles d'UFW. C'est utile pour ce qu'UFW ne sait pas faire, par exemple tenir une liste d'adresses bloquées pour une durée limitée.

Voici `/etc/nftables.d/blocage.nft` :

```
#!/usr/sbin/nft -f
# Table indépendante de celles d'UFW : jette les paquets des adresses listées

table inet blocage
delete table inet blocage

table inet blocage {
    set scanners {
        type ipv4_addr
        flags timeout
    }

    chain entree {
        type filter hook input priority -10; policy accept;
        ip saddr @scanners counter drop
    }
}
```

Un **ensemble** (*set*) est une liste d'adresses que le noyau consulte très vite, même si elle en contient des milliers. La chaîne s'accroche à l'entrée, avec une priorité de −10 qui la fait passer avant les règles d'UFW. Tout paquet venant d'une adresse de l'ensemble est jeté.

```
gary@serveur:~$ sudo nft -c -f /etc/nftables.d/blocage.nft && echo "syntaxe correcte"
syntaxe correcte
gary@serveur:~$ sudo nft -f /etc/nftables.d/blocage.nft
gary@serveur:~$ sudo nft add element inet blocage scanners '{ 203.0.113.9 timeout 1h }'
gary@serveur:~$ sudo nft list table inet blocage
table inet blocage {
        set scanners {
                type ipv4_addr
                flags timeout
                elements = { 203.0.113.9 timeout 1h expires 59m59s618ms }
        }
```

```
        chain entree {
                type filter hook input priority filter - 10; policy accept;
                ip saddr @scanners counter packets 0 bytes 0 drop
        }
}
```

L'adresse sortira de la liste toute seule dans une heure. `nft -c` vérifie un fichier sans l'appliquer.

Cette table disparaît au redémarrage. Pour la recharger, une unité systemd de quelques lignes suffit, sur le modèle du chapitre 12 :

```
[Unit]
Description=Table nftables de blocage
After=ufw.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/sbin/nft -f /etc/nftables.d/blocage.nft
ExecStop=/usr/sbin/nft delete table inet blocage

[Install]
WantedBy=multi-user.target
```

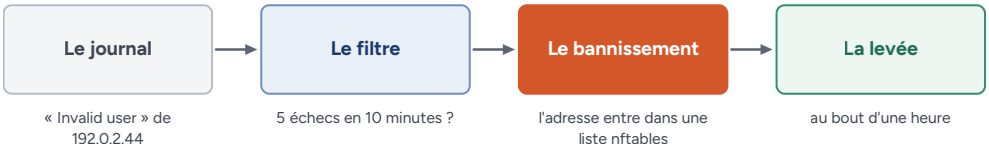
ATTENTION · Ne pas activer nftables.service à côté d'UFW

Le paquet `nftables` fournit un service du même nom, désactivé d'origine. Son fichier `/etc/nftables.conf` commence par la ligne `flush ruleset`, qui efface toutes les règles du noyau, celles d'UFW comprises. On choisit : UFW, complété au besoin par une table chargée à part, ou nftables seul, écrit de bout en bout.

21.4 Fail2ban

Le pare-feu laisse passer tout le monde vers le port 22, puisqu'il faut bien y entrer. Les robots en profitent pour essayer des noms de comptes à longueur de journée. Avec des clés, ils n'entreront pas. Mais ils remplissent les journaux et occupent le service.

Fail2ban surveille les journaux. Quand une adresse accumule les échecs, il la bannit : il ajoute une règle au pare-feu qui lui ferme le port, pour un temps donné.



Fail2ban lit les journaux et pilote le pare-feu.

Figure 21.3 — Le cycle d'un bannissement.

Il s'installe depuis le dépôt `universe`. Sur Ubuntu, la surveillance de SSH est active dès l'installation :

```
gary@serveur:~$ sudo apt install fail2ban
gary@serveur:~$ sudo fail2ban-client status
Status
|- Number of jail:      1
`- Jail list:   sshd
```

Fail2ban appelle *jail* l'association d'un journal à surveiller, d'un filtre qui reconnaît les échecs et d'une action de bannissement.

Régler

Les réglages d'origine se trouvent dans `/etc/fail2ban/jail.conf`. On ne modifie pas ce fichier, que les mises à jour remplacent. On écrit les siens dans `/etc/fail2ban/jail.local` :

```
[DEFAULT]
# Une heure de bannissement après cinq échecs en dix minutes
bantime  = 1h
findtime = 10m
maxretry = 5
# Les adresses qui ne seront jamais bannies
ignoreip = 127.0.0.1/8 ::1

[sshd]
enabled = true
```

Trois nombres définissent la sévérité : `maxretry` échecs pendant la durée `findtime` valent un bannissement de durée `bantime`. La ligne `ignoreip` liste les adresses qui ne seront jamais bannies. Ajoutez-y l'adresse fixe de votre bureau si vous en avez une, pour ne pas vous bannir vous-même un jour de maladresse.

```
gary@serveur:~$ sudo systemctl restart fail2ban
gary@serveur:~$ sudo fail2ban-client get sshd bantime
3600
```

Voir un bannissement

Depuis le poste, essayons cinq fois de suite un compte qui n'existe pas. Le serveur note chaque tentative :

```
Invalid user intrus from 192.168.1.1 port 57254
Connection closed by invalid user intrus 192.168.1.1 port 57254 [preauth]
```

Au cinquième échec, le poste est banni :

```
gary@vostro:~$ ssh intrus@192.168.1.50
ssh: connect to host 192.168.1.50 port 22: Connection refused
```

```
gary@serveur:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     5
|  `- Journal matches:  _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   192.168.1.1
```

Le journal de Fail2ban, `/var/log/fail2ban.log`, raconte la même chose : une ligne `Found` par échec, puis `[sshd] Ban 192.168.1.1`.

Fail2ban a créé sa propre table nftables, avec un ensemble d'adresses, exactement comme la nôtre :

```
table inet f2b-table {
        set addr-set-sshd {
                type ipv4_addr
                elements = { 192.168.1.1 }
        }

        chain f2b-chain {
                type filter hook input priority filter - 1; policy accept;
                tcp dport 22 ip saddr @addr-set-sshd reject with icmp port-
```

```
unreachable
        }
}
```

Pour lever un bannissement avant son terme :

```
sudo fail2ban-client set sshd unbanip 192.168.1.1
```

> **SUR LE TERRAIN · Qui a coupé la connexion ?**
> Lors du premier essai, le poste a été refusé dès la quatrième tentative, alors que Fail2ban n'en avait compté que trois. Le journal du noyau a donné l'explication : `[UFW LIMIT BLOCK]`. La règle `ufw limit` avait réagi la première. Quand deux protections se recouvrent, le journal dit laquelle a parlé. Cherchez avant de conclure.

Fail2ban sait surveiller bien d'autres services : serveurs web, messagerie, formulaires de connexion. Le dossier `/etc/fail2ban/filter.d/` contient près d'une centaine de filtres prêts à l'emploi.

Les limites

Un attaquant qui dispose de milliers d'adresses n'est pas gêné par le bannissement d'une seule. Fail2ban écarte le tout-venant, pas un adversaire décidé. La vraie protection de SSH reste l'authentification par clé du chapitre 19.

CrowdSec reprend l'idée en la mutualisant. Chaque machine équipée signale les adresses qui l'attaquent, et reçoit en retour la liste de celles que les autres ont vues. Il se trouve dans le dépôt `universe`.

Quant au blocage de pays entiers, il réduit le bruit dans les journaux, sans plus : un attaquant loue une machine dans le pays de son choix en quelques minutes. Et vous risquez de fermer la porte à un client en voyage, ou à vous-même.

En résumé

- Le noyau filtre les paquets. UFW écrit les règles à votre place.
- `ufw default deny incoming`, `ufw allow OpenSSH`, puis seulement `ufw enable`.
- Une règle par service, restreinte à un réseau d'origine chaque fois que c'est possible.

- La première règle qui correspond décide : `ufw insert 1` place un refus en tête.
- `journalctl -k -g 'UFW BLOCK'` montre ce qui est refusé. `nmap` vérifie de l'extérieur.
- `nft list ruleset` montre les règles réelles. On ne touche pas aux tables d'UFW, on peut ajouter la sienne.
- Fail2ban bannit les adresses qui accumulent les échecs. `fail2ban-client status sshd` montre où il en est.
- Ni le pare-feu ni Fail2ban ne remplacent des services bien configurés.

Atelier — Exposer un serveur web

Objectif : ne laisser entrer que SSH et le web, freiner les rafales, bannir les insistants, bloquer une liste d'adresses, et contrôler chaque point depuis une autre machine. Comptez quarante-cinq minutes.

Avant

1. Depuis le poste, relevez ce que le serveur expose : `nmap -Pn 192.168.1.50`. Notez les ports ouverts.
2. Sur le serveur, lancez un service d'essai sur le port 8000, comme au chapitre 19, cette fois sur toutes les interfaces : `python3 -m http.server 8000`. Vérifiez depuis le poste qu'il répond.

Le pare-feu

3. Posez la politique par défaut, autorisez `OpenSSH` et `Nginx Full`, contrôlez avec `sudo ufw show added`, puis activez.
4. Depuis le poste, vérifiez les trois comportements : SSH répond, `curl http://192.168.1.50/` répond, le port 8000 ne répond plus.
5. Retrouvez dans le journal du serveur la trace du refus.
6. Ouvrez le port 8000 au seul réseau local, vérifiez, puis supprimez la règle par son numéro et arrêtez le service d'essai.
7. Remplacez la règle SSH par `sudo ufw limit OpenSSH`.

Les insistants

8. Installez Fail2ban, créez `jail.local` sur le modèle du chapitre, et redémarrez le service.

9. Depuis le poste, tentez cinq connexions sous un nom de compte inexistant, en laissant une dizaine de secondes entre chacune. Constatez le bannissement des deux côtés, puis levez-le.

Si vous n'avez qu'un poste, faites cette étape depuis la console du serveur, pas par SSH : vous allez bannir votre propre adresse.

La liste noire

10. Créez la table `blocage` et son unité systemd. Ajoutez-y une adresse pour une heure, et affichez la table.
11. Redémarrez le serveur. Vérifiez que tout est revenu : `sudo ufw status`, `sudo fail2ban-client status`, `sudo nft list table inet blocage`.

Vérifier

- `nmap -Pn 192.168.1.50`, depuis le poste, ne montre plus que les ports 22 et 80.
- `sudo ufw status numbered` tient en quelques lignes, et vous savez justifier chacune.
- Relancez Lynis : l'avertissement sur l'absence de pare-feu a disparu. Sur le serveur du livre, l'indice est passé à 71, sans aucun avertissement.

Exercices

Exercice 1 — Lire des règles

Voici la sortie de `ufw status numbered` sur un serveur :

```
[ 1] 22/tcp                     ALLOW IN    Anywhere
[ 2] 80,443/tcp                 ALLOW IN    Anywhere
[ 3] Anywhere                   DENY IN     198.51.100.20
[ 4] 3306/tcp                   ALLOW IN    10.0.5.0/24
```

1. L'adresse `198.51.100.20` peut-elle atteindre le site web ? Pourquoi ?
2. Que faut-il faire pour qu'elle ne le puisse plus ?
3. Qui peut joindre la base de données ?
4. Que devient une connexion vers le port 8080 ?

Exercice 2 — Écrire des règles

Donnez les commandes UFW pour :

1. autoriser le port 51820 en UDP ;
2. autoriser PostgreSQL depuis la seule machine `10.0.5.12` ;
3. refuser tout ce qui vient du réseau `203.0.113.0/24`, en priorité sur les autres règles ;
4. supprimer la règle qui ouvre le port 8080.

Exercice 3 — Diagnostic

1. Après `sudo ufw enable`, un site hébergé sur le serveur ne répond plus. Quelles commandes lancez-vous, sur le serveur et depuis votre poste ?
2. `nmap` annonce un port `closed`, un autre `filtered`. Que signifie chaque mot ?
3. Vous ne pouvez plus vous connecter en SSH depuis le bureau, alors qu'un collègue y arrive depuis chez lui. Que soupçonnez-vous, et comment le vérifier ?

Exercice 4 — Fail2ban

1. Combien d'adresses sont bannies en ce moment sur votre serveur ? Combien l'ont été depuis le démarrage du service ?
2. Portez le bannissement à vingt-quatre heures et le nombre d'essais à trois. Vérifiez avec `fail2ban-client get` .
3. Empêchez que le réseau `192.168.1.0/24` soit jamais banni.
4. Où lit-on l'historique des bannissements ?

Exercice 5 — nftables

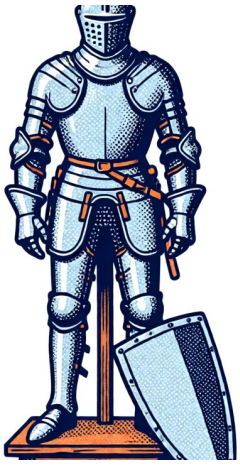
1. Combien de tables le noyau contient-il sur votre serveur ? À qui appartient chacune ?
2. Dans quelle chaîne se trouvent les règles que vous avez écrites avec UFW ?
3. Ajoutez deux adresses à l'ensemble `scanners`, l'une pour dix minutes, l'autre pour un jour. Affichez le temps qu'il leur reste.
4. Pourquoi la chaîne de la table `blocage` a-t-elle une priorité de −10 ?

Pour aller plus loin

Sur la machine. `man ufw` est court et complet. `man nft` décrit toute la syntaxe de nftables. Les fichiers `/etc/ufw/before.rules` et `/etc/ufw/after.rules` contiennent le socle de règles d'UFW, en syntaxe iptables.

En ligne. Le wiki `wiki.nftables.org` explique nftables pas à pas, avec des exemples. La documentation de CrowdSec, sur `docs.crowdsec.net`, décrit son installation à côté d'un pare-feu existant.

La suite. Le pare-feu contrôle ce qui atteint les services. Le chapitre 22 s'occupe de ce qu'un service a le droit de faire une fois atteint : c'est le rôle d'AppArmor.



CHAPITRE 22

AppArmor et le confinement

Les droits Unix disent ce qu'un utilisateur peut faire. AppArmor dit ce qu'un programme peut faire, quel que soit l'utilisateur, root compris. Un service compromis reste alors enfermé dans ce que son profil avait prévu.

DANS CE CHAPITRE

- Expliquer ce qu'un contrôle d'accès obligatoire ajoute aux droits Unix.
- Lire l'état d'AppArmor et repérer les programmes confinés.
- Lire un profil.
- Trouver et comprendre un refus dans les journaux.
- Passer un profil du mode d'apprentissage au mode d'application.
- Comprendre la restriction des espaces de noms d'Ubuntu 24.04.
- Écrire un profil pour son propre service et le lui attacher avec systemd.

> **AVANT DE COMMENCER**
>
> Les chapitres 6, 12 et 20. Le serveur fait tourner le service `veilleur` du chapitre 12, dans sa version durcie. AppArmor a besoin d'un vrai noyau : faites les manipulations dans une machine virtuelle, pas dans un conteneur.

22.1 Un second contrôle

Les droits des fichiers, vus au chapitre 6, reposent sur une idée : un programme agit au nom de l'utilisateur qui le lance, avec tous ses droits. Un serveur web lancé par `root` peut donc lire `/etc/shadow`, alors qu'il n'a rien à y faire. Si une faille permet de le détourner, l'attaquant hérite de tout.

Un **contrôle d'accès obligatoire** ajoute une seconde vérification, qui porte sur le programme et non sur l'utilisateur. Chaque programme confiné a un **profil** : la liste de ce qu'il a le droit de toucher. Le noyau la fait respecter, et personne n'y échappe.

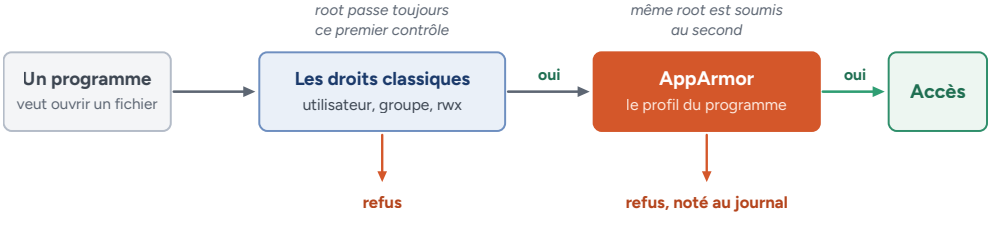


Figure 22.1 — Deux contrôles successifs.

Ubuntu utilise pour cela **AppArmor**. Ses profils désignent les fichiers par leur chemin, ce qui les rend lisibles.

> **REPÈRE HISTORIQUE · D'Immunix à Canonical**
>
> AppArmor est né à la fin des années 1990 sous le nom de SubDomain, dans la distribution Immunix. Novell l'a racheté en 2005 et lui a donné son nom actuel. Canonical en assure le développement depuis 2009, et l'active d'origine dans Ubuntu depuis la version 7.10. Red Hat a fait un autre choix : SELinux, issu des travaux de la NSA, qui étiquette chaque fichier au lieu de se fier aux chemins. Il est plus fin, et nettement plus difficile à régler.

22.2 L'état des lieux

```
gary@serveur:~$ aa-enabled
Yes
gary@serveur:~$ sudo aa-status | head -n 3
apparmor module is loaded.
117 profiles are loaded.
23 profiles are in enforce mode.
gary@serveur:~$ sudo aa-status | grep -A 1 'processes are in enforce'
1 processes are in enforce mode.
   /usr/sbin/rsyslogd (855) rsyslogd
```

Cent dix-sept profils sont chargés, et un seul processus est réellement confiné : rsyslog. Un profil chargé ne confine un programme que lorsque celui-ci est lancé, et la plupart des profils concernent des programmes qui ne tournent pas en ce moment.

Les profils sont des fichiers texte, rangés dans `/etc/apparmor.d/`. Les paquets y déposent les leurs.

`ps` montre le profil de chaque processus dans la colonne `LABEL` :

```
gary@serveur:~$ ps -eo pid,label,comm | grep -v unconfined | head -n 4
    PID LABEL                          COMMAND
    855 rsyslogd (enforce)             rsyslogd
```

À l'inverse, `aa-unconfined`, du paquet `apparmor-utils`, liste les programmes qui écoutent le réseau sans être confinés. Ce sont les premiers candidats à un profil :

```
gary@serveur:~$ sudo aa-unconfined
1 /usr/lib/systemd/systemd not confined
49104 /usr/sbin/sshd (sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups) not confined
49116 /usr/lib/systemd/systemd-resolved not confined
90570 /usr/sbin/nginx (nginx: master process […]) not confined
[…]
```

22.3 Lire un profil

Le profil de `tcpdump` se trouve dans `/etc/apparmor.d/usr.bin.tcpdump`. En voici l'essentiel, commentaires retirés :

```
profile tcpdump /usr/bin/tcpdump {
  #include <abstractions/base>
  #include <abstractions/nameservice>
  #include <abstractions/user-tmp>
```

```
  capability net_raw,
  capability setuid,
  capability setgid,
  […]
  network raw,
  network packet,
  […]
  /etc/ethers r,
  /{usr/,}bin/gzip ixr,
  […]
  audit deny @{HOME}/.* mrwkl,
  […]
  owner @{HOME}/** rw,
  /**.[pP][cC][aA][pP] rw,
  […]
  /usr/bin/tcpdump mr,
  /dev/pts/* rw,

  #include <local/usr.bin.tcpdump>
}
```

La première ligne nomme le profil et le programme auquel il s'applique. Chaque règle se termine par une virgule.

Règle	Signification
`capability net_raw,`	Le programme peut utiliser ce privilège de `root` : ici, capturer des paquets
`network raw,`	Il peut ouvrir ce type de connexion réseau
`/etc/ethers r,`	Il peut lire ce fichier
`owner @{HOME}/** rw,`	Il peut lire et écrire dans le répertoire personnel, pour les fichiers de l'utilisateur
`/**.[pP][cC][aA][pP] rw,`	Il peut lire et écrire partout les fichiers dont le nom finit par `.pcap`
`/{usr/,}bin/gzip ixr,`	Il peut exécuter `gzip`, qui héritera du même profil
`audit deny @{HOME}/.* mrwkl,`	Refus explicite, toujours noté au journal

Les lettres qui suivent un chemin sont les permissions : `r` pour lire, `w` pour écrire, `m` pour charger en mémoire du code exécutable, `x` pour exécuter, `k` pour verrouiller, `l` pour créer un lien. Dans les chemins, `*` remplace une suite de caractères dans un nom, `**` traverse aussi les sous-dossiers, et les accolades proposent un choix : `/{usr/,}bin/gzip` vaut pour `/usr/bin/gzip` comme pour `/bin/gzip`.

La règle fondamentale n'est écrite nulle part : tout ce qui n'est pas autorisé est refusé.

Les lignes `#include` ne sont pas des commentaires, malgré le dièse. C'est l'ancienne écriture de `include`, que les profils récents écrivent sans dièse. Elles importent des blocs de règles communes, rangés dans `/etc/apparmor.d/abstractions/` : `base` pour ce dont tout programme a besoin, `nameservice` pour la résolution des noms, `user-tmp` pour les fichiers temporaires. La dernière, `local/usr.bin.tcpdump`, est réservée à vos ajouts. Nous y revenons plus bas.

22.4 Voir un refus

`tcpdump` sait relire une capture avec `-r`. Demandons-lui de lire un fichier qui n'en est pas une :

```
gary@serveur:~$ sudo tcpdump -r /etc/shadow
tcpdump: /etc/shadow: Permission denied
gary@serveur:~$ sudo cat /etc/shadow | wc -l
36
```

La commande est lancée par `root`. `cat` lit le fichier sans difficulté, `tcpdump` se le voit refuser. Ce ne sont pas les droits du fichier qui ont parlé : c'est le profil, qui n'a prévu nulle part que `tcpdump` lise `/etc/shadow`.

Le refus est consigné :

```
gary@serveur:~$ sudo grep 'apparmor="DENIED"' /var/log/audit/audit.log | tail -n 1
type=AVC msg=audit(…): apparmor="DENIED" operation="open" class="file"
  profile="tcpdump" name="/etc/shadow" pid=132454 comm="tcpdump"
  requested_mask="r" denied_mask="r" […]
```

Champ	Ce qu'il dit
`apparmor="DENIED"`	Le verdict
`operation="open"`	L'opération tentée
`profile="tcpdump"`	Le profil qui a refusé
`name="/etc/shadow"`	Le fichier visé
`requested_mask="r"`	La permission demandée

Avec ces cinq champs, on sait quelle règle ajouterait l'accès. Quand un programme confiné échoue sur un « Permission denied » que les droits du fichier n'expliquent pas, c'est la ligne à chercher.

SUR LE TERRAIN · Où sont les refus ?

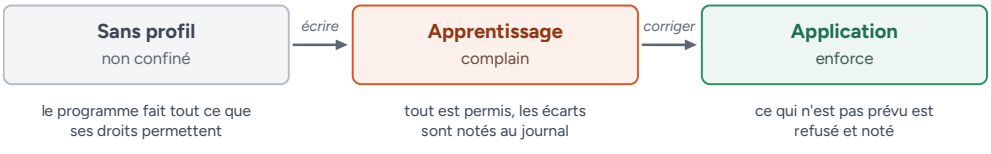
Sur ce serveur, ils se trouvent dans `/var/log/audit/audit.log`, parce que auditd a été installé au chapitre 20 et reçoit tous les messages de ce type. Sans auditd, ils vont dans le journal du noyau :

```
gary@serveur:~$ sudo journalctl -k -g 'apparmor="DENIED"' -n 1 -o cat
audit: type=1400 audit(…): apparmor="DENIED" operation="open"
class="file"
  profile="tcpdump" […]
```

Si une recherche ne donne rien, essayez l'autre.

22.5 Les modes

Un profil s'applique de deux façons.



On met un profil au point en mode d'apprentissage, puis on l'applique.

Figure 22.2 — Les trois situations d'un programme.

En mode **enforce**, ce qui n'est pas prévu est refusé. En mode **complain**, tout est permis, mais chaque écart est noté. C'est le mode de mise au point. Les commandes `aa-complain` et `aa-enforce`, du paquet `apparmor-utils`, font passer de l'un à l'autre :

```
gary@serveur:~$ sudo aa-complain /usr/bin/tcpdump
Setting /usr/bin/tcpdump to complain mode.
Warning: profile tcpdump represents multiple programs
Warning: profile tcpdump represents multiple programs
gary@serveur:~$ sudo tcpdump -r /etc/shadow 2>&1 | tail -n 1
tcpdump: unknown file format
gary@serveur:~$ sudo grep 'apparmor="ALLOWED"' /var/log/audit/audit.log | tail -n 1
type=AVC msg=audit(…): apparmor="ALLOWED" operation="open" class="file"
  profile="tcpdump" […]
```

```
gary@serveur:~$ sudo aa-enforce /usr/bin/tcpdump
Setting /usr/bin/tcpdump to enforce mode.
Warning: profile tcpdump represents multiple programs
gary@serveur:~$ sudo tcpdump -r /etc/shadow 2>&1 | tail -n 1
tcpdump: /etc/shadow: Permission denied
```

En mode complain, `tcpdump` a pu ouvrir le fichier, qu'il n'a pas su interpréter, et le journal porte la mention `ALLOWED`. Revenu en mode enforce, il est de nouveau refusé. L'avertissement est sans gravité : il rappelle que le profil porte un nom, `tcpdump`, qui n'est pas le chemin du programme.

Commande	Effet
`sudo aa-status`	L'état de tous les profils et des processus confinés
`sudo aa-complain programme`	Passer un profil en mode d'apprentissage
`sudo aa-enforce programme`	Le passer en mode d'application
`sudo aa-disable programme`	Le désactiver
`sudo apparmor_parser -r fichier`	Recharger un profil après l'avoir modifié
`sudo systemctl reload apparmor`	Recharger tous les profils

Ajuster un profil livré par un paquet

Quand un profil livré par un paquet gêne un usage légitime, on ne le désactive pas. On ne le modifie pas non plus : une mise à jour du paquet pourrait remettre en cause vos retouches. On ajoute la règle manquante dans le fichier du même nom sous `/etc/apparmor.d/local/`, que le profil inclut, puis on recharge. Supposons qu'on veuille laisser `tcpdump` lire `/etc/hostname` :

```
gary@serveur:~$ sudo tcpdump -r /etc/hostname
tcpdump: /etc/hostname: Permission denied
gary@serveur:~$ echo '/etc/hostname r,' | sudo tee -a /etc/apparmor.d/local/usr.bin.tcpdump
/etc/hostname r,
gary@serveur:~$ sudo apparmor_parser -r /etc/apparmor.d/usr.bin.tcpdump
gary@serveur:~$ sudo tcpdump -r /etc/hostname
tcpdump: unknown file format
```

Le refus a disparu : `tcpdump` ouvre le fichier, et constate que ce n'est pas une capture. Pour revenir en arrière, on retire la ligne et on recharge de nouveau.

22.6 Les espaces de noms sous Ubuntu 24.04

Ubuntu 24.04 se sert d'AppArmor pour une restriction nouvelle. Un *espace de noms d'utilisateur* permet à un programme ordinaire de se croire `root` dans un monde à lui. Les conteneurs et les navigateurs s'en servent pour s'isoler. Les attaquants aussi, car cet accès ouvre des parties du noyau où l'on a trouvé beaucoup de failles.

Depuis la 24.04, un programme sans profil peut toujours créer un tel espace, mais il n'y reçoit plus aucun privilège, ce qui le prive de tout intérêt :

```
gary@serveur:~$ sysctl kernel.apparmor_restrict_unprivileged_userns
kernel.apparmor_restrict_unprivileged_userns = 1
gary@serveur:~$ unshare --user --map-root-user true
unshare: write failed /proc/self/uid_map: Operation not permitted
```

`unshare` a créé l'espace, puis s'est vu refuser d'y devenir `root`.

Les logiciels qui ont besoin de ce mécanisme reçoivent une autorisation nominative. C'est le rôle des quatre-vingt-dix profils que `aa-status` range dans la catégorie *unconfined* : ils ne restreignent rien, et portent la règle `userns,`. Celui de Firefox, `/etc/apparmor.d/firefox`, tient en trois lignes.

La conséquence se voit surtout sur un poste de travail. Une application téléchargée hors des dépôts, au format AppImage ou bâtie sur Electron, peut refuser de démarrer, avec un message qui mentionne le *sandbox*. Le remède propre est de lui écrire un profil du même genre. Pour une application installée dans `/opt/monappli`, le fichier `/etc/apparmor.d/monappli` contiendrait :

```
abi <abi/4.0>,
include <tunables/global>

profile monappli /opt/monappli/monappli flags=(unconfined) {
  userns,
}
```

Faisons l'essai avec une copie de `unshare` placée à cet endroit :

```
gary@serveur:~$ /opt/monappli/monappli --user --map-root-user id -u
monappli: write failed /proc/self/uid_map: Operation not permitted
gary@serveur:~$ sudo apparmor_parser -r /etc/apparmor.d/monappli
gary@serveur:~$ /opt/monappli/monappli --user --map-root-user id -u
0
```

Une fois le profil chargé, le programme obtient son espace de noms et s'y voit `root`, sous le numéro 0. Lui seul a gagné ce droit.

L'autre remède, que proposent beaucoup de forums, consiste à remettre le paramètre du noyau à 0. Il lève la restriction pour tous les programmes de la machine. C'est se priver d'une protection pour régler le cas d'une seule application.

> **NOTE · Une couche, pas un rempart**
>
> Des chercheurs ont publié en 2025 plusieurs moyens de contourner cette restriction. Elle complique le travail d'un attaquant, elle ne le rend pas impossible. C'est vrai de chaque mesure de ce chapitre et du chapitre 20 : on les empile parce qu'aucune ne suffit seule.

22.7 Écrire un profil

Le service `veilleur` du chapitre 12 tourne sous un compte à lui, dans un espace cloisonné par systemd. Ajoutons-lui un profil AppArmor. Il lit son script et la charge de la machine, il écrit dans `/var/lib/veilleur` : le profil ne dira rien de plus.

Un premier jet, en apprentissage

Le script est exécuté par l'interpréteur Python. Un profil attaché à `/usr/bin/python3` confinerait tous les programmes Python de la machine. On écrit donc un profil sans chemin, que systemd attachera à ce seul service. Voici `/etc/apparmor.d/veilleur`, dans sa première version :

```
abi <abi/4.0>,
include <tunables/global>

profile veilleur flags=(complain) {
  include <abstractions/base>
  include <abstractions/python>

  # l'interpréteur et le script
  /usr/bin/python3* mr,
  /opt/veilleur/veilleur.py r,

  # ce que le script lit et écrit
  @{PROC}/loadavg r,
  /var/lib/veilleur/ r,
  /var/lib/veilleur/** rw,
}
```

`flags=(complain)` le place en mode d'apprentissage. On le charge, puis on demande à systemd de l'appliquer au service, par un fichier de surcharge :

```
gary@serveur:~$ sudo apparmor_parser -r /etc/apparmor.d/veilleur
gary@serveur:~$ cat /etc/systemd/system/veilleur.service.d/apparmor.conf
[Service]
AppArmorProfile=veilleur
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl restart veilleur
gary@serveur:~$ sudo cat /proc/$(systemctl show -p MainPID --value veilleur)/attr/current
veilleur (complain)
```

Ce que le journal révèle

Laissons le service tourner quelques instants, puis lisons ce qu'AppArmor a noté :

```
gary@serveur:~$ sudo grep 'profile="veilleur"' /var/log/audit/audit.log | \
    grep -oE 'apparmor="[A-Z]+" operation="[a-z]+".*name="[^"]+"' | sort -u
apparmor="ALLOWED" operation="getattr" class="file"
  info="Failed name lookup - disconnected path" error=-13
  profile="veilleur" name="dev/null"
apparmor="ALLOWED" operation="open" class="file" profile="veilleur"
  name="/opt/veilleur/"
```

Deux oublis. Python ouvre le dossier du script, et pas seulement le script : il manque `/opt/veilleur/ r,`. Le second message est plus déroutant. Le service vit dans un espace de montage privé, créé par les directives de cloisonnement de son unité, et AppArmor ne sait plus rattacher `/dev/null` à un chemin complet. L'option `attach_disconnected` règle ce cas, fréquent avec les services durcis.

En mode enforce, le service aurait échoué au démarrage sur ces deux points. C'est tout l'intérêt de l'apprentissage : on corrige avant d'appliquer.

La version finale

```
abi <abi/4.0>,
include <tunables/global>

profile veilleur flags=(attach_disconnected) {
  include <abstractions/base>
  include <abstractions/python>

  # l'interpréteur et le script
  /usr/bin/python3* mr,
  /opt/veilleur/ r,
  /opt/veilleur/veilleur.py r,

  # ce que le script lit et écrit
  @{PROC}/loadavg r,
```

```
  /var/lib/veilleur/ r,
  /var/lib/veilleur/** rw,
}
```

```
gary@serveur:~$ sudo apparmor_parser -r /etc/apparmor.d/veilleur
gary@serveur:~$ sudo systemctl restart veilleur
gary@serveur:~$ systemctl is-active veilleur
active
gary@serveur:~$ sudo aa-status | grep -A 2 'processes are in enforce'
2 processes are in enforce mode.
   /usr/sbin/rsyslogd (855) rsyslogd
   /usr/bin/python3.12 (132628) veilleur
```

Le service tourne, confiné. `aa-exec` lance une commande sous un profil donné : c'est le moyen de vérifier ce que le profil permet.

```
gary@serveur:~$ sudo aa-exec -p veilleur -- python3 -c "print(open('/proc/loadavg').read())"
0.00 0.03 0.02 1/176 132643
gary@serveur:~$ sudo aa-exec -p veilleur -- python3 -c "print(open('/etc/hostname').read())"
[…]
PermissionError: [Errno 13] Permission denied: '/etc/hostname'
[…]
```

Sous ce profil, Python lit la charge de la machine, et se voit refuser un fichier aussi anodin que `/etc/hostname`. Une faille dans le script ne donnerait accès à rien d'autre qu'à son propre dossier de données.

L'erreur s'accompagne d'une longue trace, coupée ici. Elle vient du mécanisme d'Ubuntu qui prépare un rapport d'incident quand un programme Python plante : lui aussi tourne sous le profil, et se voit refuser les fichiers qu'il voulait consulter.

Une erreur de syntaxe dans un profil est signalée au chargement. Voici ce que donne une virgule oubliée à la fin de la ligne 14, celle de `loadavg` :

```
gary@serveur:~$ sudo apparmor_parser -r /etc/apparmor.d/veilleur
AppArmor parser error for /etc/apparmor.d/veilleur in profile
  /etc/apparmor.d/veilleur at line 15: missing an end of line character?
  (entry: @{PROC}/loadavg)
```

L'analyseur ne s'aperçoit de l'oubli qu'à la ligne suivante : c'est donc la ligne du dessus qu'il faut regarder. Tant que le fichier est fautif, l'ancienne version du profil reste en vigueur dans le noyau.

> **NOTE · Deux cloisons valent mieux qu'une**
>
> Le service est maintenant protégé deux fois. Les directives de systemd lui cachent une partie du système de fichiers. Le profil AppArmor énumère ce qu'il peut toucher dans ce qui reste. Les deux mécanismes sont indépendants : une erreur dans l'un est rattrapée par l'autre. Les conteneurs du chapitre 28 combinent les mêmes briques, et Docker applique d'office un profil AppArmor à chacun d'eux.

Se faire aider

Pour un programme plus complexe, écrire le profil à la main serait fastidieux. Deux outils du paquet `apparmor-utils` y aident. `aa-genprof programme` crée un profil vide en mode d'apprentissage, vous demande d'utiliser le programme dans un autre terminal, puis vous soumet chaque accès observé : autoriser, refuser, généraliser. `aa-logprof` refait ce travail plus tard, à partir des journaux, pour compléter un profil existant.

Leurs propositions demandent à être relues. Un profil qui autorise `/** rw,` ne protège plus de rien.

En résumé

- AppArmor ajoute aux droits Unix un contrôle par programme, auquel `root` est soumis.
- `aa-status` montre les profils chargés et les processus confinés. `aa-unconfined` montre les services réseau qui ne le sont pas.
- Un profil liste ce qui est permis. Le reste est refusé.
- Un refus laisse une ligne `apparmor="DENIED"`, dans `/var/log/audit/audit.log` ou dans le journal du noyau.
- Mode complain pour mettre au point, mode enforce pour protéger. `aa-complain`, `aa-enforce`.
- Les ajustements d'un profil livré par un paquet vont dans `/etc/apparmor.d/local/`.
- Sur 24.04, les programmes sans profil ne créent plus d'espaces de noms d'utilisateur.
- `AppArmorProfile=` dans une unité systemd attache un profil à un service.

Atelier — Confiner son propre service

Objectif : donner un profil AppArmor au service `veilleur`, en passant par le mode d'apprentissage, et prouver qu'il le confine. Comptez quarante-cinq minutes.

Préparer

1. Vérifiez que le service tourne et relevez son état : `systemctl is-active veilleur`, puis `ps -eo pid,label,comm | grep python3`. La colonne `LABEL` doit afficher `unconfined`.
2. Installez les outils : `sudo apt install apparmor-utils`.

Apprendre

3. Créez `/etc/apparmor.d/veilleur` avec le premier jet du chapitre, en mode complain, et chargez-le avec `apparmor_parser -r`.
4. Créez la surcharge `apparmor.conf` avec `sudo systemctl edit --drop-in=apparmor veilleur`, et redémarrez le service.
5. Vérifiez que le processus porte bien le profil, en mode complain.
6. Attendez une minute, puis relevez dans le journal les lignes qui concernent le profil. Retrouvez-vous les deux écarts du chapitre ? En voyez-vous d'autres ?

Appliquer

7. Corrigez le profil : ajoutez les règles manquantes, l'option `attach_disconnected`, et retirez `complain`.
8. Rechargez le profil, redémarrez le service, vérifiez qu'il est actif et qu'il écrit toujours ses relevés.

Prouver

9. Avec `aa-exec -p veilleur`, faites lire à Python un fichier permis, puis un fichier interdit.
10. Retrouvez dans le journal la ligne `DENIED` correspondante. Quelle règle faudrait-il ajouter pour autoriser cette lecture ? Ne l'ajoutez pas.

Casser, pour voir

11. Retirez du profil la ligne qui autorise l'écriture dans `/var/lib/veilleur`, rechargez et redémarrez. Que devient le service ? Que disent `systemctl status veilleur` et le journal d'AppArmor ? Remettez la ligne.

Vérifier

- `sudo aa-status` compte un processus confiné de plus, sous le profil `veilleur`.
- Le fichier `/var/lib/veilleur/charge.csv` continue de grandir.
- Vous savez retrouver un refus d'AppArmor et dire quelle règle il réclame.

Exercices

Exercice 1 — État des lieux

1. Combien de profils sont chargés sur votre machine ? Combien en mode enforce ?
2. Quels processus sont confinés en ce moment ?
3. Quels services réseau ne le sont pas ?
4. Où se trouve le profil de `man` ? Peut-il écrire dans votre répertoire personnel ?

Exercice 2 — Lire des règles

Que permet chacune de ces lignes ?

```
/var/log/monappli/*.log w,
/etc/monappli/** r,
owner @{HOME}/.config/monappli/ rw,
network inet stream,
/usr/bin/convert ix,
deny /etc/shadow r,
```

Exercice 3 — Lire un refus

```
apparmor="DENIED" operation="open" class="file" profile="tcpdump"
  name="/etc/hostname" pid=133102 comm="tcpdump" requested_mask="r"
  denied_mask="r"
```

1. Quel programme a été refusé, et que tentait-il ?
2. Reproduisez ce refus sur votre machine et retrouvez la ligne dans le journal.
3. Copiez `/etc/hostname` sous le nom `/etc/hostname.pcap` et recommencez avec la copie. Le message change. Quelle règle du profil l'explique ?
4. Supprimez la copie.

Exercice 4 — Modes

1. Passez le profil de `tcpdump` en mode complain, relancez la commande refusée, puis revenez en mode enforce.
2. Quelle trace le mode complain a-t-il laissée ?
3. Pourquoi ne faut-il pas laisser un profil en mode complain sur un serveur en service ?

Exercice 5 — Espaces de noms

1. Quelle est la valeur de `kernel.apparmor_restrict_unprivileged_userns` sur votre machine ?
2. Lancez `unshare --user --map-root-user true`. Que se passe-t-il ?
3. Combien de profils de `/etc/apparmor.d/` portent l'indication `flags=(unconfined)` ? À quoi servent-ils ?
4. Un forum conseille de mettre ce paramètre à 0 pour faire fonctionner une application. Que proposez-vous à la place ?

Pour aller plus loin

Sur la machine. `man apparmor.d` décrit toute la syntaxe des profils. `man aa-status`, `man aa-exec`, `man apparmor_parser` pour les outils. Le dossier `/etc/apparmor.d/abstractions/` mérite une visite : on y voit comment sont découpées les règles communes.

Paquets. `apparmor-profiles` et `apparmor-profiles-extra` apportent des profils supplémentaires pour des logiciels courants, à étudier et à essayer en mode complain avant de les appliquer.

En ligne. Le wiki du projet, sur `gitlab.com/apparmor/apparmor/-/wikis`, contient un guide d'écriture des profils. La page consacrée à AppArmor dans la documentation d'Ubuntu Server en donne une présentation plus courte.

PARTIE VIII

Services serveur essentiels

Déployer les services que l'on attend d'un serveur : sites web, bases de données, partage de fichiers, supervision.





CHAPITRE 23

Serveurs web

Un serveur web reçoit une demande et renvoie une page. Tout le reste est affaire de configuration, c'est-à-dire savoir quel site répond à quel nom, où sont les fichiers, à quelle application passer la main, avec quel certificat. Ce chapitre installe deux sites sur le même serveur et les fait passer en HTTPS.

DANS CE CHAPITRE

- Lire et écrire une configuration Nginx.
- Servir un site statique.
- Placer Nginx devant une application, en relais.
- Héberger plusieurs sites sur une seule machine.
- Obtenir un certificat Let's Encrypt et le renouveler sans y penser.
- Régler HTTP/2, la compression et le cache.
- Lire les journaux d'accès et d'erreurs.
- Retrouver ses marques dans Apache, Caddy et Traefik.

> **AVANT DE COMMENCER**
>
> Les chapitres 12, 17 et 21. Le serveur fait tourner Nginx, installé au chapitre 12, et le pare-feu laisse passer les ports 80 et 443 depuis le chapitre 21. Le service `veilleur` du chapitre 12 fournit les données de l'application d'exemple. Les essais se font depuis le poste `vostro`.

23.1 Demande et réponse

Quand vous tapez une adresse dans un navigateur, celui-ci ouvre une connexion vers le port 80 ou 443 de la machine visée et lui envoie quelques lignes de texte : la page voulue, le nom du site, la langue préférée. Le serveur répond par un code à trois chiffres, quelques lignes d'en-tête, puis le contenu. C'est le protocole **HTTP**. `curl -I` montre le début d'une réponse :

```
gary@serveur:~$ curl -sI http://localhost
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Sun, 04 Oct 2026 23:37:40 GMT
Content-Type: text/html
Content-Length: 615
[…]
```

Le code 200 signifie que tout va bien. Vous en croiserez d'autres dans ce chapitre.

Code	Sens
200	La page est servie
301	La page a déménagé, voici sa nouvelle adresse
403	Accès refusé
404	Page introuvable
502	Le serveur web n'a pas pu joindre l'application qu'il relaie

Un site est **statique** quand le serveur se contente d'envoyer des fichiers tels qu'ils sont sur le disque. Il est **dynamique** quand un programme fabrique la réponse à chaque demande. Le serveur web sait faire le premier travail seul. Pour le second, il passe la main à une application.

REPÈRE HISTORIQUE · Trente ans de serveurs web

Apache naît en avril 1995, d'une série de correctifs appliqués au serveur du NCSA, alors à l'abandon : *a patchy server*, un serveur rapiécé. Il a longtemps été le serveur le plus répandu. En octobre 2004, le Russe Igor Sysoev publie Nginx, conçu pour tenir des milliers de connexions à la fois avec peu de mémoire. Ces deux-là restent les plus utilisés. Une génération plus récente, écrite en Go, mise sur l'automatisation : Caddy, qui obtient ses certificats tout seul, et Traefik, né en 2015 pour suivre des conteneurs qui apparaissent et disparaissent. Citons enfin OpenLiteSpeed, version libre du serveur LiteSpeed.

23.2 Nginx sur Ubuntu

Nginx est installé depuis le chapitre 12 (`sudo apt install nginx`). Vérifions qu'il tourne et qu'il écoute :

```
gary@serveur:~$ nginx -v
nginx version: nginx/1.24.0 (Ubuntu)
gary@serveur:~$ systemctl is-active nginx
active
gary@serveur:~$ sudo ss -ltnp | grep nginx
LISTEN 0      511          0.0.0.0:80        0.0.0.0:*    users:(("nginx",pid=90574,…
LISTEN 0      511             [::]:80           [::]:*    users:(("nginx",pid=90574,…
```

Il écoute le port 80, en IPv4 et en IPv6. Sa configuration tient dans `/etc/nginx` :

```
gary@serveur:~$ ls /etc/nginx
conf.d          koi-win             nginx.conf       sites-enabled
fastcgi.conf    mime.types          proxy_params     snippets
fastcgi_params  modules-available   scgi_params      uwsgi_params
koi-utf         modules-enabled     sites-available  win-utf
```

Emplacement	Rôle
`nginx.conf`	Le fichier principal : réglages généraux
`sites-available/`	Un fichier par site, actif ou non
`sites-enabled/`	Des liens vers les sites actifs
`conf.d/`	Des réglages communs à tous les sites
`snippets/`	Des morceaux de configuration à inclure
`/var/www/`	Les fichiers des sites

Emplacement	Rôle
`/var/log/nginx/`	Les journaux

Un site est actif quand `sites-enabled` contient un lien vers son fichier. Pour le désactiver, on supprime le lien : le fichier reste dans `sites-available`. Ce rangement est propre à Debian et à Ubuntu.

23.3 Anatomie d'une configuration

Voici le fichier principal, commentaires retirés :

```
gary@serveur:~$ grep -vE '^\s*#|^\s*$' /etc/nginx/nginx.conf
user www-data;
worker_processes auto;
pid /run/nginx.pid;
error_log /var/log/nginx/error.log;
include /etc/nginx/modules-enabled/*.conf;
events {
        worker_connections 768;
}
http {
        sendfile on;
        tcp_nopush on;
        types_hash_max_size 2048;
        include /etc/nginx/mime.types;
        default_type application/octet-stream;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref:
POODLE
        ssl_prefer_server_ciphers on;
        access_log /var/log/nginx/access.log;
        gzip on;
        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

Une configuration Nginx est faite de **directives**, terminées par un point-virgule, et de **blocs** entre accolades, qui s'emboîtent. Le bloc `http` contient tout ce qui concerne le Web. Ses deux dernières lignes vont chercher les fichiers de `conf.d` et de `sites-enabled`. C'est là que se trouvent les sites, chacun dans un bloc `server`. Le site livré avec le paquet en donne un exemple :

```
gary@serveur:~$ grep -vE '^\s*#|^\s*$' /etc/nginx/sites-available/default
server {
        listen 80 default_server;
        listen [::]:80 default_server;
```

```
        root /var/www/html;
        index index.html index.htm index.nginx-debian.html;
        server_name _;
        location / {
                try_files $uri $uri/ =404;
        }
}
```

Directive	Rôle
`listen`	Le port écouté, une ligne pour IPv4 et une pour IPv6
`server_name`	Le ou les noms auxquels ce site répond
`root`	Le dossier où se trouvent les fichiers
`index`	Le fichier servi quand on demande un dossier
`location`	Des règles pour une partie des adresses
`try_files`	Essayer le fichier, puis le dossier, sinon répondre 404

Les blocs s'emboîtent sur trois niveaux : `http` pour tout le serveur, `server` pour un site, `location` pour une partie d'un site. Une directive posée à un niveau vaut pour tout ce qu'il contient, sauf si un niveau plus profond la redéfinit.

`worker_processes auto` lance un processus de travail par processeur. Le processus maître appartient à `root`, parce qu'il faut ce privilège pour ouvrir le port 80. Les processus de travail, qui traitent les demandes, tournent sous le compte `www-data` :

```
gary@serveur:~$ ps -o user,pid,cmd -C nginx
USER         PID CMD
root       90570 nginx: master process /usr/sbin/nginx -g daemon on; master_pr
www-data  135481 nginx: worker process
www-data  135482 nginx: worker process
```

Retenez ce compte. Un fichier que `www-data` ne peut pas lire ne sera jamais servi.

23.4 Servir un site statique

Notre premier site, un carnet, répondra au nom `blog.maison.lan`. Il lui faut un dossier, des fichiers et un bloc `server`.

```
gary@serveur:~$ sudo mkdir /var/www/blog
```

On y dépose une page et sa feuille de style :

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Le carnet du serveur</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Le carnet du serveur</h1>
  <p>Cette page est servie par Nginx depuis /var/www/blog.</p>
</body>
</html>
```

```
gary@serveur:~$ ls -l /var/www/blog
total 8
-rw-r--r-- 1 root root 272 Oct  4 23:37 index.html
-rw-r--r-- 1 root root 333 Oct  4 23:37 style.css
```

Les fichiers appartiennent à `root` et tout le monde peut les lire : `www-data` les servira, sans pouvoir les modifier. C'est le bon réglage pour un site statique.

Le site est décrit dans `/etc/nginx/sites-available/blog` :

```
server {
    listen 80;
    listen [::]:80;
    server_name blog.maison.lan;

    root /var/www/blog;
    index index.html;

    access_log /var/log/nginx/blog.access.log;
    error_log  /var/log/nginx/blog.error.log;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Il a ses propres journaux, ce qui évite de démêler plus tard les visites de plusieurs sites. Reste à l'activer. Trois gestes, toujours les mêmes :

```
gary@serveur:~$ sudo ln -s /etc/nginx/sites-available/blog /etc/nginx/sites-enabled/
gary@serveur:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
gary@serveur:~$ sudo systemctl reload nginx
```

`nginx -t` relit toute la configuration sans rien changer au serveur en marche. `reload` la fait appliquer sans couper les connexions en cours.

Essayer depuis le poste

Le nom `blog.maison.lan` n'existe dans aucun DNS. Sur un réseau d'essai, le plus simple est de l'inscrire dans le fichier `/etc/hosts` du poste, vu au chapitre 17. Ajoutez-y une ligne, avec le nom du second site que nous créerons plus loin :

```
gary@vostro:~$ grep maison.lan /etc/hosts
192.168.1.50 blog.maison.lan appli.maison.lan
gary@vostro:~$ curl -s http://blog.maison.lan | grep title
  <title>Le carnet du serveur</title>
gary@vostro:~$ curl -s http://192.168.1.50 | grep title
<title>Welcome to nginx!</title>
```

La même machine, à la même adresse, renvoie deux pages différentes. Tout dépend du nom demandé.

> **ASTUCE · Tester un site sans toucher au DNS**
>
> Le navigateur indique le nom du site dans un en-tête de sa demande, `Host`. `curl` peut l'écrire à votre place : `curl -H 'Host: blog.maison.lan' http://localhost` interroge le serveur comme si le nom était déjà en service. C'est le moyen de vérifier un site avant de faire pointer son nom vers la machine.

Quand ça ne marche pas

Trois pannes reviennent sans cesse. Provoquons-les.

La faute de frappe. Retirons le point-virgule qui termine la ligne `root` :

```
gary@serveur:~$ sudo nginx -t
2026/10/04 23:37:45 [emerg] 135459#135459: invalid number of arguments in "root"
  directive in /etc/nginx/sites-enabled/blog:7
nginx: configuration file /etc/nginx/nginx.conf test failed
gary@serveur:~$ sudo systemctl reload nginx
Job for nginx.service failed.
See "systemctl status nginx.service" and "journalctl -xeu nginx.service" for details.
gary@serveur:~$ systemctl is-active nginx
```

```
active
gary@serveur:~$ curl -sI -H 'Host: blog.maison.lan' http://localhost | head -n 1
HTTP/1.1 200 OK
```

Le message donne le fichier et la ligne. Surtout, le rechargement a échoué sans rien casser : Nginx a gardé l'ancienne configuration, et le site répond toujours. Voilà pourquoi on recharge au lieu de redémarrer, et pourquoi on lance `nginx -t` avant.

La page absente renvoie un code 404. Rien à chercher du côté du serveur : le fichier n'existe pas, ou l'adresse est mal écrite.

Le fichier illisible. Créons un fichier que seul `root` peut lire :

```
gary@serveur:~$ sudo cp /etc/hostname /var/www/blog/note.txt
gary@serveur:~$ sudo chmod 600 /var/www/blog/note.txt
gary@serveur:~$ curl -sI -H 'Host: blog.maison.lan' http://localhost/note.txt | head -n 1
HTTP/1.1 403 Forbidden
gary@serveur:~$ sudo tail -n 1 /var/log/nginx/blog.error.log
2026/10/04 23:37:49 [error] 135481#135481: *9 open() "/var/www/blog/note.txt" failed
  (13: Permission denied), client: ::1, server: blog.maison.lan, request: "HEAD
  /note.txt HTTP/1.1", host: "blog.maison.lan"
```

Le journal d'erreurs dit tout : le fichier, la cause, le site, la demande. Un `chmod 644` règle le problème. Devant un 403 ou un 500, ouvrez ce journal avant toute autre chose.

23.5 Relayer une application

Un site dynamique est produit par un programme : du Python, du PHP, du JavaScript avec Node.js. Ce programme sait répondre à des demandes HTTP, mais il n'a pas à s'occuper du chiffrement, des journaux ou des fichiers statiques. On le place derrière Nginx, qui joue le **relais**. Le terme anglais est *reverse proxy*.



Le client ne parle qu'à Nginx, qui transmet la demande et rapporte la réponse.

Figure 23.1 — Nginx en relais devant une application.

L'application n'écoute que l'adresse `127.0.0.1` : elle est injoignable depuis le réseau. Seul Nginx lui parle.

L'application

Le service `veilleur` note la charge du serveur toutes les dix secondes dans un fichier. Affichons ses derniers relevés dans une page web. Le programme tient en vingt lignes de Python, rangées dans `/opt/tableau/tableau.py` :

```
"""Tableau de la charge : les derniers relevés du service veilleur."""
from pathlib import Path

RELEVES = Path("/var/lib/veilleur/charge.csv")


def application(environ, start_response):
    lignes = RELEVES.read_text().splitlines()[-5:]
    client = environ.get("HTTP_X_REAL_IP", environ["REMOTE_ADDR"])
    texte = "Charge du serveur, cinq derniers relevés\n"
    texte += "\n".join(ligne.replace(";", "  ") for ligne in lignes)
    texte += f"\n\nDemandé par {client}\n"
    corps = texte.encode()
    start_response("200 OK", [
        ("Content-Type", "text/plain; charset=utf-8"),
        ("Content-Length", str(len(corps))),
    ])
    return [corps]
```

Il lui faut un serveur d'applications pour recevoir les demandes et les lui confier. En Python, le plus courant est Gunicorn (`sudo apt install gunicorn`). On en fait un service, dans `/etc/systemd/system/tableau.service` :

```
[Unit]
Description=Tableau de la charge (application web)
After=network.target

[Service]
DynamicUser=yes
WorkingDirectory=/opt/tableau
ExecStart=/usr/bin/gunicorn --bind 127.0.0.1:8000 --workers 2 tableau:application
Restart=on-failure
ProtectHome=yes
PrivateDevices=yes

[Install]
WantedBy=multi-user.target
```

`--bind 127.0.0.1:8000` fixe l'adresse d'écoute, `--workers 2` le nombre de processus. `DynamicUser=yes`, vu au chapitre 12, fait tourner le tout sous un compte créé pour l'occasion.

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl enable --now tableau
Created symlink /etc/systemd/system/multi-user.target.wants/tableau.service → …
gary@serveur:~$ sudo ss -ltnp | grep 8000
LISTEN 0      2048       127.0.0.1:8000      0.0.0.0:*    users:(("gunicorn",…
gary@serveur:~$ curl -s http://127.0.0.1:8000/
Charge du serveur, cinq derniers relevés
2026-10-04 23:38:10  0.05
2026-10-04 23:38:20  0.04
2026-10-04 23:38:30  0.12
2026-10-04 23:38:40  0.25
2026-10-04 23:38:50  0.37

Demandé par 127.0.0.1
```

L'application répond, sur la machine elle-même et nulle part ailleurs.

Le relais

Le second site, `appli.maison.lan`, tient en un bloc. Voici `/etc/nginx/sites-available/appli` :

```
server {
    listen 80;
    listen [::]:80;
    server_name appli.maison.lan;

    access_log /var/log/nginx/appli.access.log;
    error_log  /var/log/nginx/appli.error.log;

    location / {
        proxy_pass http://127.0.0.1:8000;
        include proxy_params;
    }
}
```

`proxy_pass` désigne l'application. La ligne `include proxy_params` va chercher un fichier livré avec le paquet :

```
gary@serveur:~$ cat /etc/nginx/proxy_params
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

Ces quatre en-têtes transmettent à l'application ce qu'elle ne peut plus voir. De son point de vue, toutes les demandes viennent de Nginx, donc de `127.0.0.1`. Nginx

lui indique le nom demandé, la véritable adresse du visiteur, et si celui-ci est arrivé en HTTP ou en HTTPS.

On active le site par les trois mêmes gestes, le lien, `nginx -t` et `reload`, puis on l'essaie depuis le poste :

```
gary@vostro:~$ curl -s http://appli.maison.lan
Charge du serveur, cinq derniers relevés
[…]

Demandé par 192.168.1.1
```

La dernière ligne a changé. L'application lit l'en-tête `X-Real-IP` et connaît l'adresse du poste. Sans `proxy_params`, elle afficherait `127.0.0.1` pour tous les visiteurs, et ses journaux ne vaudraient rien.

Quand l'application tombe

```
gary@serveur:~$ sudo systemctl stop tableau
gary@serveur:~$ curl -sI -H 'Host: appli.maison.lan' http://localhost/ | head -n 1
HTTP/1.1 502 Bad Gateway
gary@serveur:~$ sudo tail -n 1 /var/log/nginx/appli.error.log
2026/10/04 23:38:56 [error] 135757#135757: *18 connect() failed (111: Connection
  refused) while connecting to upstream, client: ::1, server: appli.maison.lan,
  request: "HEAD / HTTP/1.1", upstream: "http://127.0.0.1:8000/", host: […]
gary@serveur:~$ sudo systemctl start tableau
```

Un 502 ne signale presque jamais une panne de Nginx. Il dit que Nginx est en vie et que personne ne répond derrière lui. *Connection refused* : l'application est arrêtée, ou n'écoute pas à l'adresse indiquée. Le réflexe est `systemctl status` sur le service de l'application, puis son journal.

Le cas de PHP

PHP ne parle pas HTTP. Il passe par un service à part, PHP-FPM, que Nginx interroge dans un autre protocole, FastCGI.

```
gary@serveur:~$ sudo apt install php-fpm
[…]
gary@serveur:~$ systemctl is-active php8.3-fpm
active
gary@serveur:~$ ls -l /run/php/
total 4
```

```
lrwxrwxrwx 1 root     root     30 Oct  4 23:39 php-fpm.sock -> /etc/alternatives/…
-rw-r--r-- 1 root     root      6 Oct  4 23:39 php8.3-fpm.pid
srw-rw---- 1 www-data www-data  0 Oct  4 23:39 php8.3-fpm.sock
```

PHP-FPM n'écoute pas un port, mais un socket Unix, un fichier spécial réservé à `www-data`. Dans le bloc `server` du site, on ajoute une règle pour les adresses qui finissent par `.php` :

```
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
    }
```

Le `~` annonce une expression rationnelle. Le fichier inclus, livré avec Nginx, vérifie que le script existe avant de le confier à PHP. Un fichier `heure.php` de deux lignes suffit à l'essai :

```
gary@serveur:~$ cat /var/www/blog/heure.php
<?php
echo "Il est ", date("H:i:s"), " sur le serveur (PHP ", PHP_VERSION, ").\n";
gary@serveur:~$ curl -s -H 'Host: blog.maison.lan' http://localhost/heure.php
Il est 23:39:17 sur le serveur (PHP 8.3.6).
```

Notre carnet n'a pas besoin de PHP : nous retirons cette règle et ce fichier avant de continuer. Un interpréteur qui ne sert à rien est une porte de plus à surveiller.

23.6 Plusieurs sites sur un serveur

Le serveur héberge maintenant trois sites sur une seule adresse et un seul port.

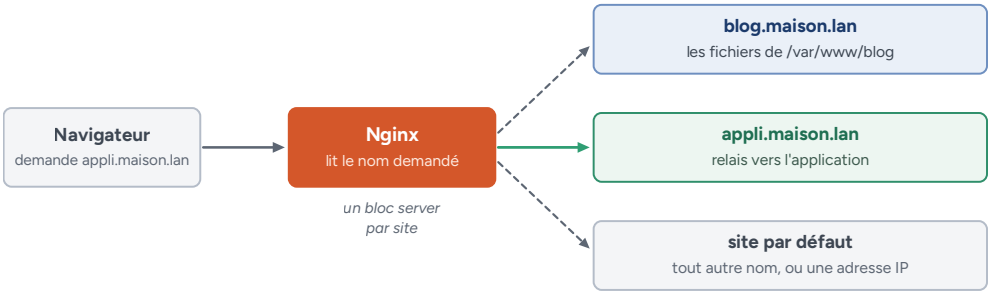


Figure 23.2 — Nginx choisit le site d'après le nom demandé.

À chaque demande, Nginx compare le nom reçu aux `server_name` de tous les blocs qui écoutent ce port. S'il en trouve un, il lui confie la demande. Sinon, il la donne au bloc marqué `default_server`. C'est lui qui a répondu tout à l'heure à `http://192.168.1.50`.

Un `server_name` peut porter plusieurs noms, séparés par des espaces : `server_name exemple.fr www.exemple.fr;`.

Pour ajouter un site, la marche ne change pas : un dossier dans `/var/www` ou une application sur un port local, un fichier dans `sites-available`, un lien dans `sites-enabled`, `nginx -t`, `reload`. Et un nom qui pointe vers la machine, dans le DNS.

23.7 HTTPS

En HTTP, tout circule en clair : les pages, les mots de passe, les cookies. HTTPS chiffre l'échange et prouve au visiteur qu'il parle au bon serveur. Cette preuve est un **certificat** : un fichier qui associe un nom de domaine à une clé publique, signé par une **autorité de certification** que les navigateurs connaissent.

> **REPÈRE HISTORIQUE · Let's Encrypt**
>
> Jusqu'en 2015, un certificat s'achetait, et s'installait à la main une fois par an. Let's Encrypt, porté par une association à but non lucratif, l'ISRG, a ouvert son service au public le 3 décembre 2015. Ses certificats sont gratuits, délivrés par un programme en quelques secondes, et valables quatre-vingt-dix jours seulement : la brièveté oblige à automatiser le renouvellement. Le chiffrement est devenu la norme sur le Web.

Obtenir un certificat avec Certbot

Let's Encrypt ne vous connaît pas. Il vérifie une seule chose : que vous tenez bien la machine vers laquelle pointe le nom.

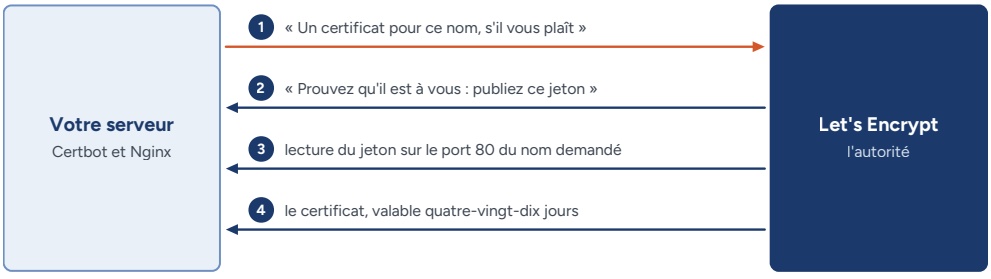


Figure 23.3 — La vérification d'un nom par Let's Encrypt.

Il faut donc un nom de domaine public, qui pointe vers le serveur, et un port 80 joignable depuis Internet. **Certbot** mène tout l'échange, et modifie la configuration de Nginx pour vous :

```
gary@serveur:~$ sudo apt install certbot python3-certbot-nginx
```

> **NOTE · D'où viennent les sorties de cette section**
>
> Un nom comme `blog.maison.lan` n'existe que sur notre réseau : Let's Encrypt ne peut pas le vérifier. Pour montrer la manœuvre en entier, le réseau d'essai de ce livre héberge Pebble, le serveur de test que Let's Encrypt publie pour cet usage. Sur votre serveur, mettez votre nom de domaine à la place du nôtre : les commandes sont les mêmes.

```
gary@serveur:~$ sudo certbot --nginx -d blog.maison.lan -m gary@maison.lan \
    --agree-tos --no-eff-email
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Account registered.
Requesting a certificate for blog.maison.lan

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/blog.maison.lan/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/blog.maison.lan/privkey.pem
This certificate expires on 2027-01-02.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate
  in the background.

Deploying certificate
Successfully deployed certificate for blog.maison.lan to /etc/nginx/sites-enabled/blog
Congratulations! You have successfully enabled HTTPS on https://blog.maison.lan
[…]
```

Option	Rôle
`--nginx`	Se servir de Nginx pour la vérification, puis y installer le certificat
`-d`	Le nom à certifier. Répétez l'option pour en couvrir plusieurs
`-m`	L'adresse électronique du compte
`--agree-tos`	Accepter les conditions d'utilisation de Let's Encrypt
`--no-eff-email`	Ne pas s'abonner à la lettre de l'EFF, qui développe Certbot

Sans les trois dernières, Certbot pose les questions une à une.

Ce que Certbot a changé

Le fichier du site n'est plus le même. Dans le bloc d'origine, les deux lignes `listen 80` ont laissé la place à ceci :

```
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/blog.maison.lan/fullchain.pem; # …
    ssl_certificate_key /etc/letsencrypt/live/blog.maison.lan/privkey.pem; # …
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
```

Le site écoute le port 443 en mode chiffré, avec le certificat et sa clé privée. Le fichier inclus contient des réglages de chiffrement tenus à jour par Certbot. Un second bloc, ajouté à la fin, reprend le port 80 et renvoie tous les visiteurs vers HTTPS :

```
server {
    if ($host = blog.maison.lan) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;
    server_name blog.maison.lan;
    return 404; # managed by Certbot
}
```

Depuis le poste :

```
gary@vostro:~$ curl -sI http://blog.maison.lan | head -n 1
HTTP/1.1 301 Moved Permanently
gary@vostro:~$ curl -sI https://blog.maison.lan | head -n 1
HTTP/1.1 200 OK
```

```
gary@vostro:~$ curl -sv -o /dev/null https://blog.maison.lan 2>&1 | \
    grep -E 'SSL conn|expire|verify ok'
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384 / X25519 / id-ecPublicKey
*  expire date: Jan  2 23:40:22 2027 GMT
*  SSL certificate verify ok.
```

La demande en clair est redirigée, la connexion chiffrée aboutit, et le certificat est reconnu.

Le renouvellement

```
gary@serveur:~$ sudo certbot certificates
[…]
  Certificate Name: blog.maison.lan
    Serial Number: a35d0d74bada94d
    Key Type: ECDSA
    Domains: blog.maison.lan
    Expiry Date: 2027-01-02 23:40:22+00:00 (VALID: 89 days)
    Certificate Path: /etc/letsencrypt/live/blog.maison.lan/fullchain.pem
    Private Key Path: /etc/letsencrypt/live/blog.maison.lan/privkey.pem
gary@serveur:~$ systemctl list-timers certbot.timer --no-pager
NEXT                         LEFT LAST PASSED UNIT          ACTIVATES
Mon 2026-10-05 06:04:34 UTC   6h -         - certbot.timer certbot.service
```

Le paquet a installé une minuterie systemd, du genre de celles du chapitre 13. Deux fois par jour, elle lance `certbot renew`, qui ne fait rien tant que le certificat a plus de trente jours devant lui :

```
gary@serveur:~$ sudo certbot renew
[…]
The following certificates are not due for renewal yet:
  /etc/letsencrypt/live/blog.maison.lan/fullchain.pem expires on 2027-01-02 (skipped)
No renewals were attempted.
```

Le moment venu, Certbot obtient un nouveau certificat et recharge Nginx. Pour vérifier que tout se passera bien ce jour-là, faites une répétition :

```
gary@serveur:~$ sudo certbot renew --dry-run
[…]
Congratulations, all simulated renewals succeeded:
  /etc/letsencrypt/live/blog.maison.lan/fullchain.pem (success)
```

> **ATTENTION · Personne ne vous préviendra**
>
> Depuis juin 2025, Let's Encrypt n'envoie plus de courriel quand un certificat approche de son terme. Si le renouvellement casse, à cause d'un port 80 fermé ou d'un nom qui ne pointe plus vers la machine, vous l'apprendrez par vos visiteurs. Faites la répétition après chaque changement de configuration, et surveillez la date d'expiration : le chapitre 26 montre comment.

Sans nom public : le certificat autosigné

Une application interne, joignable seulement depuis le réseau local, ne peut pas passer par Let's Encrypt. On peut tout de même chiffrer les échanges avec un certificat que l'on signe soi-même :

```
gary@serveur:~$ sudo openssl req -x509 -newkey rsa:2048 -nodes -days 365 \
    -keyout /etc/ssl/private/appli.key -out /etc/ssl/certs/appli.crt \
    -subj "/CN=appli.maison.lan" -addext "subjectAltName=DNS:appli.maison.lan"
[…]
gary@serveur:~$ sudo ls -l /etc/ssl/private/appli.key /etc/ssl/certs/appli.crt
-rw-r--r-- 1 root root 1172 Oct  4 23:41 /etc/ssl/certs/appli.crt
-rw------- 1 root root 1704 Oct  4 23:41 /etc/ssl/private/appli.key
gary@serveur:~$ openssl x509 -in /etc/ssl/certs/appli.crt -noout -subject -issuer -dates
subject=CN = appli.maison.lan
issuer=CN = appli.maison.lan
notBefore=Oct  4 23:41:25 2026 GMT
notAfter=Oct  4 23:41:25 2027 GMT
```

La commande fabrique une clé privée et un certificat valable un an. Le sujet et l'émetteur sont identiques : personne d'autre ne se porte garant. La clé n'est lisible que par `root`. Le certificat, lui, est public.

Le fichier du site passe à deux blocs, comme celui que Certbot a écrit :

```
server {
    listen 80;
    listen [::]:80;
    server_name appli.maison.lan;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name appli.maison.lan;

    ssl_certificate     /etc/ssl/certs/appli.crt;
    ssl_certificate_key /etc/ssl/private/appli.key;

    access_log /var/log/nginx/appli.access.log;
```

```
    error_log  /var/log/nginx/appli.error.log;

    location / {
        proxy_pass http://127.0.0.1:8000;
        include proxy_params;
    }
}
```

Après `nginx -t` et `reload`, essayons depuis le poste :

```
gary@vostro:~$ curl -I https://appli.maison.lan
curl: (60) SSL certificate problem: self-signed certificate
More details here: https://curl.se/docs/sslcerts.html
[…]
gary@vostro:~$ curl -skI https://appli.maison.lan | head -n 1
HTTP/1.1 200 OK
```

`curl` refuse : rien ne lui prouve que ce certificat est le bon. Un navigateur afficherait un avertissement en pleine page. L'option `-k` passe outre, ce qui revient à ne rien vérifier du tout. La bonne manière est de remettre le certificat au poste par un canal sûr, puis de le lui désigner :

```
gary@vostro:~$ scp serveur:/etc/ssl/certs/appli.crt .
Serveur de travail, usage restreint.
Les connexions sont inscrites au journal.
Enter passphrase for key '/home/gary/.ssh/id_ed25519':
appli.crt                                     100% 1172   975.2KB/s   00:00
gary@vostro:~$ curl -sI --cacert appli.crt https://appli.maison.lan | head -n 1
HTTP/1.1 200 OK
```

Cette fois le chiffrement est vérifié. Pour que tous les programmes du poste fassent confiance au certificat, copiez-le dans `/usr/local/share/ca-certificates/` et lancez `sudo update-ca-certificates`.

> **ATTENTION · Jamais sur un site public**
>
> Un certificat autosigné habitue les visiteurs à cliquer sur « continuer quand même », le geste qu'un attaquant attend d'eux. Réservez-le aux réseaux d'essai et aux outils internes dont vous maîtrisez tous les postes. Dès qu'un site a un nom public, Let's Encrypt est gratuit et ne demande pas plus de travail.

23.8 Vitesse et discrétion

Quatre réglages améliorent un site sans rien lui coûter.

HTTP/2

HTTP/2, normalisé en 2015, fait passer toutes les demandes d'une page par une seule connexion. Les navigateurs ne l'emploient qu'en HTTPS. Sur la version de Nginx livrée avec Ubuntu 24.04, il s'active en ajoutant `http2` aux lignes `listen` du port 443 :

```
gary@serveur:~$ grep -n 'listen.*443' /etc/nginx/sites-available/*
/etc/nginx/sites-available/appli:9:    listen 443 ssl http2;
/etc/nginx/sites-available/appli:10:    listen [::]:443 ssl http2;
/etc/nginx/sites-available/blog:18:    listen [::]:443 ssl http2 ipv6only=on; # …
/etc/nginx/sites-available/blog:19:    listen 443 ssl http2; # managed by Certbot
[…]
```

SUR LE TERRAIN · Sur tous les sites, ou sur aucun

Le mot `http2` s'applique au port, pas au site. Si vous ne l'ajoutez qu'à l'un des deux, `nginx -t` vous prévient, et l'active tout de même pour les deux :

```
[warn] 142554#142554: protocol options redefined for [::]:443 in
  /etc/nginx/sites-enabled/blog:14
```

Écrivez la même chose sur toutes les lignes `listen` d'un même port.

HTTP/3, normalisé en 2022, remplace TCP par un protocole bâti sur UDP. Nginx le propose depuis sa version 1.25. La version 1.24 d'Ubuntu 24.04 ne l'a pas.

Compression, cache et version

Le reste se règle dans un fichier commun à tous les sites, `/etc/nginx/conf.d/reglages.conf` :

```
# Ne pas annoncer la version de Nginx
server_tokens off;

# Compresser aussi les feuilles de style, les scripts et les données
gzip_types text/css application/javascript application/json image/svg+xml;
gzip_min_length 256;
```

`server_tokens off` retire le numéro de version des réponses : inutile de dire aux curieux quelle faille chercher. La ligne `gzip on` du fichier principal ne compresse que les pages HTML. `gzip_types` étend la compression aux autres fichiers de texte. Les images, déjà compressées, n'y gagneraient rien.

Dans le bloc `server` du carnet, une règle de plus demande aux navigateurs de garder les fichiers qui changent rarement :

```
    location ~* \.(css|js|png|jpg|svg|woff2)$ {
        expires 30d;
    }
```

Comparons la même demande, avant et après :

```
gary@vostro:~$ curl -sI -H 'Accept-Encoding: gzip' https://blog.maison.lan/style.css
HTTP/1.1 200 OK
Server: nginx/1.24.0 (Ubuntu)
Date: Sun, 04 Oct 2026 23:42:31 GMT
Content-Type: text/css
Content-Length: 333
[…]
```

```
gary@vostro:~$ curl -sI -H 'Accept-Encoding: gzip' https://blog.maison.lan/style.css
HTTP/2 200
server: nginx
date: Sun, 04 Oct 2026 23:42:37 GMT
content-type: text/css
last-modified: Sun, 04 Oct 2026 23:37:42 GMT
etag: W/"6ac2e346-14d"
expires: Tue, 03 Nov 2026 23:42:37 GMT
cache-control: max-age=2592000
content-encoding: gzip
```

La réponse arrive en HTTP/2, sans numéro de version, compressée, et le navigateur est autorisé à la garder trente jours, soit 2 592 000 secondes. La feuille de style pèse 225 octets sur le réseau au lieu de 333. Sur des fichiers plus gros, le gain est bien plus net.

> **NOTE · Les vieux protocoles de chiffrement**
>
> La ligne `ssl_protocols` du fichier principal cite encore TLS 1.0 et 1.1, abandonnés depuis des années. Elle n'a pas d'effet : la bibliothèque de chiffrement d'Ubuntu 24.04 les refuse, et une tentative de connexion en TLS 1.1 se solde par une alerte *protocol version*. Vous pouvez réduire la ligne à `TLSv1.2 TLSv1.3` pour que le fichier dise ce que le serveur fait.

23.9 Les journaux

Chaque site écrit deux journaux dans `/var/log/nginx/` : les accès et les erreurs. Une ligne d'accès se lit de gauche à droite :

```
gary@serveur:~$ tail -n 3 /var/log/nginx/blog.access.log
192.168.1.1 - - [04/Oct/2026:23:43:43 +0000] "GET /admin/ HTTP/2.0" 404 146
  "-" "curl/8.5.0"
192.168.1.1 - - [04/Oct/2026:23:43:43 +0000] "GET /style.css HTTP/2.0" 200 333
  "-" "curl/8.5.0"
192.168.1.1 - - [04/Oct/2026:23:43:43 +0000] "GET / HTTP/2.0" 200 272
  "-" "curl/8.5.0"
```

Champ	Exemple	Sens
1	`192.168.1.1`	L'adresse du visiteur
4 et 5	`[04/Oct/2026:23:43:43 +0000]`	La date et l'heure
6 à 8	`"GET /admin/ HTTP/2.0"`	La demande
9	`404`	Le code de la réponse
10	`146`	Sa taille en octets
11	`"-"`	La page d'où vient le visiteur
12 et suivants	`"curl/8.5.0"`	Le programme qui a fait la demande

Les outils du chapitre 9 en tirent l'essentiel en une ligne. La répartition des codes, puis les pages introuvables les plus demandées :

```
gary@serveur:~$ awk '{print $9}' /var/log/nginx/blog.access.log | sort | uniq -c | sort -rn
     30 200
      5 404
      1 403
gary@serveur:~$ awk '$9 == 404 {print $7}' /var/log/nginx/blog.access.log | \
    sort | uniq -c | sort -rn
      2 /absent.html
      1 /wp-login.php
      1 /admin/
      1 /.env
```

Ces lignes viennent de nos propres essais. Sur un serveur public, vous trouverez les mêmes adresses par centaines sans les avoir demandées : des robots parcourent Internet à la recherche de pages d'administration et de fichiers de mots de passe oubliés. Un 404 est la bonne réponse. Les outils du chapitre 21 s'occupent des plus insistants.

Le journal d'erreurs, lui, est celui qu'on ouvre quand quelque chose ne marche pas. Nous l'avons lu deux fois dans ce chapitre : pour un fichier illisible et pour une application arrêtée.

`logrotate`, vu au chapitre 13, archive ces journaux chaque jour et en garde quatorze. Les membres du groupe `adm` peuvent les lire sans `sudo`.

23.10 Apache, pour ceux qui en héritent

Vous croiserez Apache sur des serveurs installés avant vous. Les idées sont les mêmes, le vocabulaire diffère.

	Nginx	Apache
Paquet et service	`nginx`	`apache2`
Fichier principal	`/etc/nginx/nginx.conf`	`/etc/apache2/apache2.conf`
Ports écoutés	`listen`, dans chaque site	`/etc/apache2/ports.conf`
Un site	bloc `server`	bloc `<VirtualHost>`
Son nom	`server_name`	`ServerName`
Ses fichiers	`root`	`DocumentRoot`
Activer un site	lien dans `sites-enabled`	`a2ensite`, `a2dissite`
Activer un module	paquet à installer	`a2enmod`, `a2dismod`
Relais	`proxy_pass`	`ProxyPass`
Vérifier	`nginx -t`	`apache2ctl configtest`
Journaux	`/var/log/nginx/`	`/var/log/apache2/`
Certbot	`python3-certbot-nginx`	`python3-certbot-apache`

Deux serveurs ne peuvent pas écouter le même port. Installé à côté de Nginx, Apache ne démarre pas :

```
gary@serveur:~$ sudo apt install apache2
[…]
gary@serveur:~$ systemctl is-active apache2
failed
gary@serveur:~$ sudo journalctl -u apache2 -n 8 --no-pager -o cat
[…]
(98)Address already in use: AH00072: make_sock: could not bind to address [::]:80
(98)Address already in use: AH00072: make_sock: could not bind to address 0.0.0.0:80
no listening sockets available, shutting down
[…]
```

Address already in use : le message vaut pour n'importe quel service qui trouve son port occupé. Le temps de l'essai, plaçons Apache sur le port 8080, puis décrivons le carnet dans `/etc/apache2/sites-available/blog.conf` :

```
gary@serveur:~$ sudo sed -i 's/^Listen 80$/Listen 8080/' /etc/apache2/ports.conf
```

```
<VirtualHost *:8080>
    ServerName blog.maison.lan
    DocumentRoot /var/www/blog

    ErrorLog ${APACHE_LOG_DIR}/blog.error.log
    CustomLog ${APACHE_LOG_DIR}/blog.access.log combined
</VirtualHost>
```

```
gary@serveur:~$ sudo a2ensite blog
Enabling site blog.
To activate the new configuration, you need to run:
  systemctl reload apache2
gary@serveur:~$ sudo a2dissite 000-default
Site 000-default disabled.
[…]
gary@serveur:~$ sudo apache2ctl configtest
[…]
Syntax OK
gary@serveur:~$ sudo systemctl restart apache2
gary@serveur:~$ curl -sI -H 'Host: blog.maison.lan' http://localhost:8080 | head -n 3
HTTP/1.1 200 OK
Date: Sun, 04 Oct 2026 23:43:59 GMT
Server: Apache/2.4.58 (Ubuntu)
```

`a2ensite` crée le lien que nous posions à la main avec Nginx. Pour le relais, il faut d'abord activer deux modules :

```
<VirtualHost *:8080>
    ServerName appli.maison.lan

    ProxyPreserveHost On
    ProxyPass        / http://127.0.0.1:8000/
    ProxyPassReverse / http://127.0.0.1:8000/
</VirtualHost>
```

```
gary@serveur:~$ sudo a2enmod proxy proxy_http
[…]
gary@serveur:~$ sudo a2ensite appli
[…]
gary@serveur:~$ sudo systemctl restart apache2
```

```
gary@serveur:~$ curl -s -H 'Host: appli.maison.lan' http://localhost:8080/ | head -n 2
Charge du serveur, cinq derniers relevés
2026-10-04 23:43:20  0.03
```

Apache a une particularité : les fichiers `.htaccess`, qui permettent de régler le serveur depuis le dossier d'un site. Beaucoup de logiciels PHP en dépendent. Sur Ubuntu, ils sont ignorés sous `/var/www` tant que la directive `AllowOverride None` n'a pas été assouplie pour le dossier concerné.

L'essai terminé, on arrête Apache pour qu'il ne dispute pas le port 80 à Nginx au prochain démarrage :

```
gary@serveur:~$ sudo systemctl disable --now apache2
```

23.11 Caddy et Traefik

Caddy : HTTPS sans y penser

Caddy fait d'office ce que nous venons de construire à la main : il obtient les certificats, les renouvelle et redirige HTTP vers HTTPS. Il est dans les dépôts (`sudo apt install caddy`), en version 2.6.2 sur Ubuntu 24.04. Nos deux sites tiennent dans un seul fichier, `/etc/caddy/Caddyfile` :

```
blog.maison.lan {
    root * /var/www/blog
    file_server
}

appli.maison.lan {
    reverse_proxy 127.0.0.1:8000
}
```

Aucune ligne ne parle de certificat. Nginx arrêté pour libérer les ports, on démarre Caddy et on patiente quelques dizaines de secondes :

```
gary@vostro:~$ curl -sI http://blog.maison.lan | head -n 3
HTTP/1.1 308 Permanent Redirect
Connection: close
Location: https://blog.maison.lan/
gary@vostro:~$ curl -sI https://blog.maison.lan | head -n 3
HTTP/2 200
accept-ranges: bytes
alt-svc: h3=":443"; ma=2592000
```

```
gary@vostro:~$ curl -s https://appli.maison.lan | head -n 2
Charge du serveur, cinq derniers relevés
2026-10-04 23:46:10  0.01
```

Les deux certificats ont été demandés et installés sans intervention. Comme pour Certbot, notre réseau d'essai a dirigé Caddy vers le serveur de test, par deux lignes d'options qui n'ont pas leur place sur un serveur public. La ligne `alt-svc` annonce en prime HTTP/3. Sur un réseau sans nom public, ajoutez `tls internal` dans chaque bloc : Caddy signe alors lui-même ses certificats, avec les limites vues plus haut.

Caddy convient très bien à un petit serveur. Nginx garde l'avantage de l'ancienneté : presque toutes les documentations d'installation en donnent la configuration.

Traefik : le relais qui se configure seul

Traefik ne sert pas de fichiers. C'est un relais, et rien d'autre. Sa particularité est de découvrir lui-même les applications à relayer, en interrogeant Docker ou un autre orchestrateur : un conteneur démarre, Traefik lui ouvre une route et lui obtient un certificat. Nous le mettrons au travail de cette façon au chapitre 28.

Il n'est pas dans les dépôts d'Ubuntu. On télécharge le programme depuis la page du projet, en vérifiant son empreinte :

```
gary@serveur:~$ B=https://github.com/traefik/traefik/releases/download/v3.7.13
gary@serveur:~$ wget -q $B/traefik_v3.7.13_linux_amd64.tar.gz
gary@serveur:~$ wget -q $B/traefik_v3.7.13_checksums.txt
gary@serveur:~$ sha256sum -c --ignore-missing traefik_v3.7.13_checksums.txt
traefik_v3.7.13_linux_amd64.tar.gz: OK
gary@serveur:~$ tar -xzf traefik_v3.7.13_linux_amd64.tar.gz traefik
gary@serveur:~$ sudo install -m 755 traefik /usr/local/bin/
```

Sans Docker, on lui décrit les routes dans un fichier. Sa configuration se partage en deux. La partie fixe, `/etc/traefik/traefik.yml`, dit où écouter et où trouver les routes :

```
entryPoints:
  web:
    address: ":8081"

providers:
  file:
    filename: /etc/traefik/routes.yml
```

La partie mobile, `/etc/traefik/routes.yml`, peut changer sans redémarrage :

```
http:
  routers:
    appli:
      rule: "Host(`appli.maison.lan`)"
      service: tableau

  services:
    tableau:
      loadBalancer:
        servers:
          - url: "http://127.0.0.1:8000"
```

Mot	Sens	Équivalent Nginx
`entryPoints`	Les ports écoutés	`listen`
`routers`	Les règles qui trient les demandes	`server_name`, `location`
`services`	Les applications vers lesquelles relayer	`proxy_pass`
`providers`	Les sources où Traefik lit ses routes	les fichiers de `sites-enabled`

Lancé dans un second terminal avec `sudo traefik --configFile=/etc/traefik/traefik.yml`, il relaie notre application sur le port 8081 :

```
gary@serveur:~$ curl -s -H 'Host: appli.maison.lan' http://localhost:8081/ | head -n 2
Charge du serveur, cinq derniers relevés
2026-10-04 23:49:00  0.14
gary@serveur:~$ curl -sI -H 'Host: autre.maison.lan' http://localhost:8081/ | head -n 1
HTTP/1.1 404 Not Found
```

Un nom qu'aucun routeur ne connaît reçoit un 404. Traefik surveille son fichier de routes : remplacez le nom dans la règle, et le second essai répond 200 quelques secondes plus tard, sans qu'on ait rien relancé.

Lequel choisir ?

	Nginx	Apache	Caddy	Traefik
Sert des fichiers	oui	oui	oui	non
Relaie des applications	oui	oui	oui	oui
Certificats	avec Certbot	avec Certbot	automatiques	automatiques
Dans les dépôts d'Ubuntu	oui	oui	oui	non

	Nginx	Apache	Caddy	Traefik
Terrain de prédilection	serveur généraliste	sites PHP anciens	petit serveur	conteneurs

Pour un premier serveur, prenez Nginx : il fait tout, et vous le retrouverez partout. Les projets de la fin du livre s'appuient sur lui.

En résumé

- Un site, c'est un bloc `server` dans `sites-available`, un lien dans `sites-enabled`, puis `nginx -t` et `systemctl reload nginx`.
- Nginx choisit le site d'après le nom demandé. Sans correspondance, il sert le `default_server`.
- Les processus de travail tournent sous `www-data` : les fichiers d'un site doivent lui être lisibles.
- Devant une application, Nginx fait le relais avec `proxy_pass` et `include proxy_params`.
- Un 403 se règle dans les droits des fichiers, un 502 du côté de l'application. Le journal d'erreurs dit lequel.
- `certbot --nginx -d nom` obtient un certificat Let's Encrypt et configure HTTPS. Une minuterie le renouvelle. `certbot renew --dry-run` vérifie qu'elle y arrivera.
- Un certificat autosigné chiffre, mais ne prouve rien : pour l'interne seulement.
- `http2` sur les lignes `listen`, `gzip_types`, `expires`, `server_tokens off` : quatre réglages à poser une fois.

Atelier — Deux sites, un serveur

Objectif : héberger un site statique et une application sur la même machine, chacun sous son nom, en HTTPS. Comptez une heure et demie.

Le site statique

1. Créez `/var/www/docs` avec une page `index.html` de votre cru et une feuille de style.

2. Écrivez `/etc/nginx/sites-available/docs` pour le nom `docs.maison.lan`, avec ses propres journaux.
3. Activez le site. Vérifiez la configuration avant de recharger.
4. Testez depuis le serveur avec `curl -H 'Host: docs.maison.lan' http://localhost`.
5. Inscrivez le nom dans le fichier `/etc/hosts` du poste et testez depuis celui-ci.

L'application

6. Installez Gunicorn, copiez `tableau.py` dans `/opt/tableau` et créez le service `tableau`.
7. Vérifiez sur le serveur que l'application répond sur `127.0.0.1:8000`, et qu'elle ne répond pas depuis le poste sur `192.168.1.50:8000`.
8. Écrivez et activez le site `appli.maison.lan`, en relais.
9. Depuis le poste, vérifiez que la page affiche l'adresse du poste et non `127.0.0.1`.

Le chiffrement

10. Si votre serveur a un nom public, obtenez un certificat avec Certbot pour chaque site et faites une répétition du renouvellement. Sinon, fabriquez un certificat autosigné par site et écrivez les blocs du port 443 et de la redirection.
11. Activez HTTP/2 sur les deux sites. `nginx -t` ne doit afficher aucun avertissement.
12. Créez `/etc/nginx/conf.d/reglages.conf` et ajoutez la règle `expires` au site statique.

Casser, pour voir

13. Arrêtez le service `tableau` et demandez la page. Quel code obtenez-vous ? Que dit le journal d'erreurs du site ? Redémarrez le service.
14. Passez `index.html` en mode 600. Même question. Rétablissez le mode 644.

Vérifier

- `curl -sI http://docs.maison.lan` répond 301 et indique l'adresse en `https`.
- `curl -skI https://docs.maison.lan/style.css -H 'Accept-Encoding: gzip'` montre `HTTP/2`, `content-encoding: gzip` et `cache-control`, et ne montre plus la version de Nginx.
- `sudo ss -ltn` montre les ports 80 et 443 ouverts à tous, et le port 8000 sur `127.0.0.1` seulement.

Exercices

Exercice 1 — Lire une configuration

```
server {
    listen 80;
    server_name photos.exemple.fr www.photos.exemple.fr;
    root /srv/photos/public;
    index index.html;

    location /prive/ {
        return 403;
    }

    location ~* \.(jpg|png)$ {
        expires 90d;
    }
}
```

1. À quels noms ce site répond-il ?
2. Quel fichier est servi pour l'adresse `http://photos.exemple.fr/` ?
3. Que reçoit un visiteur qui demande `/prive/notes.txt` ?
4. Combien de temps un navigateur peut-il garder `/vacances/plage.jpg` ?
5. Ce site répond-il en IPv6 ?

Exercice 2 — Un troisième site

Créez un site `stats.maison.lan` qui sert le contenu de `/var/www/stats`. Déposez-y un fichier `index.html`. Vérifiez depuis le serveur, sans modifier aucun fichier `hosts`, que les trois sites répondent chacun avec sa propre page.

Exercice 3 — Trois pannes

Pour chacune, donnez la cause la plus probable et la première commande à lancer.

1. `curl -I` renvoie `502 Bad Gateway` sur un site en relais.
2. `curl -I` renvoie `403 Forbidden` sur une page qui existe.
3. `sudo systemctl reload nginx` répond `Job for nginx.service failed`.
4. Le navigateur affiche la page « Welcome to nginx! » au lieu de votre site.

Exercice 4 — Certificats

1. Listez les certificats gérés par Certbot sur votre serveur, ou, à défaut, affichez les dates de validité de votre certificat autosigné.
2. Quand la minuterie `certbot.timer` se déclenchera-t-elle la prochaine fois ?
3. Pourquoi les certificats de Let's Encrypt ne durent-ils que quatre-vingt-dix jours ?
4. Un collègue propose d'utiliser `curl -k` dans un script de surveillance « pour que ça marche ». Que lui répondez-vous ?

Exercice 5 — Faire parler le journal

À partir du journal d'accès de l'un de vos sites :

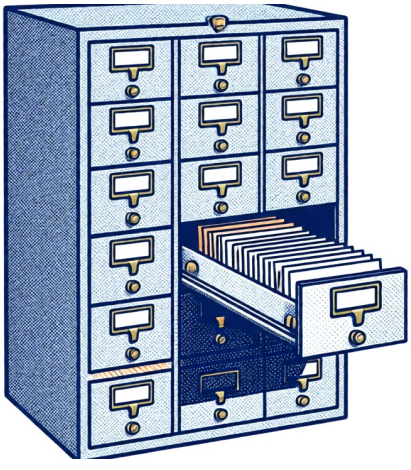
1. Combien de demandes a-t-il reçues aujourd'hui ?
2. Quelles sont les cinq adresses les plus actives ?
3. Quelles pages ont renvoyé un code 404 ?
4. Quelle part des demandes est arrivée en HTTP/2 ?

Pour aller plus loin

Sur la machine. `man nginx` est bref. La vraie documentation est en ligne, sur `nginx.org/en/docs/`, avec une page par directive. `certbot --help all` liste toutes les options de Certbot.

Paquets. `goaccess` transforme un journal d'accès en tableau de bord, dans le terminal ou en page HTML. `apache2-utils` fournit `ab`, pour mesurer ce qu'un site encaisse, et `htpasswd`, pour protéger un dossier par mot de passe.

En ligne. Le générateur de configuration de Mozilla, `ssl-config.mozilla.org`, produit des réglages de chiffrement à jour pour Nginx, Apache et Caddy. La documentation de Let's Encrypt, `letsencrypt.org/fr/docs/`, explique les limites du service et les autres modes de vérification, dont celui par le DNS, nécessaire pour les certificats couvrant tous les sous-domaines. Celles de Caddy (`caddyserver.com/docs`) et de Traefik (`doc.traefik.io`) sont claires et complètes.



CHAPITRE 24

Bases de données

Presque toutes les applications rangent leurs données dans une base. L'administrateur n'a pas à en écrire les requêtes, mais c'est à lui de l'installer, d'en fermer les portes, de donner à chacun les droits qu'il lui faut, et surtout de pouvoir la restaurer.

DANS CE CHAPITRE

- Choisir un type de base selon le besoin.
- Installer PostgreSQL et comprendre comment on s'y connecte.
- Créer une base, une table, des rôles aux droits limités.
- Ouvrir une base au réseau local sans l'exposer.
- Exporter, restaurer, et automatiser l'export chaque nuit.
- Retrouver les mêmes gestes dans MariaDB.
- Savoir à quoi servent Redis et SQLite.

AVANT DE COMMENCER

Les chapitres 6, 13, 16 et 21. Le service `veilleur` du chapitre 12 fournit les données d'exemple. Le serveur dispose du dossier `/srv/donnees` et de la sauvegarde restic du chapitre 16.

24.1 Quelle base pour quel besoin

Une **base de données** range des informations de façon à les retrouver vite, à plusieurs en même temps, sans les abîmer. Les plus répandues sont les bases **relationnelles** : les données y sont rangées dans des tables, faites de lignes et de colonnes, et on les interroge dans un langage commun, le **SQL**.

Famille	Exemples	Pour quoi faire
Relationnelle, avec serveur	PostgreSQL, MariaDB	Les données d'une application : clients, commandes, articles
Relationnelle, dans un fichier	SQLite	Une application à un seul utilisateur, un outil, un petit site
Clé-valeur en mémoire	Redis	Cache, sessions, compteurs, files d'attente

Si une application vous laisse le choix et que vous n'avez pas de raison d'en décider autrement, prenez PostgreSQL. Il est rigoureux, bien documenté, et ne vous enfermera pas. Vous croiserez MariaDB, ou son cousin MySQL, derrière beaucoup de logiciels PHP, à commencer par WordPress.

REPÈRE HISTORIQUE · Cinquante ans de tables

L'idée de ranger les données dans des tables liées entre elles date de 1970. PostgreSQL descend du projet POSTGRES, mené par Michael Stonebraker à l'université de Berkeley, et porte son nom actuel depuis 1996. MySQL est né en 1995. Quand Oracle s'est apprêté à en devenir propriétaire, son auteur, Michael Widenius, en a tiré en 2009 une version indépendante, MariaDB. Ubuntu fournit les deux, et leurs commandes sont presque identiques. SQLite, écrit en 2000 par Richard Hipp, tient tout entier dans une bibliothèque : on le trouve dans les téléphones, les navigateurs, et sur votre serveur.

24.2 PostgreSQL

Installer

```
gary@serveur:~$ sudo apt install postgresql
[…]
Creating new PostgreSQL cluster 16/main ...
The database cluster will be initialized with locale "C.UTF-8".
The default database encoding has accordingly been set to "UTF8".
[…]
```

Le paquet installe la version 16, crée une première **grappe** et la démarre. Une grappe, *cluster* en anglais, est une instance du serveur : un dossier de données, un port, un ensemble de bases.

```
gary@serveur:~$ pg_lsclusters
Ver Cluster Port Status Owner    Data directory              Log file
16  main    5432 online postgres /var/lib/postgresql/16/main /var/log/postgresql/…
gary@serveur:~$ sudo ss -ltnp | grep 5432 | cut -c1-70
LISTEN 0      200        127.0.0.1:5432      0.0.0.0:*    users:(("pos
LISTEN 0      200            [::1]:5432         [::]:*    users:(("pos
```

Le serveur écoute le port 5432, sur la machine elle-même seulement. Tel quel, il est injoignable depuis le réseau.

Emplacement	Contenu
`/etc/postgresql/16/main/postgresql.conf`	Les réglages du serveur
`/etc/postgresql/16/main/conf.d/`	Vos réglages, un fichier par sujet
`/etc/postgresql/16/main/pg_hba.conf`	Qui a le droit de se connecter, et comment
`/var/lib/postgresql/16/main/`	Les données. On n'y touche jamais à la main
`/var/log/postgresql/`	Le journal

NOTE · Deux services pour un serveur

`systemctl status postgresql` affiche `active (exited)`, ce qui surprend. Cette unité ne fait que lancer les grappes. Le vrai service de la nôtre s'appelle `postgresql@16-main`. Pour redémarrer ou recharger, `sudo systemctl restart postgresql` agit sur toutes les grappes.

Se connecter

Le client s'appelle `psql`. Premier essai :

```
gary@serveur:~$ psql
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432"
  failed: FATAL:  role "gary" does not exist
```

PostgreSQL a ses propres comptes, qu'il appelle des **rôles**. Ils n'ont rien à voir avec ceux de `/etc/passwd`. À l'installation, il n'en existe qu'un, `postgres`, qui a tous les pouvoirs. Aucun mot de passe ne lui est attribué : on l'emprunte en passant par le compte Unix du même nom.

```
gary@serveur:~$ sudo -u postgres psql
psql (16.15 (Ubuntu 16.15-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# \conninfo
You are connected to database "postgres" as user "postgres" via socket in
  "/var/run/postgresql" at port "5432".
postgres=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS

postgres=# \q
```

L'invite `postgres=#` donne le nom de la base. Le `#` final signale un superutilisateur, comme dans le shell. Les commandes qui commencent par une barre oblique inverse sont celles de `psql`. Le reste est du SQL, et se termine par un point-virgule.

La règle qui nous a laissés entrer s'appelle **peer** : par le socket local, le rôle doit porter le même nom que le compte Unix qui se connecte. C'est pourquoi `gary` a été refusé, et `postgres` accepté.

Une base et son propriétaire

Créons un rôle `gary` et une base `releves` qui lui appartient. Deux commandes du shell s'en chargent :

```
gary@serveur:~$ sudo -u postgres createuser gary
gary@serveur:~$ sudo -u postgres createdb -O gary releves
gary@serveur:~$ psql releves
psql (16.15 (Ubuntu 16.15-0ubuntu0.24.04.1))
Type "help" for help.
```

```
releves=> \conninfo
You are connected to database "releves" as user "gary" via socket in
  "/var/run/postgresql" at port "5432".
```

Plus besoin de `sudo` ni de mot de passe : le compte Unix `gary` correspond au rôle `gary`. L'invite se termine par `>`, signe d'un rôle ordinaire.

Une table et des requêtes

Le service `veilleur` écrit ses relevés dans un fichier, une ligne toutes les dix secondes : la date, un point-virgule, la charge. Donnons-leur une table.

```
releves=> CREATE TABLE releves (
releves(>     id       bigserial PRIMARY KEY,
releves(>     instant  timestamp NOT NULL,
releves(>     charge   numeric(5,2) NOT NULL
releves(> );
CREATE TABLE
```

Chaque colonne a un nom et un type. `bigserial` numérote les lignes tout seul. `NOT NULL` interdit les cases vides. `PRIMARY KEY` désigne la colonne qui identifie chaque ligne. La base refusera tout ce qui ne respecte pas ces règles, ce qui est le premier service qu'elle rend.

`\copy` charge un fichier dans une table :

```
releves=> \copy releves (instant, charge) FROM '/var/lib/veilleur/charge.csv' DELIMITER ';'
COPY 290
releves=> SELECT count(*) FROM releves;
 count
-------
   290
(1 row)

releves=> SELECT * FROM releves ORDER BY instant DESC LIMIT 3;
 id  |       instant       | charge
-----+---------------------+--------
 290 | 2026-10-05 00:03:10 |   0.00
 289 | 2026-10-05 00:03:00 |   0.00
 288 | 2026-10-05 00:02:50 |   0.00
(3 rows)
```

Une requête dit ce qu'on veut, pas comment l'obtenir. Celle-ci calcule la charge moyenne et le maximum, minute par minute :

```
releves=> SELECT date_trunc('minute', instant) AS minute,
releves->        round(avg(charge), 2) AS moyenne, max(charge) AS maximum
releves-> FROM releves GROUP BY 1 ORDER BY 1 DESC LIMIT 4;
       minute        | moyenne | maximum
```

```
---------------------+---------+---------
 2026-10-05 00:03:00 |    0.00 |    0.00
 2026-10-05 00:02:00 |    0.00 |    0.00
 2026-10-05 00:01:00 |    0.01 |    0.02
 2026-10-05 00:00:00 |    0.03 |    0.05
(4 rows)
```

Avec les outils du chapitre 9, le même calcul aurait demandé un script. Voici les commandes de `psql` qui servent tous les jours :

Commande	Effet
`\l`	Lister les bases
`\c base`	Changer de base
`\dt`	Lister les tables
`\d table`	Décrire une table : colonnes, types, index
`\du`	Lister les rôles
`\dp table`	Montrer les droits sur une table
`\x`	Afficher chaque ligne à la verticale, pour les tables larges
`\password role`	Changer un mot de passe
`\?` et `\h`	L'aide de `psql`, et celle du SQL
`\q`	Quitter

```
releves=> \dt
        List of relations
 Schema |  Name   | Type  | Owner
--------+---------+-------+-------
 public | releves | table | gary
(1 row)
```

Un rôle aux droits limités

L'application qui affichera ces relevés n'a besoin que de les lire. Elle aura son propre rôle, `tableau`, et rien de plus que ce droit. Créer un rôle demande les pouvoirs de `postgres` :

```
gary@serveur:~$ sudo -u postgres psql releves
[…]
releves=# CREATE ROLE tableau LOGIN;
CREATE ROLE
```

```
releves=# \password tableau
Enter new password for user "tableau":
Enter it again:
releves=# GRANT SELECT ON releves TO tableau;
GRANT
releves=# \dp releves
                            Access privileges
 Schema |  Name   | Type  | Access privileges | Column privileges | Policies
--------+---------+-------+-------------------+-------------------+----------
 public | releves | table | gary=arwdDxt/gary+|                   |
        |         |       | tableau=r/gary    |                   |
(1 row)
```

`\password` demande le mot de passe sans l'afficher. Écrit en clair dans un `CREATE ROLE … PASSWORD`, il resterait dans l'historique de `psql` et peut-être dans le journal du serveur. La dernière colonne résume les droits : `gary` les a tous, `tableau` n'a que `r`, la lecture.

Essayons ce rôle :

```
gary@serveur:~$ psql -U tableau releves
psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432"
  failed: FATAL:  Peer authentication failed for user "tableau"
```

Ce message déroute tous les débutants. Le mot de passe n'y est pour rien : il n'a même pas été demandé. Par le socket local, la règle *peer* exige que le compte Unix s'appelle `tableau`. Pour s'identifier par mot de passe, il faut passer par le réseau, fût-ce à l'adresse de la machine elle-même, avec `-h` :

```
gary@serveur:~$ psql -h 127.0.0.1 -U tableau releves
Password for user tableau:
psql (16.15 (Ubuntu 16.15-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

releves=> SELECT count(*) FROM releves;
 count
-------
   290
(1 row)

releves=> INSERT INTO releves (instant, charge) VALUES (now(), 9.99);
ERROR:  permission denied for table releves
releves=> DELETE FROM releves;
ERROR:  permission denied for table releves
```

Le rôle lit, et ne peut rien écrire ni effacer. Si l'application est un jour détournée, les relevés ne risquent rien.

pg_hba.conf, le portier

Ces règles d'entrée sont écrites dans `pg_hba.conf`. Le serveur lit le fichier de haut en bas et applique la première ligne qui correspond à la connexion :

```
gary@serveur:~$ sudo grep -vE '^\s*#|^\s*$' /etc/postgresql/16/main/pg_hba.conf
local   all             postgres                                peer
local   all             all                                     peer
host    all             all             127.0.0.1/32            scram-sha-256
host    all             all             ::1/128                 scram-sha-256
[…]
```

Colonne	Sens
Type	`local` pour le socket Unix, `host` pour le réseau, `hostssl` pour le réseau chiffré seulement
Base	La base visée, ou `all`
Rôle	Le rôle qui se présente, ou `all`
Adresse	D'où vient la connexion. Absente pour `local`
Méthode	`peer`, `scram-sha-256` pour un mot de passe, `reject` pour refuser

On retrouve nos deux essais : `peer` par le socket, mot de passe par `127.0.0.1`. Vous verrez parfois la méthode `trust` dans des tutoriels. Elle laisse entrer sans rien demander : ne l'employez pas.

24.3 Ouvrir la base au réseau

Supposons que l'application tourne sur une autre machine du réseau local. Depuis le poste, avec un délai d'attente de cinq secondes :

```
gary@vostro:~$ PGCONNECT_TIMEOUT=5 psql -h 192.168.1.50 -U tableau releves
psql: error: connection to server at "192.168.1.50", port 5432 failed: timeout expired
```

Il y a trois portes à ouvrir, et chacune a son message quand elle est fermée.

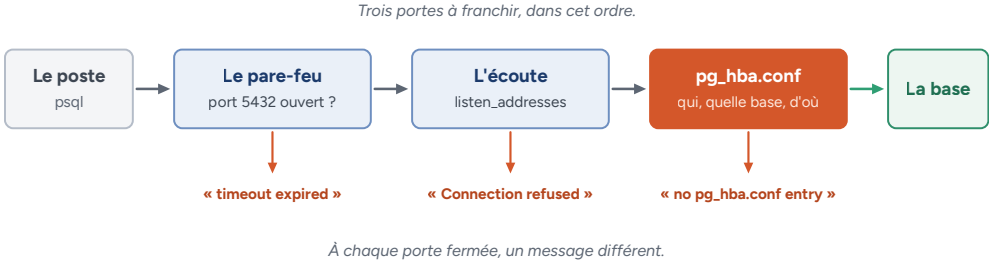


Figure 24.1 — Les trois portes d'une connexion distante.

Le pare-feu. Aucune réponse : le pare-feu du chapitre 21 jette les paquets. On ouvre le port, pour le réseau local seulement :

```
gary@serveur:~$ sudo ufw allow from 192.168.1.0/24 to any port 5432 proto tcp
Rule added
```

```
gary@vostro:~$ psql -h 192.168.1.50 -U tableau releves
psql: error: connection to server at "192.168.1.50", port 5432 failed: Connection refused
        Is the server running on that host and accepting TCP/IP connections?
```

L'écoute. La réponse est immédiate, et c'est un refus : le paquet arrive, mais personne n'écoute à cette adresse. PostgreSQL ne répond que sur `localhost`. On lui ajoute l'adresse du serveur, dans un fichier à nous :

```
gary@serveur:~$ cat /etc/postgresql/16/main/conf.d/ecoute.conf
# Écouter aussi sur l'adresse du réseau local
listen_addresses = 'localhost,192.168.1.50'
gary@serveur:~$ sudo systemctl restart postgresql
gary@serveur:~$ sudo ss -ltnp | grep 5432 | cut -c1-70
LISTEN 0      200      192.168.1.50:5432      0.0.0.0:*    users:(("pos
LISTEN 0      200         127.0.0.1:5432      0.0.0.0:*    users:(("pos
LISTEN 0      200             [::1]:5432         [::]:*    users:(("pos
```

```
gary@vostro:~$ psql -h 192.168.1.50 -U tableau releves
psql: error: connection to server at "192.168.1.50", port 5432 failed: FATAL:  no
  pg_hba.conf entry for host "192.168.1.1", user "tableau", database "releves",
  SSL encryption
[…]
```

Le portier. Cette fois, c'est PostgreSQL qui répond, et il est précis : aucune ligne de `pg_hba.conf` ne prévoit ce rôle, sur cette base, depuis cette adresse. Ajoutons-la, aussi étroite que possible :

```
gary@serveur:~$ echo 'hostssl releves tableau 192.168.1.0/24 scram-sha-256' | \
    sudo tee -a /etc/postgresql/16/main/pg_hba.conf
hostssl releves tableau 192.168.1.0/24 scram-sha-256
gary@serveur:~$ sudo systemctl reload postgresql
```

Un seul rôle, une seule base, un seul réseau, en connexion chiffrée. Un changement d'adresse d'écoute demande un redémarrage. Pour `pg_hba.conf`, un rechargement suffit.

```
gary@vostro:~$ psql -h 192.168.1.50 -U tableau releves
Password for user tableau:
psql (16.15 (Ubuntu 16.15-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

releves=> SELECT max(charge) FROM releves;
 max
------
 1.34
(1 row)
```

La connexion est chiffrée d'office : le paquet a configuré le serveur avec un certificat autosigné. Il protège des écoutes, pas d'un faux serveur, pour les raisons vues au chapitre 23.

Un script ne peut pas taper de mot de passe. `psql` le lit dans le fichier `~/.pgpass`, à condition que vous soyez seul à pouvoir le lire :

```
gary@vostro:~$ echo '192.168.1.50:5432:releves:tableau:Savane-Orange-58' > ~/.pgpass
gary@vostro:~$ chmod 600 ~/.pgpass
gary@vostro:~$ psql -h 192.168.1.50 -U tableau releves -c 'SELECT count(*) FROM releves'
 count
-------
   290
(1 row)
```

DANGER · Jamais sur Internet

Une base ouverte à tout Internet est sondée dans l'heure, et les mots de passe faibles tombent vite. N'écrivez jamais `listen_addresses = '*'` accompagné d'une ligne `host all all 0.0.0.0/0`. Si une application distante doit atteindre la base, faites passer la connexion par un tunnel SSH, comme au chapitre 19, ou par un réseau privé.

24.4 Sauvegarder et restaurer

Le chapitre 16 l'a dit : on ne copie pas les fichiers d'une base en marche. On lui demande un **export**, cohérent même si des écritures ont lieu pendant ce temps, et c'est lui que l'on sauvegarde.

Exporter

```
gary@serveur:~$ pg_dump releves > releves.sql
gary@serveur:~$ grep -nE '^(CREATE TABLE|COPY|ALTER TABLE|GRANT)' releves.sql
29:CREATE TABLE public.releves (
36:ALTER TABLE public.releves OWNER TO gary;
63:ALTER TABLE ONLY public.releves ALTER COLUMN id SET DEFAULT nextval('public.…
70:COPY public.releves (id, instant, charge) FROM stdin;
375:ALTER TABLE ONLY public.releves
383:GRANT SELECT ON TABLE public.releves TO tableau;
```

L'export est un fichier de texte : les ordres SQL qui recréent la table, ses données, ses droits. On peut le lire, le comparer, le rejouer avec `psql base < fichier.sql`.

`pg_dump` connaît un second format, compressé, que l'on préférera pour les sauvegardes :

```
gary@serveur:~$ pg_dump -Fc releves -f releves.dump
gary@serveur:~$ ls -lh releves.sql releves.dump
-rw-rw-r-- 1 gary gary 4.4K Oct  5 00:05 releves.dump
-rw-rw-r-- 1 gary gary  11K Oct  5 00:05 releves.sql
```

Il se restaure avec `pg_restore`, qui sait ne reprendre qu'une table ou travailler sur plusieurs processeurs.

Un export de base ne contient pas les rôles, qui appartiennent à la grappe entière. On les exporte à part :

```
gary@serveur:~$ sudo -u postgres pg_dumpall --globals-only | grep ROLE
CREATE ROLE gary;
ALTER ROLE gary WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN […];
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN […];
CREATE ROLE tableau;
ALTER ROLE tableau WITH NOSUPERUSER […] PASSWORD 'SCRAM-SHA-256$4096:jna7DJT8H…';
```

Les mots de passe y figurent sous forme d'empreintes. Ce fichier se protège comme `/etc/shadow`.

Restaurer, pour de bon

Un export qu'on n'a jamais restauré ne vaut pas grand-chose. Restaurons le nôtre dans une base d'essai :

```
gary@serveur:~$ sudo -u postgres createdb -O gary essai
gary@serveur:~$ pg_restore -d essai releves.dump
gary@serveur:~$ psql essai -c 'SELECT count(*) FROM releves'
 count
-------
   290
(1 row)

gary@serveur:~$ sudo -u postgres dropdb essai
```

Les 290 lignes sont là, et les droits de `tableau` aussi. Dans l'ordre, une restauration complète rejoue donc l'export des rôles, crée la base, puis y restaure l'export.

Chaque nuit



On ne copie jamais les fichiers d'une base en marche : on sauvegarde son export.

Figure 24.2 — De la base au dépôt de sauvegarde.

Un script exporte toutes les bases dans `/srv/donnees/bases`, un dossier que restic sauvegarde déjà :

```
#!/bin/bash
#
# sauvegarde-bases.sh — exporte chaque base PostgreSQL, puis les rôles
# Lancé chaque nuit par sauvegarde-bases.service, sous le compte postgres.

set -euo pipefail

dossier=/srv/donnees/bases
jour=$(date +%F)

bases=$(psql -XAtc "SELECT datname FROM pg_database
                    WHERE NOT datistemplate AND datname <> 'postgres'")

for base in $bases; do
    pg_dump -Fc "$base" -f "$dossier/$base-$jour.dump"
```

```
    echo "base $base exportée"
done

pg_dumpall --globals-only > "$dossier/roles-$jour.sql"

# sept jours sur place ; restic garde l'historique (chapitre 16)
find "$dossier" -type f -mtime +7 -delete
```

L'option `-X` de `psql` lui fait ignorer le fichier de préférences `~/.psqlrc`. Dans un script, mettez-la toujours : un réglage personnel suffit à polluer la sortie, et la liste des bases avec.

Le service et sa minuterie suivent le modèle du chapitre 13 :

```
[Unit]
Description=Export des bases PostgreSQL
After=postgresql.service

[Service]
Type=oneshot
User=postgres
ExecStart=/usr/local/sbin/sauvegarde-bases.sh
```

```
[Unit]
Description=Export des bases PostgreSQL, chaque nuit

[Timer]
OnCalendar=*-*-* 01:00:00
Persistent=true

[Install]
WantedBy=timers.target
```

Le script tourne sous le compte `postgres`, qui entre sans mot de passe. Il passe à une heure du matin, une demi-heure avant restic.

```
gary@serveur:~$ sudo install -d -o postgres -g postgres -m 750 /srv/donnees/bases
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl enable --now sauvegarde-bases.timer
Created symlink /etc/systemd/system/timers.target.wants/sauvegarde-bases.timer → …
gary@serveur:~$ sudo systemctl start sauvegarde-bases.service
gary@serveur:~$ sudo journalctl -u sauvegarde-bases -n 4 --no-pager -o cat
Starting sauvegarde-bases.service - Export des bases PostgreSQL...
base releves exportée
sauvegarde-bases.service: Deactivated successfully.
Finished sauvegarde-bases.service - Export des bases PostgreSQL.
gary@serveur:~$ sudo ls -l /srv/donnees/bases
```

```
total 12
-rw-r--r-- 1 postgres postgres 4449 Oct  5 00:06 releves-2026-10-05.dump
-rw-r--r-- 1 postgres postgres  909 Oct  5 00:06 roles-2026-10-05.sql
```

Une base créée demain sera exportée sans qu'on touche au script.

> **NOTE · Revenir à la minute près**
>
> Un export nocturne fait perdre, au pire, une journée de données. Pour mieux, PostgreSQL sait archiver en continu son journal de transactions, ce qui permet de rejouer l'histoire jusqu'à un instant choisi. Les outils `pgbackrest` et `barman`, présents dans les dépôts, organisent ce travail. C'est un chantier à part, à ouvrir quand une journée perdue coûte trop cher.

24.5 Régler la mémoire

PostgreSQL est livré avec des réglages prudents, faits pour démarrer partout.

```
gary@serveur:~$ psql releves -c 'SHOW shared_buffers' -c 'SHOW work_mem' \
    -c 'SHOW max_connections'
 shared_buffers
----------------
 128MB
(1 row)

 work_mem
----------
 4MB
(1 row)

 max_connections
-----------------
 100
(1 row)
```

Réglage	Rôle	Point de départ
`shared_buffers`	La mémoire où PostgreSQL garde les données lues	Un quart de la mémoire, sur un serveur réservé à la base
`work_mem`	La mémoire d'un tri ou d'un regroupement	Quelques mégaoctets. Chaque connexion peut en prendre plusieurs fois autant
`max_connections`	Le nombre de connexions simultanées	Cent suffisent presque toujours

Le quart de la mémoire est la valeur que conseille la documentation de PostgreSQL pour un serveur qui ne fait que cela. Le nôtre a 1,5 Go et héberge aussi un serveur web : restons en dessous.

```
gary@serveur:~$ cat /etc/postgresql/16/main/conf.d/memoire.conf
# Serveur de 1,5 Go de mémoire, partagé avec d'autres services
shared_buffers = 256MB
work_mem = 8MB
gary@serveur:~$ sudo systemctl restart postgresql
gary@serveur:~$ psql releves -c 'SHOW shared_buffers' -c 'SHOW work_mem'
 shared_buffers
----------------
 256MB
(1 row)

 work_mem
----------
 8MB
(1 row)
```

Ne réglez pas à l'aveugle. Une base de quelques mégaoctets, comme la nôtre, tient tout entière dans les réglages d'origine. Ces valeurs comptent quand la base dépasse la mémoire de la machine, et les outils du chapitre 26 vous diront si c'est le cas.

24.6 MariaDB

Les idées sont les mêmes, et les noms changent.

	PostgreSQL	MariaDB
Paquet	`postgresql`	`mariadb-server`
Client	`psql`	`mariadb`
Port	5432	3306
Administrer sur place	`sudo -u postgres psql`	`sudo mariadb`
Configuration	`/etc/postgresql/16/main/`	`/etc/mysql/mariadb.conf.d/`
Adresse d'écoute	`listen_addresses`	`bind-address`
Qui peut entrer	`pg_hba.conf`	L'hôte fait partie du nom du compte
Lister les bases	`\l`	`SHOW DATABASES;`
Lister les tables	`\dt`	`SHOW TABLES;`

	PostgreSQL	MariaDB
Exporter	`pg_dump`	`mariadb-dump`
Mémoire principale	`shared_buffers`	`innodb_buffer_pool_size`

Les anciens noms de commandes, `mysql` et `mysqldump`, fonctionnent toujours.

```
gary@serveur:~$ sudo apt install mariadb-server
[…]
gary@serveur:~$ sudo ss -ltnp | grep 3306 | cut -c1-70
LISTEN 0      80         127.0.0.1:3306      0.0.0.0:*    users:(("mar
gary@serveur:~$ mariadb
ERROR 1698 (28000): Access denied for user 'gary'@'localhost'
```

Comme PostgreSQL, MariaDB n'écoute que la machine elle-même, et son compte d'administration, `root`, n'a pas de mot de passe : on y entre par `sudo`.

```
gary@serveur:~$ sudo mariadb
[…]
MariaDB [(none)]> CREATE DATABASE releves;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> CREATE USER 'tableau'@'localhost' IDENTIFIED BY 'Savane-Orange-58';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT SELECT ON releves.* TO 'tableau'@'localhost';
Query OK, 0 rows affected (0.031 sec)
```

La différence la plus visible est là : un compte MariaDB porte le nom de la machine d'où il vient. `'tableau'@'localhost'` et `'tableau'@'192.168.1.%'` sont deux comptes distincts, avec chacun son mot de passe et ses droits. C'est ce qui tient lieu de `pg_hba.conf`.

```
MariaDB [(none)]> USE releves
Database changed
MariaDB [releves]> CREATE TABLE releves (
    ->     id      BIGINT AUTO_INCREMENT PRIMARY KEY,
    ->     instant DATETIME NOT NULL,
    ->     charge  DECIMAL(5,2) NOT NULL
    -> );
Query OK, 0 rows affected (0.039 sec)

MariaDB [releves]> INSERT INTO releves (instant, charge)
    -> VALUES (NOW(), 0.42), (NOW(), 0.57);
Query OK, 2 rows affected (0.013 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Les types ne portent pas tout à fait les mêmes noms : chaque base a son dialecte. Le rôle limité se comporte comme attendu :

```
gary@serveur:~$ mariadb -u tableau -p releves
Enter password:
[…]
MariaDB [releves]> SELECT MAX(charge) FROM releves;
+-------------+
| MAX(charge) |
+-------------+
|        0.57 |
+-------------+
1 row in set (0.000 sec)

MariaDB [releves]> DELETE FROM releves;
ERROR 1142 (42000): DELETE command denied to user 'tableau'@'localhost' for table
  `releves`.`releves`
```

L'export et la restauration :

```
gary@serveur:~$ sudo mariadb-dump releves > releves-mariadb.sql
gary@serveur:~$ sudo mariadb -e 'CREATE DATABASE essai'
gary@serveur:~$ sudo mariadb essai < releves-mariadb.sql
gary@serveur:~$ sudo mariadb -e 'SELECT COUNT(*) FROM essai.releves; DROP DATABASE essai'
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

`sudo mariadb-dump --all-databases` exporte tout, comptes compris. Pour l'automatiser, reprenez le service et la minuterie de PostgreSQL en changeant la commande.

Le réglage de mémoire qui compte s'appelle `innodb_buffer_pool_size`. Il vaut 128 Mo à l'installation, et se change dans `/etc/mysql/mariadb.conf.d/50-server.cnf`.

> **SUR LE TERRAIN · mariadb-secure-installation**
>
> Tous les tutoriels font lancer ce script après l'installation. Il propose de donner un mot de passe à `root`, de supprimer les comptes anonymes, d'interdire à `root` de se connecter à distance et d'effacer une base `test`. Sur Ubuntu 24.04, le paquet est déjà livré dans cet état : `root` n'entre que par `sudo`, et il n'y a ni compte anonyme ni base `test`. Le script le dit lui-même, *you already have your root account protected*. Vous pouvez le lancer sans risque et répondre non aux deux premières questions. Il reste utile sur d'autres systèmes.

24.7 Redis

Redis garde ses données en mémoire, sous la forme la plus simple qui soit : une clé, une valeur. Il répond très vite, et sert à tout ce qui doit aller vite sans être précieux.

```
gary@serveur:~$ sudo apt install redis-server
[…]
gary@serveur:~$ redis-cli ping
PONG
gary@serveur:~$ redis-cli set visites 0
OK
gary@serveur:~$ redis-cli incr visites
(integer) 1
gary@serveur:~$ redis-cli incr visites
(integer) 2
```

Un compteur, tenu à jour sans qu'on ait rien déclaré. Une valeur peut aussi s'effacer toute seule :

```
gary@serveur:~$ redis-cli set session:42 gary EX 30
OK
gary@serveur:~$ redis-cli ttl session:42
(integer) 30
gary@serveur:~$ redis-cli ttl session:42
(integer) 26
```

`EX 30` donne trente secondes à vivre à la clé, et `ttl` montre ce qu'il lui reste. C'est le mécanisme des sessions d'un site web et des caches : on y range le résultat d'un calcul coûteux, pour un temps.

Usage	Ce qu'on y met
Cache	Le résultat d'une requête lente, pour quelques minutes
Sessions	L'état des visiteurs connectés
Compteurs et limites	Le nombre de tentatives par adresse
Files d'attente	Les tâches à traiter en arrière-plan

```
gary@serveur:~$ sudo grep -nE '^(bind|protected-mode|port|maxmemory|save) ' \
    /etc/redis/redis.conf
87:bind 127.0.0.1 -::1
111:protected-mode yes
138:port 6379
```

Redis n'écoute que la machine elle-même, et ne demande aucun mot de passe. Laissez-le ainsi : l'application qui s'en sert tourne presque toujours sur le même serveur. Ne l'exposez jamais tel quel sur un réseau.

> **NOTE · Redis et Valkey**
>
> En mars 2024, l'éditeur de Redis a remplacé sa licence libre par des licences plus restrictives. La Linux Foundation a aussitôt repris la dernière version libre sous le nom de Valkey. Ubuntu 24.04 fournit les deux : `redis-server` en version 7.0, antérieure au changement, et `valkey-server`. Les commandes sont les mêmes. Si un logiciel vous demande l'un, l'autre fera l'affaire.

24.8 SQLite

SQLite n'a ni serveur, ni port, ni compte. La base est un fichier, et le programme qui s'en sert la lit directement.

```
gary@serveur:~$ sudo apt install sqlite3
[…]
gary@serveur:~$ sqlite3 carnet.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> CREATE TABLE notes (id INTEGER PRIMARY KEY, texte TEXT NOT NULL,
   ...>     creee TEXT DEFAULT CURRENT_TIMESTAMP);
sqlite> INSERT INTO notes (texte) VALUES ('Renouveler le certificat du blog'),
   ...>     ('Tester la restauration');
sqlite> SELECT * FROM notes;
1|Renouveler le certificat du blog|2026-10-05 00:08:50
2|Tester la restauration|2026-10-05 00:08:50
sqlite> .backup carnet-copie.db
sqlite> .quit
gary@serveur:~$ ls -l carnet.db carnet-copie.db
-rw-r--r-- 1 gary gary 8192 Oct  5 00:08 carnet-copie.db
-rw-r--r-- 1 gary gary 8192 Oct  5 00:08 carnet.db
```

Les droits sont ceux du fichier. La sauvegarde se fait avec `.backup`, qui produit une copie cohérente même si un programme écrit au même moment. Un simple `cp` pendant une écriture peut donner un fichier abîmé.

Votre serveur s'en sert déjà sans vous l'avoir dit :

```
gary@serveur:~$ sudo find /var/lib -name '*.sqlite3' -o \
    -name '*.db' 2>/dev/null | head -n 5
/var/lib/PackageKit/transactions.db
```

```
/var/lib/fail2ban/fail2ban.sqlite3
/var/lib/command-not-found/commands.db
/var/lib/fwupd/pending.db
```

Fail2ban y tient la liste des adresses bannies, et le shell y cherche le paquet à proposer quand une commande manque. SQLite est le bon choix tant qu'un seul programme écrit à la fois : un outil, un site à faible trafic, un prototype. Quand plusieurs machines ou beaucoup d'écritures simultanées entrent en jeu, on passe à PostgreSQL.

Les trois serveurs installés dans ce chapitre occupent ensemble, au repos, 200 Mo de mémoire environ. Sur un vrai serveur, n'installez pourtant que celui dont vous avez l'usage.

En résumé

- PostgreSQL par défaut, MariaDB quand un logiciel la réclame, SQLite pour un seul programme, Redis pour ce qui doit aller vite et peut se perdre.
- Sur Ubuntu, on administre PostgreSQL par `sudo -u postgres psql`, et MariaDB par `sudo mariadb`. Aucun mot de passe d'administration n'est à créer.
- Un rôle par application, avec les seuls droits utiles : `GRANT SELECT`, pas davantage, pour qui ne fait que lire.
- `pg_hba.conf` décide qui entre, sur quelle base, d'où et comment. *Peer authentication failed* signifie qu'il faut passer par `-h` pour s'identifier par mot de passe.
- Pour ouvrir une base au réseau : le pare-feu, `listen_addresses`, puis une ligne étroite dans `pg_hba.conf`. Jamais à tout Internet.
- On sauvegarde l'export, `pg_dump -Fc` ou `mariadb-dump`, et l'on exporte aussi les rôles. On vérifie en restaurant.
- Dans un script, `psql -X`.

Atelier — Une base pour l'inventaire

Objectif : créer une base de deux tables, lui donner un rôle de lecture, l'exporter chaque nuit et prouver qu'on sait la restaurer. Comptez une heure.

Préparer

1. Installez PostgreSQL si ce n'est pas fait. Vérifiez avec `ss` qu'il n'écoute que la machine elle-même.
2. Créez un rôle à votre nom et une base `inventaire` qui vous appartient.

Construire

3. Enregistrez ce schéma dans `inventaire.sql` et chargez-le avec `psql inventaire -f inventaire.sql` :

```
-- Inventaire du parc : les machines et les interventions faites dessus

CREATE TABLE machines (
    id       serial PRIMARY KEY,
    nom      text NOT NULL UNIQUE,
    adresse  inet,
    role     text
);

CREATE TABLE interventions (
    id       serial PRIMARY KEY,
    machine  integer NOT NULL REFERENCES machines (id),
    jour     date NOT NULL DEFAULT current_date,
    objet    text NOT NULL
);

INSERT INTO machines (nom, adresse, role) VALUES
    ('serveur', '192.168.1.50', 'web et bases'),
    ('vostro',  '192.168.1.1',  'poste de travail');

INSERT INTO interventions (machine, jour, objet) VALUES
    (1, '2026-09-28', 'Durcissement SSH'),
    (1, '2026-10-02', 'Mise en place du pare-feu'),
    (2, '2026-10-03', 'Remplacement du disque');
```

4. Listez les tables, décrivez chacune.
5. Affichez les interventions avec le nom de la machine concernée :

```
SELECT m.nom, i.jour, i.objet
FROM interventions i JOIN machines m ON m.id = i.machine
ORDER BY i.jour;
```

6. Essayez d'enregistrer une intervention sur la machine numéro 9, qui n'existe pas. Que répond la base ? Quelle ligne du schéma l'explique ?

Limiter

7. Créez un rôle `lecteur` avec un mot de passe, et donnez-lui la lecture sur toutes les tables : `GRANT SELECT ON ALL TABLES IN SCHEMA public TO lecteur;`.
8. Connectez-vous sous ce rôle. Vérifiez qu'il lit, et qu'il ne peut ni ajouter ni supprimer.

Sauvegarder

9. Installez le script, le service et la minuterie du chapitre. Lancez le service à la main et vérifiez que `inventaire` a été exportée sans que vous ayez modifié le script.
10. Restaurez l'export dans une base `inventaire_essai`. Comparez le nombre de lignes des deux tables avec l'original, puis supprimez la base d'essai.

Casser, pour voir

11. Renommez provisoirement `/srv/donnees/bases` et lancez le service. Que dit `systemctl status` ? Que dit le journal ? Remettez le dossier en place. Le chapitre 26 montrera comment être prévenu d'un tel échec.

Vérifier

- `sudo -u postgres psql -c '\du'` montre votre rôle et `lecteur`, sans attribut particulier.
- `systemctl list-timers sauvegarde-bases.timer` annonce le prochain passage.
- Le dossier des exports contient un fichier par base et un fichier de rôles, datés du jour.

Exercices

Exercice 1 — Se repérer

1. Quelle version de PostgreSQL tourne sur votre machine ? Sur quel port, à quelles adresses ?
2. Où se trouvent ses données ? Qui peut lire ce dossier ?
3. Combien de bases contient la grappe ? À qui appartiennent-elles ?
4. Quelle différence y a-t-il entre les invites `base=#` et `base=⇒` ?

Exercice 2 — Lire pg_hba.conf

```
local   all       postgres                    peer
local   all       all                         peer
host    all       all       127.0.0.1/32      scram-sha-256
hostssl compta    appli     10.0.0.0/24       scram-sha-256
host    all       all       0.0.0.0/0         reject
```

1. `sudo -u postgres psql` aboutit-il ? Par quelle ligne ?
2. Le compte Unix `marie` lance `psql -U appli compta`. Que se passe-t-il ?
3. Le rôle `appli` se connecte à la base `compta` depuis `10.0.0.7`. Quelle ligne s'applique ? Et depuis `10.0.1.7` ?
4. Le rôle `appli` se connecte depuis `10.0.0.7` à la base `paie`. Résultat ?

Exercice 3 — Trois messages

Pour chacun, dites quelle porte est fermée et comment l'ouvrir.

1. `connection to server at "192.168.1.50", port 5432 failed: Connection refused`
2. `FATAL: Peer authentication failed for user "appli"`
3. `FATAL: no pg_hba.conf entry for host "192.168.1.23", user "appli", database "compta"`
4. `ERROR: permission denied for table factures`

Exercice 4 — Sauvegarde

1. Exportez la base `releves` au format texte et au format compressé. Comparez les tailles.
2. Dans l'export texte, retrouvez l'ordre qui donne ses droits au rôle `tableau`.
3. Pourquoi faut-il aussi lancer `pg_dumpall --globals-only` ?
4. Un collègue sauvegarde sa base en copiant `/var/lib/postgresql` chaque nuit avec `rsync`. Que lui dites-vous ?

Exercice 5 — MariaDB, Redis, SQLite

1. Dans MariaDB, créez une base `essai` et un compte qui ne peut que la lire. Vérifiez.
2. Dans Redis, créez une clé qui disparaît au bout de dix secondes. Observez son compte à rebours.

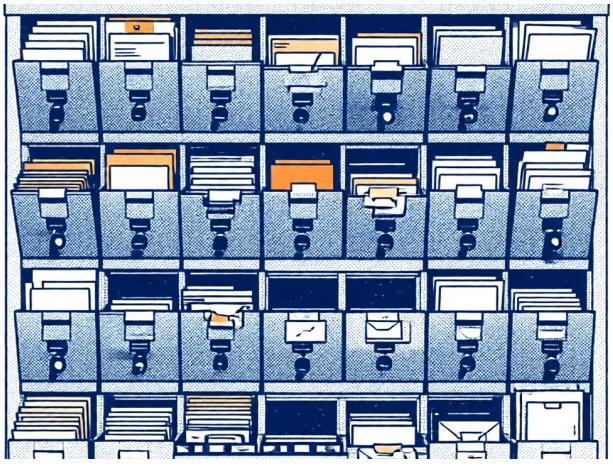
3. Avec `sqlite3`, ouvrez `/var/lib/command-not-found/commands.db` et listez ses tables.
4. Pour chacun de ces besoins, quelle base choisiriez-vous : le catalogue d'une boutique en ligne, l'historique d'un script personnel, le panier des visiteurs d'un site ?

Pour aller plus loin

Sur la machine. `man psql`, `man pg_dump`, `man pg_restore`. Dans `psql`, `\h CREATE TABLE` affiche la syntaxe d'un ordre SQL. Le fichier `postgresql.conf` est abondamment commenté et vaut une lecture.

Paquets. `pgbackrest` et `barman` pour la sauvegarde continue de PostgreSQL. `pgcli` et `mycli`, des clients avec complétion et couleurs. `pgtop` pour voir les requêtes en cours.

En ligne. La documentation de PostgreSQL, `postgresql.org/docs/16/`, est un modèle du genre, et sa première partie est un cours de SQL. Le site `pgtune.leopard.in.ua` propose des réglages de départ selon la mémoire et l'usage. La base de connaissances de MariaDB se trouve sur `mariadb.com/kb/`. Celle de SQLite, `sqlite.org/whentouse.html`, explique sans détour quand le choisir et quand y renoncer.



CHAPITRE 25

Partager des fichiers sur le réseau

Un dossier commun à plusieurs machines : la demande est simple, et il existe quatre façons d'y répondre. NFS entre machines Linux, Samba dès qu'un poste Windows ou un Mac est de la partie, SFTP pour un dépôt à distance, WebDAV quand seul le Web passe. Chacune a sa façon de dire qui a le droit de quoi.

DANS CE CHAPITRE

- Exporter un dossier en NFS et le monter sur un autre poste.
- Comprendre comment NFS identifie les utilisateurs, et ce que cela implique.
- Monter un partage au démarrage ou à la demande.
- Installer Samba, créer un partage commun et des dossiers personnels.
- Se connecter à un partage Samba depuis Linux, Windows et macOS.
- Offrir un dépôt SFTP à un compte enfermé dans son dossier.
- Choisir le protocole selon la situation.

> **AVANT DE COMMENCER**
>
> Les chapitres 6, 14, 19, 21 et 23. Le serveur partage, le poste `vostro` se connecte. Le pare-feu du chapitre 21 est actif : chaque service devra y être autorisé.

25.1 Quatre façons de partager

Protocole	Né pour	Identifie par	Chiffré
NFS	les machines Unix d'un même réseau	l'adresse de la machine et le numéro de l'utilisateur	non, sauf réglage avancé
SMB, avec Samba	les postes Windows	un nom et un mot de passe	pas d'office : un réglage du serveur l'impose
SFTP	le transfert à distance	les comptes et les clés de SSH	toujours
WebDAV	le Web	un nom et un mot de passe, sur HTTPS	avec HTTPS

Les deux premiers se montent comme un disque : le dossier distant apparaît dans l'arborescence et tous les programmes s'en servent sans le savoir. Les deux autres servent plutôt à déposer et à retirer des fichiers.

25.2 NFS

NFS, *Network File System*, vient de Sun Microsystems, qui l'a publié en 1984. C'est le partage naturel entre machines Linux : rapide, sans mot de passe, intégré au noyau.

Le serveur

```
gary@serveur:~$ sudo apt install nfs-kernel-server
[…]
gary@serveur:~$ systemctl is-active nfs-server
active
```

Créons le dossier à partager. Il appartiendra à un groupe, `partage`, dont le numéro est fixé à la main. Nous verrons pourquoi.

```
gary@serveur:~$ sudo groupadd -g 3000 partage
gary@serveur:~$ sudo usermod -aG partage gary
gary@serveur:~$ sudo install -d -o root -g partage -m 2775 /srv/partage
gary@serveur:~$ ls -ld /srv/partage
drwxrwsr-x 2 root partage 4096 Oct  5 00:18 /srv/partage
```

Le mode 2775 pose le bit *setgid* du chapitre 6 : tout ce qui sera créé dans le dossier appartiendra au groupe `partage`.

Les dossiers offerts au réseau sont listés dans `/etc/exports`, un par ligne : le dossier, puis qui peut le monter et avec quelles options.

```
gary@serveur:~$ echo '/srv/partage 192.168.1.0/24(rw,sync,no_subtree_check)' | \
    sudo tee -a /etc/exports
/srv/partage 192.168.1.0/24(rw,sync,no_subtree_check)
gary@serveur:~$ sudo exportfs -ra
gary@serveur:~$ sudo exportfs -v
/srv/partage
192.168.1.0/24(sync,wdelay,hide,no_subtree_check,sec=sys,rw,secure,
                root_squash,no_all_squash)
```

Option	Sens
`rw` ou `ro`	Lecture et écriture, ou lecture seule
`sync`	N'accuser réception d'une écriture qu'une fois les données sur le disque
`no_subtree_check`	Réglage conseillé, qui évite des contrôles coûteux
`root_squash`	Le `root` d'un poste client n'est personne sur le serveur. Actif d'office

ATTENTION · L'espace qui change tout

Dans `/etc/exports`, il n'y a pas d'espace entre l'adresse et la parenthèse. Écrire `192.168.1.0/24 (rw)` exporte le dossier en lecture seule à ce réseau, et en écriture au monde entier.

`exportfs -ra` relit le fichier. `exportfs -v` montre le résultat, avec les options que vous n'avez pas écrites. Reste le pare-feu : NFS version 4 n'a besoin que du port 2049.

```
gary@serveur:~$ sudo ufw allow from 192.168.1.0/24 to any port 2049 proto tcp
Rule added
```

Le client

```
gary@vostro:~$ sudo apt install nfs-common
[…]
gary@vostro:~$ sudo mkdir /mnt/partage
gary@vostro:~$ sudo mount -t nfs 192.168.1.50:/srv/partage /mnt/partage
gary@vostro:~$ findmnt -t nfs4 -o TARGET,SOURCE,FSTYPE
TARGET       SOURCE                    FSTYPE
/mnt/partage 192.168.1.50:/srv/partage nfs4
gary@vostro:~$ df -h /mnt/partage
Filesystem                Size  Used Avail Use% Mounted on
192.168.1.50:/srv/partage   16G  4.9G   10G  33% /mnt/partage
```

Le dossier du serveur est monté, comme une partition du chapitre 14. La source s'écrit `machine:/dossier`.

```
gary@vostro:~$ ls -ld /mnt/partage
drwxrwsr-x 2 root 3000 4096 Oct  5 00:18 /mnt/partage
```

Le groupe s'affiche sous la forme d'un numéro. Le poste ne connaît pas de groupe 3000. Créons-le, avec le même numéro que sur le serveur :

```
gary@vostro:~$ sudo groupadd -g 3000 partage
gary@vostro:~$ ls -ld /mnt/partage
drwxrwsr-x 2 root partage 4096 Oct  5 00:18 /mnt/partage
gary@vostro:~$ echo "Bonjour du poste" > /mnt/partage/bonjour.txt
gary@vostro:~$ sudo touch /mnt/partage/essai-root.txt
touch: cannot touch '/mnt/partage/essai-root.txt': Permission denied
```

`gary` écrit dans le partage. `root`, lui, est refusé : c'est l'effet de `root_squash`. Être administrateur de son poste ne donne aucun pouvoir sur les fichiers du serveur.

NFS ne connaît que des numéros

Voici le point qu'il faut avoir compris avant de déployer NFS. Sur le serveur, créons une utilisatrice, `marie`, et un fichier à elle. Sur le poste, créons un utilisateur `paul`, qui reçoit par hasard le même numéro :

```
gary@serveur:~$ sudo useradd -u 1005 -m marie
gary@serveur:~$ sudo touch /srv/partage/note-de-marie.txt
gary@serveur:~$ sudo chown marie: /srv/partage/note-de-marie.txt
```

```
gary@vostro:~$ sudo useradd -u 1005 -m paul
gary@vostro:~$ ls -l /mnt/partage/note-de-marie.txt
-rw-r--r-- 1 paul paul 0 Oct  5 00:18 /mnt/partage/note-de-marie.txt
gary@vostro:~$ sudo -u paul sh -c 'echo "Signé : paul" >> /mnt/partage/note-de-marie.txt'
```

```
gary@serveur:~$ ls -l /srv/partage
total 8
-rw-r--r-- 1 gary  partage 17 Oct  5 00:18 bonjour.txt
-rw-r--r-- 1 marie marie   14 Oct  5 00:18 note-de-marie.txt
gary@serveur:~$ cat /srv/partage/note-de-marie.txt
Signé : paul
```

Sur le poste, le fichier de `marie` appartient à `paul`, et `paul` l'a modifié.



NFS ne transmet pas de nom, et ne demande aucun mot de passe.

Figure 25.1 — NFS compare des numéros, pas des noms.

Le serveur fait confiance au poste : celui-ci annonce « je suis le numéro 1005 », et le serveur le croit. Deux conséquences.

Les numéros doivent correspondre d'une machine à l'autre. Sur un petit réseau, on crée les comptes dans le même ordre, ou l'on fixe les numéros avec `useradd -u` et `groupadd -g`. Les grandes installations confient les comptes à un annuaire central.

Quiconque est `root` sur un poste autorisé peut y créer un compte du numéro de son choix, et lire les fichiers de n'importe qui. On n'exporte donc en NFS que vers des machines que l'on administre soi-même. La ligne de `/etc/exports` désigne ces machines, aussi précisément que possible.

Monter au démarrage, ou à la demande

Une ligne dans `/etc/fstab` rend le montage permanent :

```
192.168.1.50:/srv/partage /mnt/partage nfs defaults,_netdev,nofail 0 0
```

```
gary@vostro:~$ sudo mount /mnt/partage
gary@vostro:~$ findmnt /mnt/partage -o TARGET,SOURCE,FSTYPE
TARGET       SOURCE                    FSTYPE
/mnt/partage 192.168.1.50:/srv/partage nfs4
```

`_netdev` prévient le système qu'il faut attendre le réseau. `nofail`, vu au chapitre 14, lui permet de démarrer quand même si le serveur est éteint.

Sur un portable, qui n'est pas toujours sur le réseau du serveur, mieux vaut un montage **à la demande**. Dans la ligne, on remplace `defaults,_netdev,nofail` par :

```
noauto,x-systemd.automount,x-systemd.idle-timeout=10min,_netdev
```

Après `sudo systemctl daemon-reload` et un redémarrage, ou `sudo systemctl start mnt-partage.automount`, systemd surveille le dossier. Le partage n'est monté qu'au premier accès, et démonté après dix minutes sans activité. Un `findmnt /mnt/partage` montre alors une ligne `autofs`, à laquelle s'ajoute la ligne `nfs4` tant que le partage sert.

25.3 Samba

Windows partage ses fichiers avec un autre protocole, SMB. **Samba** le parle depuis 1992 : il fait d'un serveur Linux un serveur de fichiers pour Windows, et macOS comme Linux savent s'y connecter. Quand les postes sont de plusieurs sortes, c'est lui qu'il faut.

Contrairement à NFS, Samba demande un nom et un mot de passe à chaque utilisateur.

Installer et configurer

```
gary@serveur:~$ sudo apt install samba
[…]
gary@serveur:~$ sudo ss -ltnp | grep smbd | cut -c1-70
LISTEN 0      50           0.0.0.0:139        0.0.0.0:*    users:(("sm
LISTEN 0      50           0.0.0.0:445        0.0.0.0:*    users:(("sm
[…]
```

Tout se règle dans `/etc/samba/smb.conf`. Le fichier livré est long et parle surtout d'imprimantes. Mettons-le de côté et repartons d'un fichier court :

```
gary@serveur:~$ sudo cp /etc/samba/smb.conf /etc/samba/smb.conf.origine
```

```
[global]
    workgroup = WORKGROUP
    server string = Serveur de fichiers
    server role = standalone server
    map to guest = never
    load printers = no
    log file = /var/log/samba/log.%m
    max log size = 1000

[partage]
    comment = Documents communs
    path = /srv/partage
    valid users = @partage
    read only = no
    create mask = 0664
    directory mask = 2775

[homes]
    comment = Dossier personnel
    browseable = no
    read only = no
    valid users = %S
```

La section `[global]` règle le serveur. Chaque autre section décrit un partage, et son nom entre crochets est celui que verront les postes.

Réglage	Sens
`path`	Le dossier partagé
`valid users`	Qui peut entrer. `@partage` désigne les membres du groupe
`read only = no`	Autoriser l'écriture
`create mask`, `directory mask`	Les droits des fichiers et des dossiers créés
`browseable = no`	Ne pas montrer le partage dans la liste
`map to guest = never`	Refuser les connexions sans compte

`[homes]` est un partage particulier : il offre à chaque utilisateur son répertoire personnel, sous son propre nom. `%S` y vaut le nom du partage demandé, donc celui de l'utilisateur.

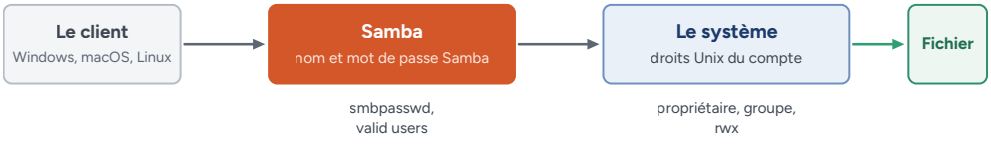
`testparm` vérifie le fichier, comme `nginx -t` au chapitre 23. Il n'affiche que ce qui s'écarte des valeurs d'origine :

```
gary@serveur:~$ testparm -s
Load smb config files from /etc/samba/smb.conf
Loaded services file OK.
[…]
Server role: ROLE_STANDALONE
[…]
[partage]
        comment = Documents communs
        create mask = 0664
        directory mask = 02775
        path = /srv/partage
        read only = No
        valid users = @partage
[…]
```

Les comptes

Samba tient sa propre liste de mots de passe, distincte de `/etc/shadow`. Un utilisateur doit donc exister deux fois : comme compte Unix, puis comme compte Samba.

```
gary@serveur:~$ sudo smbpasswd -a gary
New SMB password:
Retype new SMB password:
Added user gary.
gary@serveur:~$ sudo usermod -aG partage marie
gary@serveur:~$ sudo smbpasswd -a marie
New SMB password:
Retype new SMB password:
Added user marie.
gary@serveur:~$ sudo pdbedit -L
gary:1000:Gary
marie:1005:
gary@serveur:~$ sudo systemctl restart smbd
```



Il faut passer les deux contrôles : le second a toujours le dernier mot.

Figure 25.2 — Les deux contrôles d'un partage Samba.

Une fois le mot de passe accepté, Samba agit sur les fichiers au nom du compte Unix, avec ses droits. Si le partage autorise l'écriture et que le dossier l'interdit à ce

compte, l'écriture est refusée. Quand un utilisateur « ne peut pas écrire », regardez d'abord `ls -ld` sur le dossier.

Le paquet a déposé un profil pour le pare-feu, qui réunit les quatre ports du protocole :

```
gary@serveur:~$ sudo ufw allow from 192.168.1.0/24 to any app Samba
Rule added
```

Se connecter depuis Linux

`smbclient` interroge un serveur sans rien monter. C'est l'outil de vérification :

```
gary@vostro:~$ sudo apt install smbclient cifs-utils
[…]
gary@vostro:~$ smbclient -L //192.168.1.50 -U gary
Password for [WORKGROUP\gary]:

        Sharename       Type      Comment
        ---------       ----      -------
        partage         Disk      Documents communs
        IPC$            IPC       IPC Service (Serveur de fichiers)
        gary            Disk      Dossier personnel
[…]
gary@vostro:~$ smbclient //192.168.1.50/partage -U gary
Password for [WORKGROUP\gary]:
Try "help" to get a list of possible commands.
smb: \> put /etc/os-release os-release.txt
putting file /etc/os-release as \os-release.txt (97.7 kb/s) (average 97.7 kb/s)
smb: \> ls
  .                                   D        0  Mon Oct  5 00:21:21 2026
  ..                                  D        0  Mon Oct  5 00:21:21 2026
  bonjour.txt                         N       17  Mon Oct  5 00:18:19 2026
  note-de-marie.txt                   N       14  Mon Oct  5 00:18:21 2026
  os-release.txt                      A      400  Mon Oct  5 00:21:21 2026

                16403324 blocks of size 1024. 10350024 blocks available
smb: \> quit
```

Le serveur annonce le partage commun et le dossier personnel de celui qui se présente. Vérifions les refus :

```
gary@vostro:~$ smbclient //192.168.1.50/partage -U gary%faux -c ls
session setup failed: NT_STATUS_LOGON_FAILURE
gary@vostro:~$ smbclient //192.168.1.50/gary -U marie%Papaye-Bleue-34 -c ls
tree connect failed: NT_STATUS_ACCESS_DENIED
```

Mauvais mot de passe : la session est refusée. Bon mot de passe, mais dossier d'un autre : c'est le partage qui est refusé. Ces deux messages disent à quel contrôle on s'est arrêté.

Pour monter le partage, on range le nom et le mot de passe dans un fichier que seul `root` peut lire :

```
gary@vostro:~$ printf 'username=gary\npassword=Mangue-Verte-21\n' | \
    sudo tee /etc/smb-partage.cred > /dev/null
gary@vostro:~$ sudo chmod 600 /etc/smb-partage.cred
gary@vostro:~$ sudo mkdir /mnt/smb
gary@vostro:~$ sudo mount -t cifs //192.168.1.50/partage /mnt/smb \
    -o credentials=/etc/smb-partage.cred,uid=gary,gid=gary
gary@vostro:~$ ls -l /mnt/smb
total 12
-rwxr-xr-x 1 gary gary  17 Oct  5 00:18 bonjour.txt
-rwxr-xr-x 1 gary gary  14 Oct  5 00:18 note-de-marie.txt
-rwxr-xr-x 1 gary gary 400 Oct  5 00:21 os-release.txt
```

Les options `uid` et `gid` disent à qui attribuer les fichiers sur le poste. Avec SMB, les droits réels se décident sur le serveur, d'après le compte qui s'est connecté. Ce que le poste affiche n'est qu'un habillage. Côté serveur, les fichiers créés portent bien le groupe et les droits voulus :

```
gary@serveur:~$ ls -l /srv/partage
total 16
-rw-r--r-- 1 gary  partage  17 Oct  5 00:18 bonjour.txt
-rw-rw-r-- 1 gary  partage  15 Oct  5 00:21 montage.txt
-rw-r--r-- 1 marie marie    14 Oct  5 00:18 note-de-marie.txt
-rw-rw-r-- 1 gary  partage 400 Oct  5 00:21 os-release.txt
gary@serveur:~$ sudo smbstatus --shares

Service      pid     Machine       Connected at                     Encryption   Signing
---------------------------------------------------------------------------------------------
IPC$         198114  192.168.1.1   Mon Oct  5 00:21:28 2026 UTC     -            -
partage      198114  192.168.1.1   Mon Oct  5 00:21:28 2026 UTC     -            -
```

`smbstatus` montre qui est connecté, et à quoi. Les colonnes de droite indiquent que l'échange n'est ni chiffré ni signé : sur un réseau qui n'est pas sûr, ajoutez `server smb encrypt = required` à la section `[global]`.

Pour un montage permanent, la ligne de `/etc/fstab` suit le modèle de NFS : la source `//192.168.1.50/partage`, le type `cifs`, et les options `credentials=/etc/smb-partage.cred,uid=gary,gid=gary,_netdev,nofail`.

Dans le gestionnaire de fichiers d'Ubuntu, le même partage s'ouvre sans rien monter à la main : *Autres emplacements*, puis l'adresse `smb://192.168.1.50/partage` dans le champ du bas.

Depuis Windows et macOS

Sous Windows, tapez `\\192.168.1.50\partage` dans la barre d'adresse de l'Explorateur, puis le nom et le mot de passe Samba. Un clic droit sur le partage propose de le connecter comme lecteur réseau, avec une lettre.

Sous macOS, dans le Finder, le menu *Aller* propose *Se connecter au serveur*. L'adresse s'écrit `smb://192.168.1.50/partage`.

> **SUR LE TERRAIN · Le partage ouvert à tous, une fausse bonne idée**
>
> Beaucoup de tutoriels créent un partage « invité », sans mot de passe. Les versions récentes de Windows refusent de s'y connecter, pour de bonnes raisons de sécurité, et l'on perd une soirée à comprendre pourquoi. Créez un compte par personne. C'est aussi ce qui permet de savoir qui a effacé quoi.

Au-delà du petit réseau

Dans une entreprise équipée d'un domaine Active Directory, on ne crée pas les comptes un par un. Le serveur Samba rejoint le domaine et reconnaît les utilisateurs et les groupes de l'annuaire. Samba sait même tenir le rôle de contrôleur de domaine. Ce sont de vrais chantiers, qui sortent du cadre de ce livre. La documentation d'Ubuntu Server leur consacre plusieurs pages.

25.4 SFTP, pour déposer à distance

Vous avez déjà un partage de fichiers chiffré, qui traverse Internet sans crainte : SFTP, livré avec SSH au chapitre 19. Pour récupérer ou déposer des fichiers sur un serveur distant, il n'y a rien à installer.

Le cas qui demande un réglage est celui du prestataire, à qui l'on veut offrir un dépôt sans lui ouvrir un shell ni lui montrer le reste du système. On l'enferme dans un dossier.

```
gary@serveur:~$ sudo groupadd depot
gary@serveur:~$ sudo useradd -m -g depot -G acces-ssh -s /usr/sbin/nologin livraison
gary@serveur:~$ sudo install -d -o root -g root -m 755 /srv/depot /srv/depot/livraison
gary@serveur:~$ sudo install -d -o livraison -g depot -m 750 /srv/depot/livraison/arrivages
```

Le compte `livraison` n'a pas de shell. Il appartient au groupe `acces-ssh`, seul admis par notre serveur depuis le chapitre 19. Sa clé publique va dans son fichier `authorized_keys`, comme pour tout autre compte. Le dossier où il sera enfermé

doit appartenir à `root`, et lui seul peut y écrire : SSH l'exige. Le compte dépose donc ses fichiers un cran plus bas, dans `arrivages`.

La règle tient dans `/etc/ssh/sshd_config.d/50-depot.conf` :

```
Match Group depot
    ChrootDirectory /srv/depot/%u
    ForceCommand internal-sftp
    AllowTcpForwarding no
    X11Forwarding no
```

`ChrootDirectory` fait du dossier la racine du compte : `%u` y vaut son nom. `ForceCommand internal-sftp` remplace toute commande par le service de transfert.

```
gary@serveur:~$ sudo sshd -t
gary@serveur:~$ sudo systemctl reload ssh
```

```
gary@vostro:~$ sftp livraison@192.168.1.50
[…]
Connected to 192.168.1.50.
sftp> pwd
Remote working directory: /
sftp> cd arrivages
sftp> put /etc/os-release
Uploading /etc/os-release to /arrivages/os-release
os-release                                    100%  400   390.7KB/s   00:00
sftp> cd /etc
stat remote: No such file or directory
sftp> mkdir /essai
remote mkdir "/essai": Permission denied
sftp> bye
gary@vostro:~$ ssh livraison@192.168.1.50
[…]
This service allows sftp connections only.
Connection to 192.168.1.50 closed.
```

Pour ce compte, le système s'arrête à son dossier : `/etc` n'existe pas. Il dépose dans `arrivages`, ne peut rien créer ailleurs, et n'obtient pas de shell.

25.5 WebDAV, quand seul le Web passe

WebDAV ajoute à HTTP les verbes qui lui manquent pour gérer des fichiers : déposer, créer un dossier, déplacer, supprimer. Son intérêt est de passer partout où le

Web passe, par le port 443, avec le certificat que vous avez déjà. Windows, macOS et le gestionnaire de fichiers d'Ubuntu savent s'y connecter.

Nginx le propose à l'aide d'un module à ajouter. Les mots de passe sont rangés dans un fichier créé avec `htpasswd` :

```
gary@serveur:~$ sudo apt install libnginx-mod-http-dav-ext apache2-utils
[…]
gary@serveur:~$ sudo htpasswd -c /etc/nginx/dav.htpasswd gary
New password:
Re-type new password:
Adding password for user gary
gary@serveur:~$ sudo chown root:www-data /etc/nginx/dav.htpasswd
gary@serveur:~$ sudo chmod 640 /etc/nginx/dav.htpasswd
gary@serveur:~$ sudo install -d -o www-data -g www-data -m 750 /srv/dav
```

L'option `-c` crée le fichier : ne la remettez pas pour ajouter un second compte, elle l'écraserait. Le dossier appartient à `www-data`, puisque c'est Nginx qui y écrira. Dans le bloc `server` du site en HTTPS, on ajoute :

```
    location /dav/ {
        alias /srv/dav/;
        autoindex on;

        auth_basic "Depot";
        auth_basic_user_file /etc/nginx/dav.htpasswd;

        dav_methods PUT DELETE MKCOL COPY MOVE;
        dav_ext_methods PROPFIND OPTIONS;
        create_full_put_path on;
        client_max_body_size 200m;
    }
```

Après `nginx -t` et `reload`, essayons depuis le poste. `curl` lira le mot de passe dans le fichier `~/.netrc` grâce à l'option `-n` :

```
gary@vostro:~$ printf 'machine blog.maison.lan login gary password Citron-Vif-67\n' \
    > ~/.netrc
gary@vostro:~$ chmod 600 ~/.netrc
gary@vostro:~$ curl -sI https://blog.maison.lan/dav/ | head -n 1
HTTP/2 401
gary@vostro:~$ curl -n -s -o /dev/null -w '%{http_code}\n' -T /etc/os-release \
    https://blog.maison.lan/dav/os-release.txt
201
gary@vostro:~$ curl -n -s -o /dev/null -w '%{http_code}\n' -X MKCOL \
    https://blog.maison.lan/dav/archives/
201
gary@vostro:~$ curl -n -s -X PROPFIND -H 'Depth: 1' https://blog.maison.lan/dav/ | \
    grep -o '<D:href>[^<]*'
<D:href>/dav/
```

```
<D:href>/dav/os-release.txt
<D:href>/dav/archives/
gary@vostro:~$ curl -n -s -o /dev/null -w '%{http_code}\n' -X DELETE \
    https://blog.maison.lan/dav/os-release.txt
204
```

Sans mot de passe, le serveur répond 401. Avec, le dépôt d'un fichier et la création d'un dossier, par le verbe `MKCOL`, renvoient 201, « créé », et la suppression 204. `PROPFIND` est le verbe par lequel un client liste un dossier. Dans le gestionnaire de fichiers d'Ubuntu, l'adresse s'écrit `davs://blog.maison.lan/dav/`.

Ce WebDAV nu convient à un dépôt simple. Si vous cherchez un espace de fichiers complet, avec synchronisation, partage par lien et application mobile, c'est ce que construisent des logiciels comme Nextcloud, qui s'appuient sur le même protocole.

25.6 Lequel choisir

Situation	Choix
Des serveurs Linux qui partagent des données	NFS
Des postes Linux avec les mêmes comptes partout	NFS
Des postes Windows ou des Mac, une famille, un petit bureau	Samba
Un échange avec une machine distante ou un prestataire	SFTP
Un accès depuis l'extérieur, à travers des réseaux qui ne laissent passer que le Web	WebDAV

	NFS	Samba	SFTP	WebDAV
Se monte comme un disque	oui	oui	possible, avec `sshfs`	possible
Client fourni avec Windows	non	oui	en ligne de commande	oui
Mot de passe par utilisateur	non	oui	clé ou mot de passe	oui
À exposer sur Internet	jamais	jamais	oui	oui, en HTTPS
Port	2049	445	22	443

NFS et SMB sont faits pour un réseau local. Pour atteindre un partage depuis l'extérieur, passez par un VPN, ou servez-vous de SFTP.

En résumé

- NFS : un dossier et une ligne dans `/etc/exports`, `exportfs -ra`, le port 2049. Sur le poste, `mount -t nfs serveur:/dossier`.
- NFS se fie aux numéros d'utilisateur annoncés par le poste. On n'exporte que vers des machines de confiance, et l'on aligne les numéros.
- `root_squash` retire ses pouvoirs au `root` des postes.
- Dans `/etc/fstab`, `_netdev,nofail` pour un montage au démarrage, `x-systemd.automount` pour un montage à la demande.
- Samba : un partage par section de `smb.conf`, `testparm` pour vérifier, `smbpasswd -a` pour chaque utilisateur.
- Un partage Samba passe deux contrôles : le mot de passe Samba, puis les droits Unix du compte.
- `smbclient -L` pour voir les partages, `smbstatus` pour voir les connexions.
- SFTP enfermé : `Match Group`, `ChrootDirectory`, `ForceCommand internal-sftp`, et un dossier racine appartenant à `root`.

Atelier — Le serveur de fichiers de la maison

Objectif : offrir à deux personnes un dossier commun et un dossier personnel chacune, accessibles depuis tous les postes de la maison. Comptez une heure.

Préparer

1. Créez sur le serveur deux comptes Unix sans shell, `lea` et `tom`, avec `useradd -m -s /usr/sbin/nologin`, et un groupe `famille`. Mettez-y les deux comptes.
2. Créez `/srv/famille`, appartenant au groupe `famille`, en mode 2770.

Configurer

3. Installez Samba. Gardez une copie du fichier d'origine.
4. Écrivez un `smb.conf` court, avec un partage `[famille]` réservé au groupe et la section `[homes]`.
5. Vérifiez-le avec `testparm`, puis donnez un mot de passe Samba à chaque compte.
6. Ouvrez le pare-feu au réseau local, et à lui seul.

Essayer

7. Depuis un poste Linux, listez les partages avec `smbclient -L` sous le nom de `lea`. Voyez-vous le dossier personnel de `tom` ?
8. Sous `lea`, déposez un fichier dans `famille`. Sous `tom`, vérifiez que vous pouvez le lire et le modifier.
9. Sous `tom`, tentez d'ouvrir le dossier personnel de `lea`. Notez le message.
10. Si vous avez un poste Windows ou un Mac, connectez-vous au partage et déposez un fichier. Sur le serveur, regardez à qui il appartient et quels sont ses droits.

Rendre permanent

11. Sur le poste Linux, montez le partage `famille` à la demande, par une ligne de `/etc/fstab` avec un fichier d'identifiants en mode 600.

Casser, pour voir

12. Sur le serveur, passez `/srv/famille` en mode 2750 et essayez de déposer un fichier depuis le poste. Le mot de passe est accepté : pourquoi l'écriture échoue-t-elle ? Rétablissez le mode 2770.

Vérifier

- `sudo smbstatus --shares` montre vos connexions pendant les essais.
- Les fichiers créés dans `famille` appartiennent au groupe `famille`, quel que soit leur auteur.
- `sudo ufw status` ne montre le profil Samba que pour le réseau local.

Exercices

Exercice 1 — Lire /etc/exports

```
/srv/projets   192.168.1.0/24(rw,sync,no_subtree_check)
/srv/archives  192.168.1.0/24(ro,sync,no_subtree_check)
/srv/outils    192.168.1.20(rw,sync,no_subtree_check,no_root_squash)
```

1. Quelles machines peuvent écrire dans `/srv/projets` ?
2. Que peut faire le poste `192.168.1.30` dans `/srv/archives` ? Dans `/srv/outils` ?
3. Que change `no_root_squash` pour la machine `192.168.1.20` ? Dans quel cas l'accepteriez-vous ?

4. Après avoir modifié ce fichier, quelle commande faut-il lancer ?

Exercice 2 — Les numéros

Sur le serveur, `alice` porte le numéro 1001 et `bruno` le 1002. Sur le poste, c'est l'inverse.

1. `alice`, sur le poste, crée un fichier dans le partage NFS. À qui appartient-il sur le serveur ?
2. Peut-elle lire, depuis le poste, un fichier que `bruno` a créé sur le serveur en mode 600 ?
3. Proposez deux façons de corriger la situation.

Exercice 3 — Deux messages Samba

1. `session setup failed: NT_STATUS_LOGON_FAILURE`
2. `tree connect failed: NT_STATUS_ACCESS_DENIED`

Pour chacun : quel contrôle a échoué, et quelles vérifications faites-vous sur le serveur ?

Exercice 4 — Un dépôt SFTP

Créez un compte `fournisseur`, enfermé dans `/srv/depot/fournisseur`, avec un sous-dossier `factures` où il peut écrire.

1. Vérifiez qu'il ne peut pas ouvrir de shell.
2. Vérifiez qu'il ne voit pas `/etc`.
3. Que se passe-t-il si `/srv/depot/fournisseur` appartient au compte lui-même ? Essayez, lisez le journal de SSH, puis corrigez.

Exercice 5 — Choisir

Quel protocole proposez-vous, et pourquoi ?

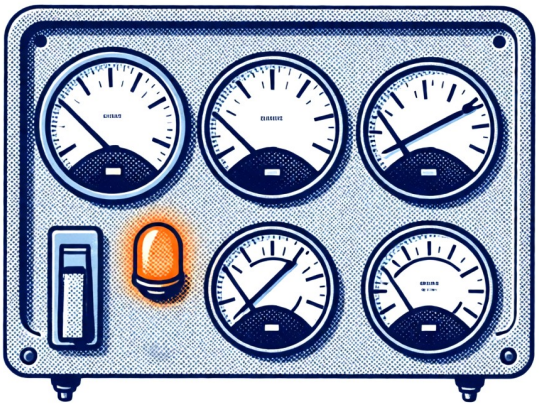
1. Trois serveurs web doivent lire les mêmes images.
2. Un cabinet de quatre personnes, sous Windows, veut un dossier commun.
3. Un photographe doit vous envoyer deux gigaoctets de fichiers chaque semaine.
4. Vous voulez consulter vos documents depuis le Wi-Fi d'un hôtel.

Pour aller plus loin

Sur la machine. `man exports` détaille toutes les options d'un export, `man nfs` celles d'un montage. `man smb.conf` est immense : cherchez-y un réglage par son nom. `man sshd_config` décrit `ChrootDirectory` et les blocs `Match`.

Paquets. `wsdd` fait apparaître le serveur Samba dans le voisinage réseau des postes Windows. `sshfs` monte un dossier distant à travers SSH, sans rien installer sur le serveur. `autofs` est l'ancien outil de montage à la demande, encore répandu. `davfs2` monte un dossier WebDAV.

En ligne. La documentation d'Ubuntu Server couvre NFS, Samba et son intégration à Active Directory. Le wiki de Samba, `wiki.samba.org`, est la référence du projet.



CHAPITRE 26

Superviser un serveur

Un serveur ne prévient pas. Il ralentit, se remplit, s'arrête, et l'on s'en aperçoit quand quelqu'un se plaint. Superviser, c'est savoir avant les autres. On commence par lire les chiffres que la machine donne d'elle-même, puis on les confie à un outil qui les garde, les dessine et sonne l'alarme.

DANS CE CHAPITRE

- Lire la charge, la mémoire, l'activité des disques et du réseau.
- Distinguer la mémoire utilisée de la mémoire disponible.
- Retrouver l'historique d'une machine avec `sar`.
- Installer Prometheus et Node Exporter, et les interroger.
- Écrire des règles d'alerte et recevoir les alertes par courriel.
- Être prévenu quand une sauvegarde ne donne plus signe de vie.
- Afficher un tableau de bord Grafana sans l'exposer au réseau.
- Surveiller un site depuis une autre machine.

> **AVANT DE COMMENCER**
>
> Les chapitres 12, 13, 16, 19 et 23. Le serveur fait tourner Nginx et la sauvegarde restic du chapitre 16. Les essais de charge utilisent `stress-ng`, à installer avec `sudo apt install stress-ng`.

26.1 Ce qu'on surveille, et pourquoi

Question	Ce qu'on mesure
Le service répond-il ?	Le site, le port, le processus
La machine tient-elle la charge ?	Le processeur, la mémoire, les disques, le réseau
Va-t-elle manquer de quelque chose ?	La place libre, et la vitesse à laquelle elle diminue
Les tâches de fond ont-elles tourné ?	Les sauvegardes, les renouvellements de certificat
Se passe-t-il quelque chose d'anormal ?	Les connexions refusées, les erreurs dans les journaux

Chaque mesure a deux usages. Sur le moment, elle explique une lenteur. Sur la durée, elle montre une tendance : un disque qui se remplit de deux pour cent par semaine sera plein dans quelques mois, et on a le temps d'agir.

26.2 Lire la machine sur le moment

La charge

```
gary@serveur:~$ uptime
 00:29:09 up  2:09,  1 user,  load average: 0.08, 0.03, 0.00
gary@serveur:~$ nproc
2
```

La charge moyenne, vue au chapitre 12, se lit en la comparant au nombre de processeurs. Faisons travailler les deux processeurs du serveur pendant une minute :

```
gary@serveur:~$ stress-ng --cpu 2 --timeout 70s > /dev/null 2>&1 &
[…]
gary@serveur:~$ uptime
 00:30:16 up  2:10,  1 user,  load average: 1.35, 0.42, 0.14
gary@serveur:~$ top -bn1 -o %CPU | sed -n '7,10p'
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
```

```
 200254 gary      20   0   60784   5580   2460 R  90.9   0.4   1:05.83 stress-+
 200255 gary      20   0   60784   5580   2460 R  90.9   0.4   1:05.85 stress-+
      1 root      20   0   23292  14592   9636 S   0.0   1.0   0:18.69 systemd
```

La charge d'une minute monte vers 2, celle de quinze minutes bouge à peine. C'est la lecture des trois chiffres : le premier dit ce qui se passe, le dernier si cela dure. Une pointe passagère ne mérite pas qu'on s'en occupe. Une charge de quinze minutes durablement supérieure au nombre de processeurs, si.

`vmstat` donne une ligne de chiffres toutes les deux secondes :

```
gary@serveur:~$ vmstat 2 3
procs -----------memory---------- ---swap-- -----io---- -system-- -------cpu-------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st gu
 2  0   1036 183740  51240 1000872    0    0   144   509  289    1  2  1 97  0  0  0
 2  0   1036 183740  51240 1000912    0    0     0     0 2118  186 100  0  0  0  0  0
 0  0   1036 191804  51240 984364    0    0    64     0 1760   90 86  0 14  0  0  0
```

La première ligne fait la moyenne depuis le démarrage : on l'ignore. Ensuite, `r` compte les processus qui veulent un processeur, `us` et `id` donnent le temps passé dans les programmes et le temps inoccupé. Pendant l'essai, le processeur est occupé à 100 %.

La mémoire

Le chapitre 12 avait laissé de côté les deux lignes de `top` consacrées à la mémoire. Les voici :

```
gary@serveur:~$ top -bn1 | head -n 5
top - 00:29:10 up  2:09,  1 user,  load average: 0.08, 0.03, 0.00
Tasks: 185 total,   1 running, 184 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  4.0 sy,  0.0 ni, 88.0 id,  4.0 wa,  0.0 hi,  4.0 si,  0.0 st
MiB Mem :   1457.6 total,    179.4 free,    494.7 used,   1011.0 buff/cache
MiB Swap:   2048.0 total,   2047.0 free,      1.0 used.    962.9 avail Mem
gary@serveur:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           1.4Gi       494Mi       179Mi        45Mi       1.0Gi       962Mi
Swap:          2.0Gi       1.0Mi       2.0Gi
```

`free -h` présente les mêmes chiffres en colonnes. À première vue, la machine n'a plus que 179 Mo de libres. C'est la lecture qui inquiète à tort.

Colonne	Sens
`used`	La mémoire que les programmes occupent
`free`	La mémoire qui ne sert à rien du tout

Colonne	Sens
`buff/cache`	La mémoire prêtée au cache des fichiers
`available`	Ce qu'un programme peut obtenir s'il en demande

Linux ne laisse pas la mémoire inemployée. Il s'en sert pour garder en mémoire les fichiers déjà lus, ce qui évite de les relire sur le disque. Dès qu'un programme a besoin de place, il rend ce cache. Le chiffre qui compte est donc `available` : 962 Mo, les deux tiers de la machine.

Remplissons la mémoire pour voir ce qui se passe vraiment :

```
gary@serveur:~$ stress-ng --vm 1 --vm-bytes 900M --vm-keep --timeout 40s > /dev/null 2>&1 &
[…]
gary@serveur:~$ free -h
               total        used        free      shared  buff/cache   available
Mem:           1.4Gi       1.3Gi       101Mi        18Mi       164Mi       102Mi
Swap:          2.0Gi        87Mi       1.9Gi
```

Le cache a fondu, de 1 Go à 164 Mo : il a cédé sa place. `available` tombe à 102 Mo, et le système commence à écrire dans l'espace d'échange, le *swap* du chapitre 14. Dans `vmstat`, les colonnes `si` et `so` comptent ces échanges. Un peu de *swap* utilisé n'est pas un problème. Des échanges continus, si : la machine passe son temps à déplacer de la mémoire au lieu de travailler.

Quand même l'échange ne suffit plus, le noyau choisit un processus et l'arrête. C'est l'*OOM killer*, et il laisse une trace dans `journalctl -k`. Une alerte sur la mémoire disponible sert à intervenir avant lui.

Les disques

Deux questions différentes : reste-t-il de la place, et les disques suivent-ils ?

```
gary@serveur:~$ df -h -x tmpfs -x devtmpfs
Filesystem                         Size  Used Avail Use% Mounted on
[…]
/dev/mapper/ubuntu--vg-ubuntu--lv   16G  5.3G  9.7G  36% /
/dev/mapper/vg--miroir-donnees     2.9G   33M  2.7G   2% /srv/miroir/donnees
[…]
/dev/sda1                          5.9G   44K  5.8G   1% /srv/donnees
/dev/mapper/coffre                 4.9G   24K  4.6G   1% /srv/coffre
gary@serveur:~$ df -i / | tail -n 1
/dev/mapper/ubuntu--vg-ubuntu--lv 1048576 97711 950865   10% /
```

`df -i` compte les inodes, vus au chapitre 4. Un système de fichiers peut refuser d'écrire alors qu'il reste de la place, parce qu'il n'a plus d'inodes : c'est le cas des dossiers où s'accumulent des millions de petits fichiers.

Pour l'activité, `iostat`, du paquet `sysstat`, mesure chaque disque. Écrivons sans arrêt sur `/srv/donnees` pendant la mesure :

```
gary@serveur:~$ iostat -dxs sda 2 2
Linux 6.8.0-142-generic (serveur)       10/05/26        _x86_64_        (2
CPU)

Device             tps      kB/s    rqm/s   await  areq-sz  aqu-sz  %util
sda               2.45   4881.66     0.05    0.57  1992.11    0.00   0.13


Device             tps      kB/s    rqm/s   await  areq-sz  aqu-sz  %util
sda             409.50 418322.00     3.50    0.63  1021.54    0.27  22.50
```

Colonne	Sens
`tps`	Opérations par seconde
`kB/s`	Débit
`await`	Temps moyen d'une opération, en millisecondes
`aqu-sz`	Longueur de la file d'attente
`%util`	Part du temps où le disque a travaillé

Le disque écrit 400 Mo par seconde, avec une attente de moins d'une milliseconde : il suit sans peine. Le signe d'un disque saturé est un `await` qui grimpe à des dizaines de millisecondes, avec une file qui s'allonge. Au même moment, `top` montre la colonne `wa`, le temps que le processeur passe à attendre le disque :

```
gary@serveur:~$ top -bn1 | sed -n 3p
%Cpu(s):  0.0 us, 42.9 sy,  0.0 ni, 47.6 id,  9.5 wa,  0.0 hi,  0.0 si,  0.0 st
```

Pour savoir quel programme sollicite le disque, `iotop` trie les processus par débit, comme `top` le fait pour le processeur. Sur Ubuntu, le paquet `iotop-c` le fournit. Il demande d'activer une mesure du noyau : `sudo sysctl kernel.task_delayacct=1`.

Le réseau

Outil	Ce qu'il montre
`ss -tunp`	Les connexions ouvertes et les programmes qui les tiennent (chapitre 17)
`iftop`	Le débit par connexion, en temps réel
`nethogs`	Le débit par programme
`vnstat`	Le volume échangé par heure, par jour, par mois

Les deux premiers outils de cette liste après `ss` s'utilisent en direct, plein écran. `vnstat` est différent : un service compte en permanence, et l'on consulte le cumul quand on veut.

```
gary@serveur:~$ vnstat -h | tail -n 6

         hour        rx      |     tx      |    total    |   avg. rate
     ------------------------+-------------+-------------+---------------
     2026-10-05
         00:00    550.68 MiB |   10.28 MiB |  560.96 MiB |    1.31 Mbit/s
     ------------------------+-------------+-------------+---------------
```

Les 550 Mo reçus dans l'heure correspondent aux installations de paquets de ce chapitre. Sur un serveur loué, dont le trafic est parfois limité, `vnstat -m` dit où l'on en est du mois.

Les tableaux de bord du terminal

Plusieurs programmes réunissent ces mesures sur un seul écran. Tous s'installent avec `apt`.

Programme	Particularité
`htop`	`top` en couleurs, à la souris, avec l'arbre des processus
`btop`	Processeur, mémoire, disques et réseau sur un écran, avec des graphiques
`atop`	Enregistre l'état de la machine à intervalles réguliers, pour l'examiner après coup
`glances`	Tout sur un écran, avec des seuils colorés

Prenez celui qui vous plaît. Ils ne montrent rien que les commandes précédentes ne donnent, mais ils le montrent d'un coup d'œil.

26.3 L'historique avec sar

Toutes ces commandes décrivent l'instant. Or, quand on vous signale une lenteur survenue à trois heures du matin, c'est l'historique qu'il faut. Le paquet `sysstat` enregistre l'activité toutes les dix minutes, et `sar` la restitue :

```
gary@serveur:~$ sar
Linux 6.8.0-142-generic (serveur)       10/05/26        _x86_64_        (2 CPU)

00:00:02        CPU     %user     %nice   %system   %iowait    %steal     %idle
00:10:00        all      0.88      1.01      2.22      0.52      0.02     95.35
00:20:03        all      0.55      0.05      0.39      0.84      0.02     98.15
00:30:17        all     11.78      0.16      0.68      0.36      0.02     87.00
Average:        all      4.47      0.41      1.09      0.57      0.02     93.45
```

La ligne de 00:30 garde la trace de notre essai de charge.

Commande	Historique de
`sar`	Le processeur
`sar -r`	La mémoire
`sar -q`	La charge
`sar -d`	Les disques
`sar -n DEV`	Le réseau
`sar -f /var/log/sysstat/sa04`	Une autre journée, ici le 4 du mois

> **SUR LE TERRAIN · La ligne ENABLED**
>
> Beaucoup de guides font passer `ENABLED` à `"true"` dans `/etc/default/sysstat`, sans quoi rien ne serait enregistré. Sur Ubuntu 24.04, la collecte est lancée par une minuterie systemd, `sysstat-collect.timer`, qui ne lit pas cette ligne. Notre serveur a enregistré son activité alors que le fichier disait `ENABLED="false"`. Vérifiez plutôt que la minuterie est active, avec `systemctl list-timers 'sysstat*'`.

26.4 Prometheus

Tout ce qui précède demande qu'on se connecte et qu'on regarde. La supervision commence quand un programme regarde à votre place, garde l'historique de tout, et vous prévient.

Prometheus est devenu l'outil de référence pour cela. Il interroge à intervalles réguliers des programmes qui exposent des mesures, appelés **exportateurs**, et range chaque valeur avec son heure. Il évalue des règles sur ces séries de valeurs, et confie les alertes à un second programme, **Alertmanager**, qui les trie et les envoie. **Grafana** dessine le tout.

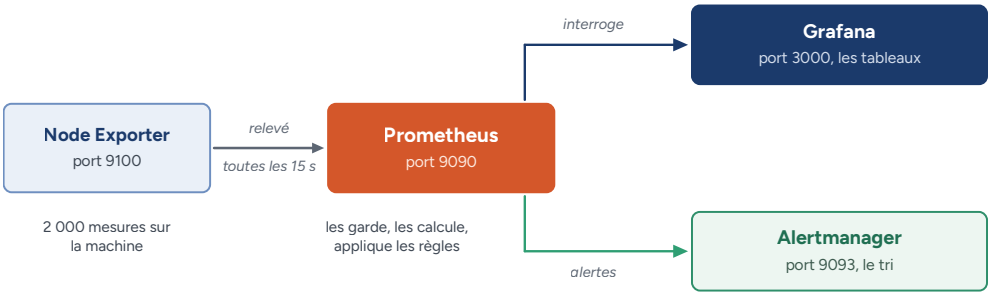


Figure 26.1 — Les éléments d'une supervision avec Prometheus.

> **REPÈRE HISTORIQUE · Né chez SoundCloud**
>
> Prometheus a été écrit à partir de 2012 chez SoundCloud, qui cherchait à surveiller des centaines de services changeants. Il a rejoint en 2016 la Cloud Native Computing Foundation, deuxième projet accueilli après Kubernetes. Grafana, apparu en 2014, est devenu son compagnon habituel. Les deux sont libres.

Installer

Prometheus, l'exportateur de la machine et Alertmanager sont dans les dépôts d'Ubuntu :

```
gary@serveur:~$ sudo apt install prometheus prometheus-node-exporter prometheus-alertmanager
[…]
gary@serveur:~$ sudo ss -ltnp | grep -E ':(9090|9093|9100) ' | cut -c1-80
LISTEN 0      4096               *:9090             *:*    users:(("prometheus",
LISTEN 0      4096               *:9093             *:*    users:(("prometheus-a
LISTEN 0      4096               *:9100             *:*    users:(("prometheus-n
```

Les trois services écoutent sur toutes les adresses, sans mot de passe. Le pare-feu du chapitre 21 les protège : ces ports n'y sont pas ouverts, et ne doivent pas l'être.

Les mesures

Node Exporter publie l'état de la machine sous forme de texte, à l'adresse `/metrics` :

```
gary@serveur:~$ curl -s localhost:9100/metrics | grep -c '^node_'
2168
gary@serveur:~$ curl -s localhost:9100/metrics | grep -B2 '^node_load1 '
# HELP node_load1 1m load average.
# TYPE node_load1 gauge
node_load1 0.18
```

Plus de deux mille mesures, chacune avec son nom, une ligne d'aide et sa valeur. On reconnaît ici la charge d'une minute. Prometheus vient lire cette page toutes les quinze secondes : c'est ce que dit son fichier de configuration, `/etc/prometheus/prometheus.yml`. Nous le réécrivons pour qu'il soit court et qu'il charge nos règles :

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

alerting:
  alertmanagers:
    - static_configs:
        - targets: ['localhost:9093']

rule_files:
  - /etc/prometheus/regles.yml

scrape_configs:
  - job_name: prometheus
    static_configs:
      - targets: ['localhost:9090']

  - job_name: node
    static_configs:
      - targets: ['localhost:9100']
```

Chaque `job` désigne une famille de cibles. Prometheus se surveille lui-même, et surveille Node Exporter.

Interroger

Prometheus a son langage de requêtes, PromQL. `promtool`, livré avec lui, permet de l'essayer dans le terminal :

```
gary@serveur:~$ promtool query instant http://localhost:9090 up
up{instance="localhost:9090", job="prometheus"} => 1 @[1791160622.814]
up{instance="localhost:9100", job="node"} => 1 @[1791160622.814]
```

`up` vaut 1 pour chaque cible que Prometheus a pu lire, 0 sinon. Entre accolades figurent les **étiquettes**, qui distinguent les séries d'une même mesure. On s'en sert pour filtrer :

```
gary@serveur:~$ promtool query instant http://localhost:9090 \
    'node_filesystem_avail_bytes{mountpoint="/"}'
node_filesystem_avail_bytes{device="/dev/mapper/ubuntu--vg-ubuntu--lv", fstype="ext4",
  instance="localhost:9100", job="node", mountpoint="/"} => 9893072896 @[…]
```

Et pour calculer. La part de mémoire occupée, au sens de la colonne `available` :

```
gary@serveur:~$ promtool query instant http://localhost:9090 \
    '100 * (1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)'
{instance="localhost:9100", job="node"} => 30.74412235261329 @[1791160623.587]
```

Certaines mesures sont des compteurs, qui ne font que croître : les octets reçus depuis le démarrage, par exemple. La fonction `rate` en tire une vitesse :

```
gary@serveur:~$ promtool query instant http://localhost:9090 \
    'rate(node_network_receive_bytes_total{device="enp0s3"}[5m])'
{device="enp0s3", instance="localhost:9100", job="node"} => 115.25978 @[…]
```

En moyenne sur les cinq dernières minutes, l'interface reçoit 115 octets par seconde. L'interface web de Prometheus, sur le port 9090, propose les mêmes requêtes avec un graphique.

26.5 Les alertes

Des règles

Une règle d'alerte est une requête, un seuil et une durée. Voici `/etc/prometheus/regles.yml` :

```
groups:
  - name: serveur
    rules:
      - alert: MemoireHaute
        expr: 100 * (1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) > 80
        for: 1m
        labels:
          severity: avertissement
        annotations:
          summary: "Mémoire occupée à {{ $value | printf \"%.0f\" }} % sur {{ $labels.insta
nce }}"

      - alert: DisqueBientotPlein
        expr: node_filesystem_avail_bytes{fstype=~"ext4|xfs"} / node_filesystem_size_bytes
< 0.10
```

```
        for: 15m
        labels:
          severity: avertissement
        annotations:
          summary: "Moins de 10 % de place sur {{ $labels.mountpoint }}"

      - alert: CibleInjoignable
        expr: up == 0
        for: 2m
        labels:
          severity: critique
        annotations:
          summary: "{{ $labels.job }} ne répond plus sur {{ $labels.instance }}"

      - alert: SauvegardeEnRetard
        expr: time() - sauvegarde_derniere_reussite_secondes > 26 * 3600
        labels:
          severity: critique
        annotations:
          summary: "Aucune sauvegarde réussie depuis plus de 26 heures"

      - alert: SauvegardeJamaisSignalee
        expr: absent(sauvegarde_derniere_reussite_secondes)
        for: 1h
        labels:
          severity: critique
        annotations:
          summary: "Le signe de vie de la sauvegarde est introuvable"
```

Champ	Rôle
`expr`	La condition. L'alerte existe tant que la requête renvoie quelque chose
`for`	La durée pendant laquelle la condition doit tenir avant qu'on alerte
`labels`	Des étiquettes pour trier, ici la gravité
`annotations`	Le texte du message. `$value` et `$labels` y sont remplacés

La durée évite d'être réveillé par une pointe de quelques secondes. Les deux dernières règles concernent la sauvegarde, et nous y viendrons.

```
gary@serveur:~$ promtool check config /etc/prometheus/prometheus.yml
Checking /etc/prometheus/prometheus.yml
  SUCCESS: 1 rule files found
 SUCCESS: /etc/prometheus/prometheus.yml is valid prometheus config file syntax

Checking /etc/prometheus/regles.yml
  SUCCESS: 5 rules found

gary@serveur:~$ sudo systemctl reload prometheus
```

Les acheminer

Alertmanager reçoit les alertes, regroupe celles qui se ressemblent, évite de répéter la même toutes les quinze secondes, et les envoie. Sa configuration, dans `/etc/prometheus/alertmanager.yml` :

```
global:
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertes@serveur.maison.lan'
  smtp_require_tls: false

route:
  receiver: courriel
  group_by: ['alertname']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h

receivers:
  - name: courriel
    email_configs:
      - to: 'gary@localhost'
        send_resolved: true
```

Réglage	Rôle
`group_wait`	Attendre un peu avant d'envoyer, pour regrouper les alertes qui arrivent ensemble
`group_interval`	Délai avant de signaler un changement dans un groupe déjà envoyé
`repeat_interval`	Rappeler une alerte toujours active
`send_resolved`	Prévenir aussi quand le problème a disparu

Les courriels partent par le serveur de messagerie de la machine. Sur notre serveur d'essai, Postfix est installé en mode « local seulement » : il ne distribue qu'aux comptes de la machine, ce qui suffit pour vérifier la chaîne. Sur un vrai serveur, `smtp_smarthost` désigne le serveur d'envoi de votre fournisseur de messagerie, avec `smtp_auth_username` et `smtp_auth_password`. Alertmanager sait aussi écrire à une messagerie instantanée : Telegram, Slack, Matrix, ou tout service qui accepte une requête HTTP.

```
gary@serveur:~$ amtool check-config /etc/prometheus/alertmanager.yml
Checking '/etc/prometheus/alertmanager.yml'  SUCCESS
Found:
 - global config
```

```
  - route
  - 0 inhibit rules
  - 1 receivers
  - 0 templates

gary@serveur:~$ sudo systemctl restart prometheus-alertmanager
```

L'épreuve

Occupons la mémoire pendant trois minutes, et suivons l'alerte :

```
gary@serveur:~$ stress-ng --vm 1 --vm-bytes 900M --vm-keep --timeout 200s > /dev/null 2>&1 &
[1] 203217
gary@serveur:~$ promtool query instant http://localhost:9090 \
    'ALERTS{alertname="MemoireHaute"}'
ALERTS{alertname="MemoireHaute", alertstate="pending", instance="localhost:9100",
  job="node", severity="avertissement"} => 1 @[1791160779.621]
```

L'alerte est `pending` : la condition est remplie, mais la minute exigée par `for` n'est pas écoulée. Une minute plus tard :

```
gary@serveur:~$ promtool query instant http://localhost:9090 \
    'ALERTS{alertname="MemoireHaute"}'
ALERTS{alertname="MemoireHaute", alertstate="firing", instance="localhost:9100",
  job="node", severity="avertissement"} => 1 @[1791160840.034]
gary@serveur:~$ amtool alert query --alertmanager.url=http://localhost:9093
Alertname     Starts At                Summary                                  State
MemoireHaute  2026-10-05 00:40:19 UTC  Mémoire occupée à 87 % sur localhost:9100  active
```

`firing` : l'alerte est déclenchée, et Alertmanager l'a reçue. Le courriel suit :

```
gary@serveur:~$ grep -E '^(From|To|Subject):' /var/mail/gary
To: gary@localhost
From: alertes@serveur.maison.lan
Subject: [FIRING:1] MemoireHaute (localhost:9100 node avertissement)
```

Quand `stress-ng` s'arrête, la mémoire se libère et l'alerte disparaît. Quelques minutes plus tard, un second message l'annonce :

```
gary@serveur:~$ grep '^Subject:' /var/mail/gary
Subject: [FIRING:1] MemoireHaute (localhost:9100 node avertissement)
Subject: [RESOLVED] MemoireHaute (localhost:9100 node avertissement)
```

> **ASTUCE · Peu d'alertes, mais toutes utiles**
>
> Une alerte qui sonne pour rien apprend à ne plus les lire. Ne gardez que celles qui demandent une action, et réglez les seuils après quelques semaines d'observation. Le reste a sa place dans un tableau de bord, que l'on consulte quand on veut.

26.6 Le signe de vie

Le chapitre 16 l'avait annoncé : une sauvegarde peut cesser de fonctionner sans rien dire. Si la machine est éteinte, si la minuterie a été désactivée, si le service ne démarre même plus, aucune erreur n'est écrite nulle part. On ne peut pas guetter une erreur qui n'arrive pas. On inverse donc la logique : la sauvegarde annonce chaque réussite, et c'est l'absence d'annonce qui déclenche l'alerte.

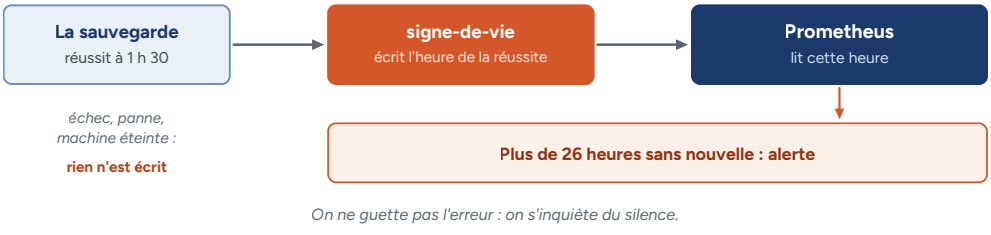


Figure 26.2 — Le principe du signe de vie.

Node Exporter sait lire des mesures déposées dans des fichiers texte, sur Ubuntu dans `/var/lib/prometheus/node-exporter/`. Un petit script y écrit l'heure de la dernière réussite :

```
#!/bin/bash
#
# signe-de-vie — note l'heure d'une réussite, pour Prometheus
# Usage : signe-de-vie NOM   (appelé quand une tâche a réussi)

set -euo pipefail

nom="$1"
dossier=/var/lib/prometheus/node-exporter
fichier="$dossier/$nom.prom"

# écrire à côté, puis renommer : Node Exporter ne lit jamais un fichier à moitié écrit
echo "${nom}_derniere_reussite_secondes $(date +%s)" > "$fichier.tmp"
mv "$fichier.tmp" "$fichier"
```

On l'installe dans `/usr/local/bin`, puis on demande à systemd de le lancer après chaque sauvegarde réussie, par un fichier de surcharge :

```
gary@serveur:~$ cat /etc/systemd/system/sauvegarde-restic.service.d/signe-de-vie.conf
[Service]
ExecStartPost=/usr/local/bin/signe-de-vie sauvegarde
```

Pour un service de type `oneshot`, `ExecStartPost` ne s'exécute que si `ExecStart` a réussi. Une sauvegarde en échec ne laisse donc aucune trace. Essayons :

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl start sauvegarde-restic.service
gary@serveur:~$ systemctl show -p Result sauvegarde-restic.service
Result=success
gary@serveur:~$ cat /var/lib/prometheus/node-exporter/sauvegarde.prom
sauvegarde_derniere_reussite_secondes 1791160709
gary@serveur:~$ promtool query instant http://localhost:9090 \
    'time() - sauvegarde_derniere_reussite_secondes'
{instance="localhost:9100", job="node"} => 21.592999935150146 @[1791160730.593]
```

La dernière sauvegarde réussie date de vingt et une secondes. Les deux dernières règles de `regles.yml` veillent sur ce chiffre. `SauvegardeEnRetard` se déclenche au-delà de vingt-six heures : une sauvegarde quotidienne, plus une marge. `SauvegardeJamaisSignalee` couvre le cas où la mesure n'existe pas, parce que le script n'a jamais tourné ou que son fichier a disparu. Avant le premier passage de la sauvegarde, c'est d'ailleurs elle que Prometheus signalait :

```
gary@serveur:~$ promtool query instant http://localhost:9090 'ALERTS'
ALERTS{alertname="SauvegardeJamaisSignalee", alertstate="pending",
  severity="critique"} => 1 @[…]
```

Le même script sert à toute tâche périodique : l'export des bases du chapitre 24, le renouvellement des certificats, une synchronisation. Il suffit de l'appeler avec un autre nom.

26.7 Grafana

Installer

Grafana n'est pas dans les dépôts d'Ubuntu. Son éditeur tient un dépôt, que l'on ajoute avec sa clé de signature, comme au chapitre 7 :

```
gary@serveur:~$ sudo mkdir -p /etc/apt/keyrings
gary@serveur:~$ wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | \
    sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null
gary@serveur:~$ echo 'deb [signed-by=/etc/apt/keyrings/grafana.gpg] \
    https://apt.grafana.com stable main' | sudo tee /etc/apt/sources.list.d/grafana.list
deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main
gary@serveur:~$ sudo apt update
[…]
gary@serveur:~$ sudo apt install grafana
[…]
gary@serveur:~$ grafana server -v
Version 13.2.3 (commit: 90ffed056f0884267356c12a0eeb72a022af53f1, […])
gary@serveur:~$ sudo systemctl enable --now grafana-server
```

Grafana écoute le port 3000.

Le raccorder par des fichiers

Tout se règle à la souris dans Grafana, mais on peut aussi le décrire dans des fichiers, que Grafana lit au démarrage. C'est plus long à écrire la première fois, et l'on sait ensuite exactement comment la machine est configurée. La source de données, dans `/etc/grafana/provisioning/datasources/prometheus.yml` :

```
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://localhost:9090
    isDefault: true
```

Le dossier des tableaux de bord, dans `/etc/grafana/provisioning/dashboards/tableaux.yml` :

```
apiVersion: 1

providers:
  - name: tableaux
    type: file
    options:
      path: /var/lib/grafana/tableaux
```

Inutile de dessiner soi-même un tableau de bord pour Node Exporter : la communauté en partage de très complets sur le site de Grafana. Le plus utilisé porte le numéro 1860.

```
gary@serveur:~$ sudo install -d -o grafana -g grafana /var/lib/grafana/tableaux
gary@serveur:~$ sudo curl -sL -o /var/lib/grafana/tableaux/node-exporter-full.json \
    https://grafana.com/api/dashboards/1860/revisions/latest/download
gary@serveur:~$ sudo systemctl restart grafana-server
```

Le compte `admin` a pour mot de passe `admin` à l'installation. Changez-le avant tout. La commande lit le nouveau sur son entrée, pour qu'il n'apparaisse pas dans l'historique :

```
gary@serveur:~$ read -rs MDP
gary@serveur:~$ echo "$MDP" | sudo grafana cli admin reset-admin-password \
    --password-from-stdin
[…]
gary@serveur:~$ curl -s -u "admin:$MDP" 'localhost:3000/api/search?type=dash-db' | \
    python3 -m json.tool | grep '"title"'
        "title": "Node Exporter Full",
gary@serveur:~$ unset MDP
```

Le consulter sans l'exposer

Le port 3000 n'est pas ouvert dans le pare-feu, et il n'a pas à l'être. Le tunnel SSH du chapitre 19 l'amène jusqu'au poste :

```
gary@vostro:~$ curl -s --max-time 3 http://192.168.1.50:3000/api/health; echo "code $?"
code 28
gary@vostro:~$ ssh -fN -L 3000:localhost:3000 serveur
gary@vostro:~$ curl -s http://localhost:3000/api/health
{
  "database": "ok",
  "version": "13.2.3",
  "commit": "90ffed056f0884267356c12a0eeb72a022af53f1"
}
```

Directement, la connexion n'aboutit pas : le code 28 de `curl` signale un délai dépassé. Par le tunnel, Grafana répond. Dans le navigateur du poste, l'adresse est `http://localhost:3000`.



Figure 26.3 — Le haut du tableau de bord Node Exporter Full, sur une heure d'activité du serveur. On y retrouve les essais du chapitre : la charge du processeur et la mémoire occupée puis rendue.

Les heures affichées sont celles du poste qui consulte la page. Pour qu'une équipe y accède sans tunnel, on place Grafana derrière Nginx, en HTTPS, exactement comme l'application du chapitre 23.

26.8 Surveiller depuis l'extérieur

Toute cette installation a une faiblesse : elle tourne sur la machine qu'elle surveille. Si le serveur s'éteint, Prometheus s'éteint avec lui, et personne ne vous prévient. La disponibilité d'un service se vérifie depuis ailleurs.

Le moyen le plus simple est un script, sur une autre machine, qui interroge le site :

```
#!/bin/bash
#
# verifier-site — vérifie depuis l'extérieur qu'un site répond
# Usage : verifier-site URL   (lancé toutes les cinq minutes par cron)

url="$1"
code=$(curl -s -o /dev/null -w '%{http_code}' --max-time 10 "$url")

if [ "$code" != "200" ]; then
    logger -t verifier-site -p user.err "$url ne répond pas (code $code)"
    echo "$url ne répond pas (code $code)"
    exit 1
fi
```

```
gary@vostro:~$ verifier-site https://blog.maison.lan/; echo "code $?"
code 0
```

Arrêtons Nginx sur le serveur, et recommençons :

```
gary@vostro:~$ verifier-site https://blog.maison.lan/; echo "code $?"
https://blog.maison.lan/ ne répond pas (code 000)
code 1
```

Lancé toutes les cinq minutes par `cron`, il inscrit chaque panne dans le journal du poste. Pour recevoir un courriel, `cron` envoie au propriétaire de la tâche tout ce qu'elle écrit, si la machine sait distribuer le courrier :

```
*/5 * * * * /usr/local/bin/verifier-site https://blog.maison.lan/
```

Des outils font ce travail mieux, avec un historique et des notifications. **Uptime Kuma** offre une interface web pour suivre des dizaines de sites et de ports, et s'installe en une commande avec Docker, au chapitre 28. Des services en ligne, souvent gratuits pour quelques sites, font la même chose depuis plusieurs pays à la fois. Ils proposent aussi le signe de vie : votre sauvegarde appelle une adresse après chaque réussite, et le service s'inquiète quand l'appel ne vient plus. C'est la protection qui reste valable quand c'est la machine entière qui a disparu.

26.9 D'autres outils

Outil	Pour qui
Monit (`apt install monit`)	Surveille quelques services sur une machine, et sait les redémarrer seul. Léger, configuré par un fichier
Netdata (`apt install netdata`)	Des centaines de graphiques en temps réel sans rien configurer. Gourmand en mémoire
Zabbix, **Checkmk**	Des parcs de dizaines ou de centaines de machines, avec inventaire et cartes

Pour un ou deux serveurs, la combinaison de ce chapitre suffit largement : Node Exporter partout, un Prometheus, une vérification extérieure. Pour en ajouter un, on installe Node Exporter dessus et on ajoute son adresse aux `targets`, en n'ouvrant le port 9100 qu'à la machine qui héberge Prometheus.

En résumé

- La charge se compare au nombre de processeurs. Le troisième chiffre dit si elle dure.
- La mémoire qui compte est `available`. Le cache se libère tout seul.
- `iostat -dxs` mesure les disques : un `await` élevé signale un disque saturé. `df -h` et `df -i` disent ce qu'il reste.
- `sar` restitue l'historique enregistré par `sysstat`. Sur Ubuntu 24.04, la collecte tourne d'office.
- Node Exporter publie les mesures, Prometheus les garde et applique les règles, Alertmanager envoie les alertes, Grafana dessine.
- Une règle d'alerte : une requête, un seuil, une durée. `promtool check config` et `amtool check-config` vérifient avant de recharger.
- Le signe de vie : chaque réussite laisse une trace datée, et c'est son absence qui alerte.
- On consulte Grafana par un tunnel SSH ou derrière Nginx, jamais sur un port ouvert.
- La disponibilité se vérifie depuis une autre machine.

Atelier — Un serveur qui prévient

Objectif : superviser le serveur avec Prometheus et Grafana, recevoir une alerte quand la mémoire dépasse 80 %, et être prévenu si la sauvegarde cesse de tourner. Comptez une heure et demie.

Les mesures

1. Installez `prometheus`, `prometheus-node-exporter` et `prometheus-alertmanager`. Vérifiez avec `ss` les trois ports, et avec `sudo ufw status` qu'aucun n'est ouvert.
2. Remplacez `prometheus.yml` par la version du chapitre, et vérifiez avec `promtool query instant` que les deux cibles valent 1.
3. Écrivez la requête qui donne la place libre, en pour cent, sur `/srv/donnees`.

Les alertes

4. Installez les règles du chapitre. Vérifiez-les, rechargez.

5. Installez Postfix en mode « Local only » et `bsd-mailx`, puis la configuration d'Alertmanager.
6. Occupez la mémoire avec `stress-ng`. Suivez l'alerte de `pending` à `firing`. Vérifiez la réception du courriel, puis celle du message de fin.

Le signe de vie

7. Installez `signe-de-vie` et la surcharge de `sauvegarde-restic.service`. Lancez une sauvegarde et vérifiez la mesure.
8. Pour simuler deux jours sans sauvegarde, écrivez à la main dans le fichier `.prom` une heure vieille de 48 heures : `echo "sauvegarde_derniere_reussite_secondes $(date -d '-48 hours' +%s)"`. Quelle alerte se déclenche ? Relancez une sauvegarde, et vérifiez que l'alerte disparaît.

Le tableau de bord

9. Installez Grafana depuis son dépôt, avec la source de données et le tableau 1860 décrits par des fichiers. Changez le mot de passe d'administration.
10. Depuis le poste, ouvrez un tunnel SSH et affichez le tableau de bord. Retrouvez vos essais de charge.

Vérifier

- Votre boîte `/var/mail/gary` contient un message `[FIRING:1] MemoireHaute` et un message `[RESOLVED]`.
- `promtool query instant http://localhost:9090 ALERTS` ne renvoie rien quand tout va bien.
- Depuis le poste, `http://192.168.1.50:3000` ne répond pas, et `http://localhost:3000` répond à travers le tunnel.

Exercices

Exercice 1 — Lire free

```
               total        used        free      shared  buff/cache   available
Mem:           7.7Gi       2.1Gi       312Mi       150Mi       5.3Gi       5.2Gi
Swap:          2.0Gi       512Mi       1.5Gi
```

1. Un collègue s'alarme : il ne reste que 312 Mo. Que lui répondez-vous ?

2. Combien de mémoire un nouveau programme peut-il obtenir ?
3. Le *swap* est utilisé à moitié. Est-ce inquiétant ? Quelle commande vous le dirait ?

Exercice 2 — Trois chiffres

Pour un serveur à quatre processeurs, interprétez ces charges : `0.50, 0.40, 0.30` ; `7.80, 2.10, 0.90` ; `6.20, 6.50, 6.40`.

Exercice 3 — Requêtes

Écrivez la requête PromQL qui donne :

1. la place libre, en gigaoctets, sur la racine ;
2. le débit d'émission de l'interface `enp0s3`, en octets par seconde, sur cinq minutes ;
3. la liste des cibles injoignables ;
4. le nombre de secondes écoulées depuis le démarrage de la machine, sachant que `node_boot_time_seconds` donne l'heure du démarrage.

Exercice 4 — Une règle

Écrivez une règle `ChargeElevee` qui se déclenche quand la charge de quinze minutes dépasse le nombre de processeurs pendant dix minutes. Le nombre de processeurs s'obtient par `count(node_cpu_seconds_total{mode="idle"})`. Vérifiez-la avec `promtool check rules`.

Exercice 5 — Le silence

Appliquez le signe de vie à l'export des bases du chapitre 24 : modifiez son service, écrivez la règle correspondante, et vérifiez la mesure après un export. Quelle durée choisissez-vous pour le seuil, et pourquoi ?

Pour aller plus loin

Sur la machine. `man vmstat`, `man iostat`, `man sar` détaillent chaque colonne. `promtool --help` et `amtool --help` listent leurs commandes de vérification et d'interrogation. L'interface web de Prometheus, sur le port 9090 par un tunnel, montre l'état des cibles et des règles.

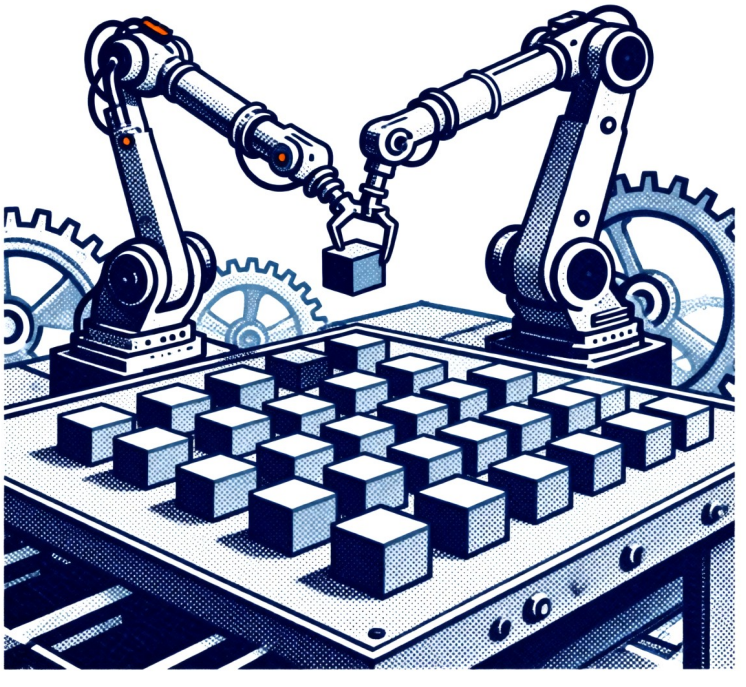
Paquets. `prometheus-blackbox-exporter` vérifie des sites et des ports depuis Prometheus, ce qui donne la vérification extérieure si on l'installe sur une autre machine. `prometheus-postgres-exporter` et `prometheus-nginx-exporter` publient les mesures de ces services. `smartmontools` surveille la santé physique des disques.

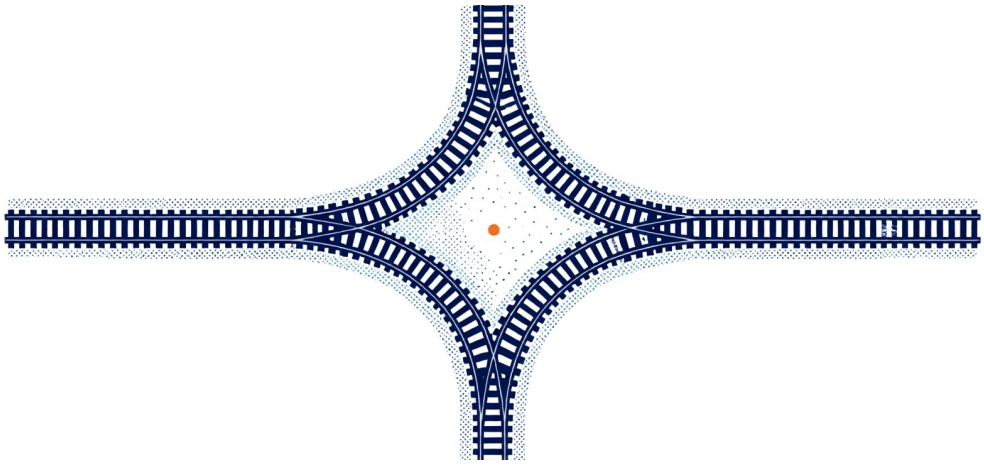
En ligne. La documentation de Prometheus, `prometheus.io/docs`, consacre une page claire aux bases de PromQL. Le recueil *Awesome Prometheus Alerts*, `samber.github.io/awesome-prometheus-alerts`, propose des règles toutes prêtes pour la plupart des services. Les tableaux de bord partagés se trouvent sur `grafana.com/grafana/dashboards`.

PARTIE IX

Automatisation et DevOps

Une initiation aux outils qui font passer de l'administration à la main à l'infrastructure reproductible : Git, Docker, Ansible.





CHAPITRE 27

Git et le suivi des versions

Un fichier modifié perd sa version précédente. Git les garde toutes, avec la date, l'auteur et la raison de chaque changement. Les développeurs ne s'en passent plus, et l'administrateur y gagne autant : ses scripts, ses fichiers de configuration, et même le dossier /etc entier ont une mémoire.

DANS CE CHAPITRE

- Configurer Git et créer un dépôt.
- Enregistrer des modifications, les lire, les annuler.
- Travailler dans une branche, fusionner, résoudre un conflit.
- Publier un dépôt sur un serveur et le partager.
- Faire vérifier ses scripts par shellcheck avant chaque enregistrement.
- Suivre les changements de /etc avec etckeeper.
- Choisir une plateforme d'hébergement.

> **AVANT DE COMMENCER**
>
> Les chapitres 8, 10 et 19. Le script `sauvegarde.sh` du chapitre 10 sert d'exemple. Le poste `vostro` se connecte au serveur par une clé SSH, comme au chapitre 19.

27.1 Pourquoi suivre les versions

Tout administrateur a connu le fichier `sshd_config.orig`, puis `sshd_config.orig2`, puis `sshd_config.bon`. Le chapitre 4 en conseillait l'usage, faute de mieux. Ces copies disent mal ce qui a changé, et pas du tout pourquoi.

Un **système de suivi de versions** enregistre l'état d'un ensemble de fichiers à chaque étape. Chaque étape porte un message, une date et un auteur. On peut comparer deux états, revenir à n'importe lequel, travailler à plusieurs sur les mêmes fichiers.

Git est devenu le standard. Chaque copie d'un dépôt contient tout l'historique, si bien qu'on travaille sans réseau et que chaque copie est aussi une sauvegarde.

> **REPÈRE HISTORIQUE · Écrit en dix jours**
>
> Jusqu'en 2005, le noyau Linux était développé avec BitKeeper, un logiciel propriétaire dont l'éditeur offrait l'usage aux développeurs du noyau. Quand cette gratuité a été retirée, Linus Torvalds a écrit son propre outil. Le premier enregistrement de Git date du 7 avril 2005, et le noyau a été géré par Git dès le mois de juin.

27.2 Installer et configurer

Git est souvent déjà présent sur un serveur. Sur un poste de travail, on l'installe :

```
gary@vostro:~$ sudo apt install git
[…]
gary@vostro:~$ git --version
git version 2.43.0
```

Avant le premier enregistrement, Git doit savoir qui vous êtes : votre nom et votre adresse figureront dans chaque commit.

```
gary@vostro:~$ git config --global user.name "Gary Nkweka"
gary@vostro:~$ git config --global user.email "gary@maison.lan"
gary@vostro:~$ git config --global init.defaultBranch main
gary@vostro:~$ git config --global core.editor nano
gary@vostro:~$ git config --global alias.lg "log --oneline --graph --decorate --all"
```

```
gary@vostro:~$ git config --global --list
user.name=Gary Nkweka
user.email=gary@maison.lan
init.defaultbranch=main
core.editor=nano
alias.lg=log --oneline --graph --decorate --all
```

Réglage	Rôle
`user.name`, `user.email`	L'auteur des commits
`init.defaultBranch`	Le nom de la branche principale des nouveaux dépôts
`core.editor`	L'éditeur ouvert quand Git demande un message
`alias.lg`	Un raccourci : `git lg` affichera l'historique en graphe

Ces réglages sont rangés dans `~/.gitconfig`. L'option `--global` les applique à tous vos dépôts.

27.3 Les trois zones

Git distingue trois endroits, et la plupart des confusions viennent de là.

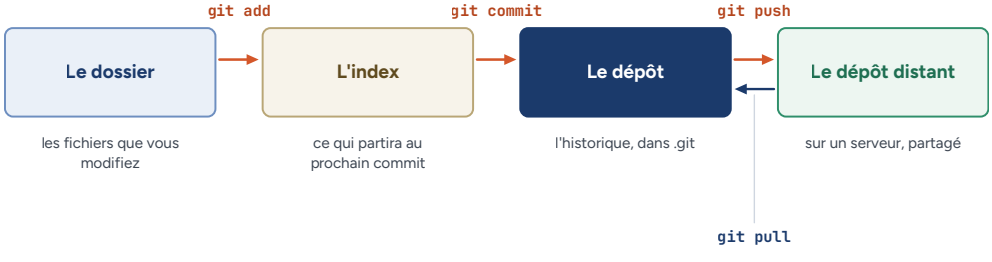


Figure 27.1 — Les zones de Git, et les commandes qui font passer de l'une à l'autre.

Le **dossier de travail** contient les fichiers tels que vous les modifiez. L'**index** prépare le prochain enregistrement : vous y placez, avec `git add`, les changements que vous voulez enregistrer ensemble. Le **dépôt**, caché dans le dossier `.git`, garde l'historique. `git commit` y range le contenu de l'index sous la forme d'un **commit** : un état complet du projet, avec son message.

27.4 Le premier dépôt

```
gary@vostro:~$ mkdir sauvegarde
gary@vostro:~$ cd sauvegarde
gary@vostro:~/sauvegarde$ git init
Initialized empty Git repository in /home/gary/sauvegarde/.git/
gary@vostro:~/sauvegarde$ cp ~/scripts/sauvegarde.sh .
gary@vostro:~/sauvegarde$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sauvegarde.sh

nothing added to commit but untracked files present (use "git add" to track)
```

`git status` est la commande à lancer sans cesse : elle dit où l'on en est, et propose souvent la commande suivante. Le script n'est pas encore suivi. Plaçons-le dans l'index, puis enregistrons :

```
gary@vostro:~/sauvegarde$ git add sauvegarde.sh
gary@vostro:~/sauvegarde$ git status --short
A  sauvegarde.sh
gary@vostro:~/sauvegarde$ git commit -m "Premier enregistrement du script de sauvegarde"
[main (root-commit) a04cb95] Premier enregistrement du script de sauvegarde
 1 file changed, 99 insertions(+)
 create mode 100644 sauvegarde.sh
gary@vostro:~/sauvegarde$ git log
commit a04cb950c71ff6e76a3bec3acebc7834daa35e71 (HEAD -> main)
Author: Gary Nkweka <gary@maison.lan>
Date:   Mon Oct 5 01:09:39 2026 +0000

    Premier enregistrement du script de sauvegarde
```

Chaque commit est désigné par une empreinte de quarante caractères, calculée à partir de son contenu. Les sept premiers suffisent à le désigner : `a04cb95`. `HEAD` indique où vous vous trouvez.

Ce qu'on ne suit pas

Un dépôt contient le source du projet, pas ce qu'il produit, ni les fichiers temporaires, ni les secrets. Le fichier `.gitignore` liste ce que Git doit ignorer :

```
# Archives produites par les essais
*.tar.gz

# Fichiers temporaires des éditeurs
```

```
*~
*.swp

# Réglages propres à chaque poste, jamais publiés
config.local
```

Ajoutons-le, avec un mode d'emploi, et créons une archive d'essai pour voir :

```
gary@vostro:~/sauvegarde$ touch essai.tar.gz
gary@vostro:~/sauvegarde$ git status --short
?? .gitignore
?? README.md
gary@vostro:~/sauvegarde$ git add README.md .gitignore
gary@vostro:~/sauvegarde$ git commit -m "Ajouter le mode d'emploi et la liste \
des fichiers ignorés"
[main c24f1e7] Ajouter le mode d'emploi et la liste des fichiers ignorés
 2 files changed, 22 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md
```

`essai.tar.gz` n'apparaît nulle part : Git l'ignore.

> **DANGER · Un secret enregistré est publié**
>
> Un mot de passe, une clé, un fichier `.env` enregistré par erreur reste dans l'historique, même si on le supprime au commit suivant. Une fois le dépôt publié, considérez le secret comme connu de tous et changez-le. Listez ces fichiers dans `.gitignore` dès la création du dépôt.

Modifier, comparer

```
gary@vostro:~/sauvegarde$ sed -i 's/^a_garder=7$/a_garder=10/' sauvegarde.sh
gary@vostro:~/sauvegarde$ git status --short
 M sauvegarde.sh
gary@vostro:~/sauvegarde$ git diff
diff --git a/sauvegarde.sh b/sauvegarde.sh
index a06fb97..df61551 100644
--- a/sauvegarde.sh
+++ b/sauvegarde.sh
@@ -9,7 +9,7 @@ set -euo pipefail
 readonly NOM_SCRIPT="${0##*/}"

 destination="$HOME/sauvegardes"
-a_garder=7
+a_garder=10
```

```
  bavard=0

  usage() {
```

`git diff` montre ce qui a changé depuis le dernier enregistrement : la ligne retirée précédée d'un `-`, la ligne ajoutée d'un `+`, avec trois lignes de contexte de part et d'autre. L'option `-a` de `git commit` place d'office dans l'index tous les fichiers déjà suivis :

```
gary@vostro:~/sauvegarde$ git commit -am "Garder dix archives par défaut"
[main 314e7f7] Garder dix archives par défaut
 1 file changed, 1 insertion(+), 1 deletion(-)
gary@vostro:~/sauvegarde$ git lg
* 314e7f7 (HEAD -> main) Garder dix archives par défaut
* c24f1e7 Ajouter le mode d'emploi et la liste des fichiers ignorés
* a04cb95 Premier enregistrement du script de sauvegarde
```

Annuler

Une modification malheureuse, pas encore enregistrée, se défait avec `git restore` :

```
gary@vostro:~/sauvegarde$ echo 'rm -rf "$destination"' >> sauvegarde.sh
gary@vostro:~/sauvegarde$ git diff --stat
 sauvegarde.sh | 1 +
 1 file changed, 1 insertion(+)
gary@vostro:~/sauvegarde$ git restore sauvegarde.sh
gary@vostro:~/sauvegarde$ git status --short
```

Le fichier est revenu à son dernier état enregistré.

Situation	Commande
Défaire les modifications d'un fichier	`git restore fichier`
Retirer un fichier de l'index, sans perdre les modifications	`git restore --staged fichier`
Corriger le message du dernier commit, ou y ajouter un oubli	`git commit --amend`
Annuler un commit déjà partagé	`git revert empreinte` : un nouveau commit qui fait l'inverse
Voir un fichier tel qu'il était	`git show empreinte:fichier`

`git revert` ne réécrit pas l'histoire : il y ajoute une page. C'est la seule façon correcte d'annuler un commit que d'autres ont déjà récupéré.

De bons commits

Un commit raconte une seule chose. « Garder dix archives par défaut » se relit dans six mois. « Modifs diverses » ne se relit jamais. La convention la plus répandue tient en trois règles :

- un titre court, d'une cinquantaine de caractères, qui dit ce que fait le commit ;
- si besoin, une ligne vide, puis un texte qui explique pourquoi ;
- un changement par commit : deux corrections indépendantes font deux commits.

`git add -p` aide à respecter la dernière règle. Il présente chaque morceau modifié et demande s'il faut l'inclure.

27.5 Les branches

Une **branche** est une ligne de travail parallèle. On y essaie une modification sans toucher à la version qui fonctionne. Si l'essai est concluant, on le **fusionne** dans la branche principale. Sinon, on supprime la branche.

Créons une branche pour ajouter une section d'exemples au mode d'emploi :

```
gary@vostro:~/sauvegarde$ git switch -c exemples
Switched to a new branch 'exemples'
gary@vostro:~/sauvegarde$ nano README.md
gary@vostro:~/sauvegarde$ git commit -am "Ajouter des exemples au mode d'emploi"
[exemples 6f20e53] Ajouter des exemples au mode d'emploi
 1 file changed, 5 insertions(+)
gary@vostro:~/sauvegarde$ git switch main
Switched to branch 'main'
gary@vostro:~/sauvegarde$ grep -c Exemples README.md
0
```

De retour sur `main`, le mode d'emploi n'a pas d'exemples : ils n'existent que dans l'autre branche. Fusionnons :

```
gary@vostro:~/sauvegarde$ git merge exemples
Updating 314e7f7..6f20e53
Fast-forward
 README.md | 5 +++++
```

```
 1 file changed, 5 insertions(+)
gary@vostro:~/sauvegarde$ git branch -d exemples
Deleted branch exemples (was 6f20e53).
```

Fast-forward : `main` n'avait pas bougé pendant ce temps, Git a simplement avancé son repère. La branche, devenue inutile, est supprimée.

`git switch` change de branche, `git switch -c` en crée une. Vous rencontrerez aussi `git checkout`, l'ancienne commande, qui fait la même chose et bien d'autres.

Un conflit

Les choses se compliquent quand les deux branches ont modifié la même ligne. Provoquons-le : dans une branche `conservation`, on garde quatorze archives. Pendant ce temps, sur `main`, quelqu'un décide d'en garder cinq.

```
gary@vostro:~/sauvegarde$ git switch -c conservation
gary@vostro:~/sauvegarde$ sed -i 's/^a_garder=10$/a_garder=14/' sauvegarde.sh
gary@vostro:~/sauvegarde$ git commit -qam "Garder deux semaines d'archives"
gary@vostro:~/sauvegarde$ git switch main
gary@vostro:~/sauvegarde$ sed -i 's/^a_garder=10$/a_garder=5/' sauvegarde.sh
gary@vostro:~/sauvegarde$ git commit -qam "Garder cinq archives, le disque est petit"
gary@vostro:~/sauvegarde$ git lg
* e938ae9 (HEAD -> main) Garder cinq archives, le disque est petit
| * 7b1951f (conservation) Garder deux semaines d'archives
|/
* 6f20e53 Ajouter des exemples au mode d'emploi
[…]
gary@vostro:~/sauvegarde$ git merge conservation
Auto-merging sauvegarde.sh
CONFLICT (content): Merge conflict in sauvegarde.sh
Automatic merge failed; fix conflicts and then commit the result.
```

Git ne choisit pas à votre place. Il marque le désaccord dans le fichier :

```
gary@vostro:~/sauvegarde$ git status --short
UU sauvegarde.sh
gary@vostro:~/sauvegarde$ sed -n '11,17p' sauvegarde.sh
destination="$HOME/sauvegardes"
<<<<<<< HEAD
a_garder=5
=======
a_garder=14
>>>>>>> conservation
bavard=0
```

Entre `<<<<<<<` et `=======`, la version de la branche où l'on se trouve. Entre `=======` et `>>>>>>>`, celle de la branche qu'on fusionne. Résoudre le conflit, c'est

éditer le fichier pour n'y laisser que la bonne version, sans les marques. Ici, on donne raison à la branche `conservation`. Ensuite, on signale la résolution par `git add`, et l'on termine la fusion :

```
gary@vostro:~/sauvegarde$ nano sauvegarde.sh
gary@vostro:~/sauvegarde$ sed -n '11,13p' sauvegarde.sh
destination="$HOME/sauvegardes"
a_garder=14
bavard=0
gary@vostro:~/sauvegarde$ git add sauvegarde.sh
gary@vostro:~/sauvegarde$ git commit -m "Fusionner la conservation sur deux semaines"
[main f916374] Fusionner la conservation sur deux semaines
gary@vostro:~/sauvegarde$ git lg
*   f916374 (HEAD -> main) Fusionner la conservation sur deux semaines
|\
| * 7b1951f (conservation) Garder deux semaines d'archives
* | e938ae9 Garder cinq archives, le disque est petit
|/
* 6f20e53 Ajouter des exemples au mode d'emploi
[…]
```

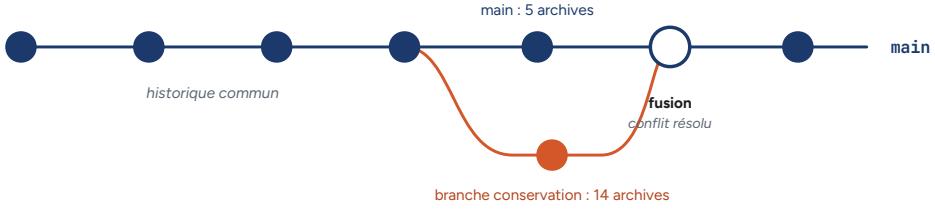


Figure 27.2 — Deux branches qui divergent, puis se rejoignent.

> **ASTUCE · Un conflit sans panique**
>
> Un conflit n'abîme rien. Tant que la fusion n'est pas enregistrée, `git merge --abort` remet tout dans l'état d'avant. Prenez le temps de lire les deux versions : souvent, la bonne réponse est un mélange des deux, et c'est à vous de l'écrire.

Marquer une version

Une **étiquette** donne un nom durable à un commit, en général un numéro de version :

```
gary@vostro:~/sauvegarde$ git tag -a v1.0 -m "Première version publiée"
gary@vostro:~/sauvegarde$ git tag
v1.0
gary@vostro:~/sauvegarde$ git describe
v1.0
```

Après quelques commits de plus, `git describe` situera le travail par rapport à elle : `v1.0-1-g581598a` signifie « un commit après v1.0, au commit 581598a ».

27.6 Faire vérifier ses scripts

Le chapitre 10 recommandait de passer chaque script à `shellcheck`. Git peut s'en charger. Avant chaque enregistrement, il lance le programme `.git/hooks/pre-commit`, s'il existe. Si ce programme échoue, le commit est refusé. Voici un tel **crochet** :

```
#!/bin/bash
#
# pre-commit — refuse l'enregistrement si shellcheck trouve une erreur
# À placer dans .git/hooks/pre-commit, rendu exécutable.

mapfile -t scripts < <(git diff --cached --name-only --diff-filter=ACM -- '*.sh')
[ "${#scripts[@]}" -eq 0 ] && exit 0

if ! shellcheck "${scripts[@]}"; then
    echo "pre-commit : corrigez les remarques de shellcheck avant d'enregistrer."
    exit 1
fi
```

```
gary@vostro:~/sauvegarde$ cp ~/scripts/pre-commit .git/hooks/pre-commit
gary@vostro:~/sauvegarde$ chmod +x .git/hooks/pre-commit
```

Essayons d'enregistrer un script négligé :

```
gary@vostro:~/sauvegarde$ printf '#!/bin/bash\ndossier=$1\ncd $dossier\nls\n' > lister.sh
gary@vostro:~/sauvegarde$ git add lister.sh
gary@vostro:~/sauvegarde$ git commit -m "Ajouter un petit script"

In lister.sh line 3:
cd $dossier
^---------^ SC2164 (warning): Use 'cd ... || exit' or 'cd ... || return' in case cd fails.
   ^------^ SC2086 (info): Double quote to prevent globbing and word splitting.

Did you mean:
cd "$dossier" || exit
[…]
pre-commit : corrigez les remarques de shellcheck avant d'enregistrer.
```

Le commit n'a pas eu lieu. Corrigé, le script passe :

```
gary@vostro:~/sauvegarde$ nano lister.sh
gary@vostro:~/sauvegarde$ cat lister.sh
#!/bin/bash
dossier="$1"
cd "$dossier" || exit 1
ls
gary@vostro:~/sauvegarde$ git add lister.sh
gary@vostro:~/sauvegarde$ git commit -m "Ajouter un petit script"
[main 581598a] Ajouter un petit script
 1 file changed, 4 insertions(+)
 create mode 100644 lister.sh
```

Le dossier `.git/hooks` n'est pas transmis avec le dépôt : chaque copie doit installer son crochet. Pour une vérification que personne ne peut contourner, on la confie à la plateforme d'hébergement, comme on le verra plus loin.

27.7 Partager un dépôt

Un dépôt sur son propre serveur

Nul besoin d'une plateforme pour partager un dépôt : un serveur SSH suffit. On y crée un **dépôt nu**, sans dossier de travail, qui ne sert qu'à recevoir et à distribuer les commits :

```
gary@vostro:~$ ssh serveur git init --bare /srv/git/sauvegarde.git
[…]
Initialized empty Git repository in /srv/git/sauvegarde.git/
```

Le dossier `/srv/git` appartient à `gary` sur le serveur. Git y affiche au passage un long conseil sur le nom de la branche principale : le réglage `init.defaultBranch` n'a été fait que sur le poste. Il est sans conséquence ici, puisque le dépôt recevra la branche `main` du poste.

On déclare ce dépôt comme **distant**, sous le nom habituel `origin`, puis on y envoie la branche et ses étiquettes :

```
gary@vostro:~/sauvegarde$ git remote add origin serveur:/srv/git/sauvegarde.git
gary@vostro:~/sauvegarde$ git remote -v
origin  serveur:/srv/git/sauvegarde.git (fetch)
origin  serveur:/srv/git/sauvegarde.git (push)
gary@vostro:~/sauvegarde$ git push -u origin main --follow-tags
[…]
To serveur:/srv/git/sauvegarde.git
```

```
 * [new branch]      main -> main
 * [new tag]         v1.0 -> v1.0
branch 'main' set up to track 'origin/main'.
```

L'adresse `serveur:/srv/git/sauvegarde.git` reprend le nom défini dans `~/.ssh/config` au chapitre 19. L'option `-u` lie la branche locale à la branche distante : les prochains `git push` et `git pull` sauront où aller sans qu'on le précise.

Cloner, récupérer, envoyer

Sur une autre machine, ou dans un autre dossier, `git clone` récupère le dépôt entier, historique compris :

```
gary@vostro:~$ git clone serveur:/srv/git/sauvegarde.git copie
Cloning into 'copie'...
[…]
gary@vostro:~$ cd copie
gary@vostro:~/copie$ git log --oneline | head -n 3
581598a Ajouter un petit script
f916374 Fusionner la conservation sur deux semaines
e938ae9 Garder cinq archives, le disque est petit
gary@vostro:~/copie$ printf '\nSignaler un problème : gary@maison.lan\n' >> README.md
gary@vostro:~/copie$ git commit -qam "Indiquer où signaler un problème"
gary@vostro:~/copie$ git push
[…]
To serveur:/srv/git/sauvegarde.git
   581598a..d609681  main -> main
```

Pendant ce temps, dans la première copie, un autre commit a été fait. `git fetch` récupère les nouveautés du serveur sans toucher à votre travail, et `git status` fait le point :

```
gary@vostro:~/sauvegarde$ git commit -qam "Commenter la variable bavard"
gary@vostro:~/sauvegarde$ git fetch
[…]
From serveur:/srv/git/sauvegarde
   581598a..d609681  main       -> origin/main
gary@vostro:~/sauvegarde$ git status | head -n 3
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
```

Les deux copies ont chacune un commit que l'autre n'a pas. `git pull` récupère et fusionne. Avec l'option `--rebase`, il replace plutôt votre commit local à la suite de celui du serveur, ce qui garde un historique en ligne droite :

```
gary@vostro:~/sauvegarde$ git pull --rebase
Successfully rebased and updated refs/heads/main.
gary@vostro:~/sauvegarde$ git lg | head -n 4
* 36dfb4d (HEAD -> main) Commenter la variable bavard
* d609681 (origin/main) Indiquer où signaler un problème
* 581598a Ajouter un petit script
*   f916374 (tag: v1.0) Fusionner la conservation sur deux semaines
gary@vostro:~/sauvegarde$ git push
[…]
   d609681..36dfb4d  main -> main
```

Commande	Effet
`git clone adresse`	Copier un dépôt distant
`git fetch`	Récupérer les nouveautés, sans rien fusionner
`git pull`	Récupérer et fusionner
`git pull --rebase`	Récupérer, puis replacer ses commits à la suite
`git push`	Envoyer ses commits

`git rebase` réécrit vos commits : il leur donne une nouvelle empreinte. Ne l'utilisez que sur des commits que vous n'avez pas encore envoyés.

SUR LE TERRAIN · Quand le pare-feu s'en mêle

Pendant la rédaction de ce chapitre, un `git pull` a échoué ainsi :

```
ssh: connect to host 192.168.1.50 port 22: Connection refused
fatal: Could not read from remote repository.
```

Le serveur n'avait rien. La règle `ufw limit` du chapitre 21 refuse une adresse qui ouvre plus de six connexions SSH en trente secondes, et chaque commande Git qui parle au serveur en ouvre une. Le remède est de réutiliser une même connexion, en ajoutant trois lignes au bloc du serveur dans `~/.ssh/config` :

```
Host serveur
    ControlMaster auto
    ControlPath ~/.ssh/cm-%r@%h:%p
    ControlPersist 10m
```

La première commande ouvre la connexion et la laisse ouverte dix minutes. Les suivantes passent par elle. Huit `git fetch` d'affilée n'ont plus rencontré le moindre refus, et l'on gagne au passage le temps de l'établissement de chaque connexion.

SSH ou HTTPS

Un dépôt distant s'adresse de deux façons : en SSH, comme ci-dessus, ou en HTTPS. HTTPS ne demande rien à configurer pour cloner un dépôt public. Pour envoyer, il faut un jeton d'accès créé sur la plateforme, à la place d'un mot de passe. SSH s'appuie sur la clé du chapitre 19 : une fois la clé publique déclarée dans votre compte, tout fonctionne sans rien taper. Pour un usage quotidien, préférez SSH.

27.8 Les plateformes

Un serveur SSH suffit pour partager. Les plateformes ajoutent une interface web, la gestion des comptes et des droits, le suivi des problèmes, et surtout la revue de code.

	GitHub	GitLab	Codeberg
Statut	Entreprise, propriété de Microsoft depuis 2018	Entreprise, logiciel libre en partie	Association à but non lucratif, en Allemagne
Hébergement chez soi	Non	Oui, édition communautaire gratuite	Oui, avec Forgejo, le logiciel qui le fait tourner
Intégration continue	GitHub Actions	GitLab CI	Woodpecker, Forgejo Actions
Pour qui	La plus grande communauté du logiciel libre	Les équipes qui veulent tout héberger elles-mêmes	Ceux qui préfèrent un service associatif

Pour un serveur personnel, Forgejo et Gitea sont légers et s'installent en quelques minutes. GitLab demande plusieurs gigaoctets de mémoire.

Demandes de fusion et intégration continue

Sur une plateforme, on ne pousse pas directement sur `main`. On pousse une branche, puis on ouvre une **demande de fusion**, appelée *pull request* sur GitHub et *merge request* sur GitLab. Les collègues relisent les changements, les commentent ligne par ligne, et la fusion n'a lieu qu'après leur accord.

L'**intégration continue** complète la relecture humaine. À chaque envoi, la plateforme lance une série de vérifications décrite dans un fichier du dépôt : shellcheck, des tests, la fabrication d'un paquet. Le résultat s'affiche dans la demande de fusion.

Notre crochet `pre-commit` tournait sur le poste, et chacun pouvait l'oublier. Placé dans l'intégration continue, le même contrôle s'applique à tous. Voici le fichier `.gitlab-ci.yml` qui ferait ce travail sur GitLab :

```
shellcheck:
  image: koalaman/shellcheck-alpine:stable
  script:
    - shellcheck *.sh
```

27.9 Suivre /etc avec etckeeper

Revenons à l'administration. Le dossier `/etc` est l'endroit qui mérite le plus un historique, et c'est le travail d'**etckeeper**. Il fait de `/etc` un dépôt Git, et enregistre tout seul les changements apportés par `apt`.

```
gary@serveur:~$ sudo apt install etckeeper
[…]
gary@serveur:~$ sudo ls -ld /etc/.git
drwx------ 8 root root 4096 Oct  5 01:12 /etc/.git
gary@serveur:~$ sudo git -C /etc log --oneline
19cd338 (HEAD -> master) Initial commit
gary@serveur:~$ sudo git -C /etc ls-files | wc -l
2530
```

À l'installation, etckeeper a enregistré les 2 530 fichiers de `/etc`. Le dépôt n'est lisible que par `root` : il contient `/etc/shadow`. L'option `-C` de Git lui fait travailler dans un autre dossier que celui où l'on se trouve.

Modifions un réglage de Fail2ban, puis enregistrons la modification avec sa raison :

```
gary@serveur:~$ sudo sed -i 's/^bantime  = 1h/bantime  = 2h/' /etc/fail2ban/jail.local
gary@serveur:~$ sudo git -C /etc status --short
 M fail2ban/jail.local
gary@serveur:~$ sudo git -C /etc diff
diff --git a/fail2ban/jail.local b/fail2ban/jail.local
[…]
 [DEFAULT]
 # Une heure de bannissement après cinq échecs en dix minutes
-bantime  = 1h
+bantime  = 2h
 findtime = 10m
 maxretry = 5
gary@serveur:~$ sudo etckeeper commit "Fail2ban : bannir deux heures"
[master 364acf8] Fail2ban : bannir deux heures
 Author: gary <gary@serveur>
 1 file changed, 1 insertion(+), 1 deletion(-)
```

etckeeper a retrouvé l'auteur réel, `gary`, derrière le `sudo`. Installons maintenant un paquet qui dépose ses fichiers dans `/etc` :

```
gary@serveur:~$ sudo apt install rsnapshot
[…]
gary@serveur:~$ sudo git -C /etc log --oneline
e427ca9 (HEAD -> master) committing changes in /etc made by "apt install -y rsnapshot"
364acf8 Fail2ban : bannir deux heures
19cd338 Initial commit
gary@serveur:~$ sudo git -C /etc show --stat HEAD | tail -n 5
 .etckeeper            |   3 +
 cron.d/rsnapshot      |  11 +++
 logrotate.d/rsnapshot |   6 ++
 rsnapshot.conf        | 250 ++++++++++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 270 insertions(+)
```

`apt` a déclenché un enregistrement, qui dit quelle commande a modifié quels fichiers. Une minuterie quotidienne enregistre aussi ce qu'on aurait oublié.

Le jour où un réglage pose problème, l'historique dit quand il a changé, et `git revert` le défait proprement :

```
gary@serveur:~$ sudo git -C /etc log --oneline -- fail2ban/jail.local
364acf8 Fail2ban : bannir deux heures
19cd338 Initial commit
gary@serveur:~$ sudo git -C /etc revert --no-edit 364acf8
[master 1ab1767] Revert "Fail2ban : bannir deux heures"
[…]
 1 file changed, 1 insertion(+), 1 deletion(-)
gary@serveur:~$ grep '^bantime' /etc/fail2ban/jail.local
bantime  = 1h
gary@serveur:~$ sudo systemctl reload fail2ban
```

Git a signalé au passage qu'il avait deviné l'identité de `root`. Pour des messages propres, donnez-lui un nom une fois pour toutes : `sudo git config --global user.name "root sur serveur"`, et de même pour l'adresse.

> **ATTENTION · Jamais sur une plateforme publique**
>
> Le dépôt de `/etc` contient les empreintes des mots de passe, les clés privées SSH de la machine et des certificats. Ne l'envoyez jamais vers une plateforme publique. Si vous voulez une copie hors de la machine, poussez-la vers un dépôt nu sur un serveur à vous, accessible à vous seul.

En résumé

- `git config --global` règle l'identité et l'éditeur, une fois pour toutes.
- Le cycle : modifier, `git status`, `git diff`, `git add`, `git commit -m "…"`.
- `.gitignore` liste ce qui ne doit pas être suivi. Un secret enregistré reste dans l'historique.
- `git restore` défait une modification, `git revert` annule un commit déjà partagé.
- `git switch -c` crée une branche, `git merge` la fusionne. Un conflit se résout en éditant le fichier, puis `git add` et `git commit`.
- `git tag -a` marque une version.
- Un crochet `pre-commit` vérifie chaque enregistrement. L'intégration continue fait de même pour toute l'équipe.
- Un dépôt nu sur un serveur SSH suffit pour partager : `git clone`, `git fetch`, `git pull --rebase`, `git push`.
- etckeeper donne un historique à `/etc`, enrichi à chaque passage d'`apt`.

Atelier — Publier un premier projet

Objectif : publier le script de sauvegarde du chapitre 10 sur une plateforme, avec un mode d'emploi, une liste d'exclusions et une première version étiquetée. Comptez quarante-cinq minutes.

Le dépôt local

1. Configurez votre identité, votre éditeur et la branche par défaut `main`.
2. Créez un dossier pour le projet, copiez-y `sauvegarde.sh`, et faites-en un dépôt.
3. Écrivez un `README.md` qui dit ce que fait le script, comment l'appeler, et sous quelle licence vous le publiez.
4. Écrivez un `.gitignore` qui exclut les archives produites et les fichiers temporaires.
5. Installez le crochet `pre-commit`. Enregistrez le tout en deux ou trois commits bien nommés.

Une évolution

6. Dans une branche, ajoutez une option `-q` qui rend le script silencieux. Vérifiez que le crochet accepte vos changements.
7. Fusionnez la branche dans `main`, supprimez-la, et étiquetez le résultat `v1.0`.

La publication

8. Créez un compte sur GitLab, GitHub ou Codeberg, et déclarez-y votre clé SSH publique.
9. Créez un projet vide, sans README : le vôtre est prêt.
10. Ajoutez l'adresse SSH du projet comme dépôt distant `origin`, et envoyez `main` avec ses étiquettes.
11. Sur la page du projet, vérifiez que le mode d'emploi s'affiche et que la version `v1.0` apparaît.

Vérifier

- `git status` dit *Your branch is up to date with 'origin/main'*.
- Un `git clone` dans un autre dossier redonne le projet complet, et `git describe` y affiche `v1.0`.
- Aucun fichier listé dans `.gitignore` ne figure sur la plateforme.

Exercices

Exercice 1 — Où en est-on ?

```
 M sauvegarde.sh
A  lister.sh
?? notes.txt
UU README.md
```

Pour chaque ligne de `git status --short`, dites l'état du fichier et la commande qui le ferait avancer.

Exercice 2 — Revenir en arrière

Dans le dépôt de l'atelier :

1. Modifiez `sauvegarde.sh`, puis annulez la modification sans l'enregistrer.

2. Modifiez-le, placez-le dans l'index, puis retirez-le de l'index en gardant la modification.
3. Enregistrez un commit avec une faute dans le message, puis corrigez le message.
4. Annulez proprement le commit précédent, comme s'il avait déjà été partagé.

Exercice 3 — Un conflit

Créez deux branches qui modifient chacune la même ligne du mode d'emploi, de façon différente. Fusionnez-les. Résolvez le conflit en gardant un mélange des deux versions. Vérifiez avec `git lg` que l'historique montre la fusion.

Exercice 4 — Le dépôt nu

Créez sur votre serveur un dépôt nu pour vos fichiers de configuration personnels : `.bashrc`, `.vimrc` ou `.nanorc`, `.ssh/config`. Clonez-le sur une seconde machine. Pourquoi ne faut-il surtout pas y mettre vos clés privées ?

Exercice 5 — etckeeper

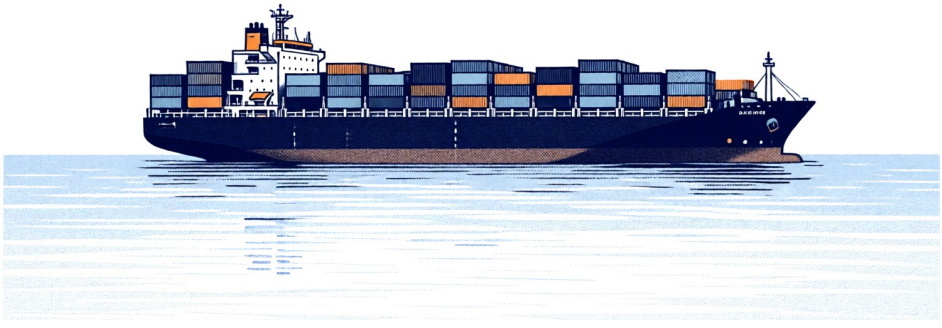
1. Installez etckeeper sur votre serveur.
2. Modifiez un réglage de SSH, enregistrez-le avec un message clair.
3. Installez un paquet qui dépose un fichier dans `/etc`, et retrouvez le commit correspondant.
4. Affichez le contenu de `ssh/sshd_config.d/` tel qu'il était à l'installation d'etckeeper.

Pour aller plus loin

Sur la machine. `git help` suivi du nom d'une commande ouvre sa documentation complète. `git help gitignore` explique la syntaxe des exclusions, `git help githooks` liste tous les crochets possibles. `man etckeeper` décrit ses réglages, dans `/etc/etckeeper/etckeeper.conf`.

Paquets. `tig` affiche l'historique dans le terminal, en navigable. `git-gui` et `gitk` en donnent une version graphique. `pre-commit`, présent dans les dépôts, gère des crochets partagés entre plusieurs dépôts.

En ligne. Le livre *Pro Git*, de Scott Chacon et Ben Straub, est publié gratuitement sur `git-scm.com/book`, et traduit en français. Le site `learngitbranching.js.org` fait manipuler les branches en dessinant l'historique à chaque commande.



CHAPITRE 28

Les conteneurs avec Docker

Installer une application, c'est souvent installer ses bibliothèques, sa base de données, ses réglages, et espérer que rien n'entre en conflit avec le reste du serveur. Un conteneur emporte tout cela avec lui. On le lance en une commande, on le supprime sans laisser de trace, et il tourne de la même façon sur votre poste et sur le serveur.

DANS CE CHAPITRE

- Distinguer un conteneur d'une machine virtuelle.
- Installer Docker Engine depuis le dépôt officiel.
- Lancer, observer, arrêter et supprimer des conteneurs.
- Garder les données dans des volumes.
- Faire communiquer des conteneurs sur un réseau.
- Écrire un Dockerfile et construire une image.
- Décrire une application complète avec Docker Compose.
- Publier un service sans trouer le pare-feu.

> **AVANT DE COMMENCER**
>
> Les chapitres 12, 21, 22 et 23. Le serveur fait tourner Nginx et Certbot comme au chapitre 23, avec l'autorité de certification de la maison. L'application `tableau` du chapitre 23 sert d'exemple.

28.1 Conteneur ou machine virtuelle

Une **machine virtuelle** imite un ordinateur complet. Elle a son propre noyau, démarre son propre système, réserve sa mémoire. Un **conteneur** n'imite rien : c'est un groupe de processus ordinaires de l'hôte, que le noyau isole du reste. Ils ont leur propre arborescence de fichiers, leur propre réseau, leurs propres numéros de processus, mais ils partagent le noyau de la machine.

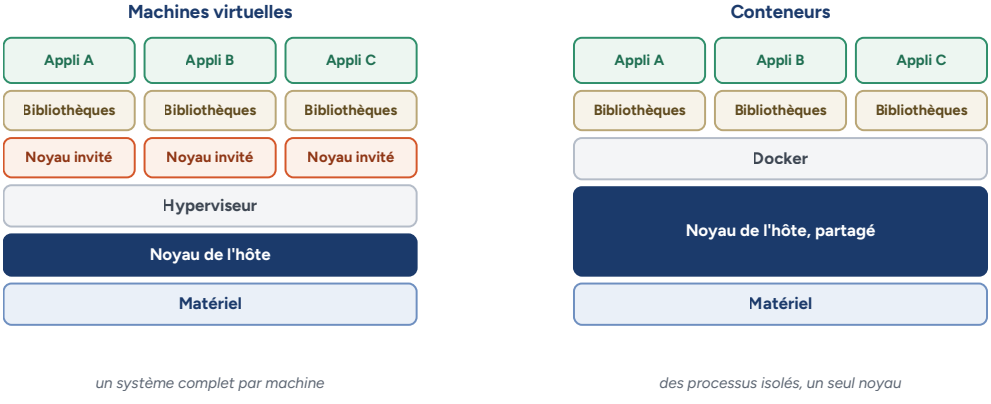


Figure 28.1 — Une machine virtuelle embarque un système entier ; un conteneur partage le noyau de l'hôte.

	Machine virtuelle	Conteneur
Démarrage	Des dizaines de secondes	Une fraction de seconde
Taille	Des gigaoctets	Quelques dizaines de mégaoctets
Isolation	Forte : un noyau par machine	Bonne, mais un seul noyau pour tous
Système invité	N'importe lequel	Linux, puisque le noyau est celui de l'hôte

Les briques sont celles du chapitre 12 : les **espaces de noms** (*namespaces*) cachent aux processus ce qui ne les concerne pas, les **groupes de contrôle** limitent leur

consommation. Le chapitre 22 ajoute AppArmor. Docker ne fait qu'assembler ces mécanismes et les rendre simples à utiliser.

> **REPÈRE HISTORIQUE · Cinq minutes à PyCon**
>
> En mars 2013, Solomon Hykes, cofondateur de la société française dotCloud, présente en cinq minutes à la conférence PyCon, en Californie, l'outil interne avec lequel sa société fait tourner les applications de ses clients. Il s'appelle Docker, et son code est publié le même mois. La société prend son nom quelques mois plus tard. Les conteneurs existaient avant lui, mais personne ne les avait rendus aussi simples.

28.2 Le vocabulaire

Terme	Sens
Image	Un modèle en lecture seule : un système de fichiers minimal et la commande à lancer
Conteneur	Une image en cours d'exécution, avec sa couche de fichiers modifiables
Registre	Un dépôt d'images. Docker Hub est celui par défaut
Volume	Un espace de stockage qui survit au conteneur
Dockerfile	La recette qui fabrique une image

Une image se désigne par un nom et une **étiquette** : `nginx:alpine` est l'image `nginx` dans sa variante construite sur Alpine, une distribution minuscule. Sans étiquette, Docker prend `latest`.

28.3 Installer Docker Engine

Ubuntu propose un paquet `docker.io`, mais il suit les versions avec retard. L'éditeur tient son propre dépôt, que l'on ajoute comme au chapitre 7 :

```
gary@serveur:~$ sudo install -m 0755 -d /etc/apt/keyrings
gary@serveur:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
    -o /etc/apt/keyrings/docker.asc
gary@serveur:~$ echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo $VERSION_CODENAME) stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
gary@serveur:~$ cat /etc/apt/sources.list.d/docker.list | cut -c1-80
```

```
deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.
gary@serveur:~$ sudo apt update
[…]
gary@serveur:~$ sudo apt install docker-ce docker-ce-cli containerd.io \
    docker-buildx-plugin docker-compose-plugin
[…]
gary@serveur:~$ docker --version
Docker version 29.8.2, build 7fc2dff
gary@serveur:~$ docker compose version
Docker Compose version v5.6.0
```

La ligne du dépôt se fabrique toute seule : l'architecture et le nom de code de la version, `noble`, sont lus sur la machine. Deux services démarrent, `docker` et `containerd`. Le premier essai consacré est l'image `hello-world` :

```
gary@serveur:~$ docker ps
permission denied while trying to connect to the docker API at unix:///var/run/docker.sock
gary@serveur:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
[…]
Hello from Docker!
This message shows that your installation appears to be working correctly.
[…]
```

Docker a cherché l'image, ne l'a pas trouvée, l'a téléchargée depuis Docker Hub, puis l'a lancée.

Le groupe docker

La commande `docker` parle au service par la prise `/var/run/docker.sock`, réservée à `root` et au groupe `docker`. Pour se passer de `sudo`, on rejoint le groupe, puis on ouvre une nouvelle session :

```
gary@serveur:~$ sudo usermod -aG docker gary
[…]
gary@serveur:~$ id -nG
gary adm cdrom sudo dip plugdev docker acces-ssh partage
```

> **DANGER · Le groupe docker vaut root**
>
> Qui peut lancer un conteneur peut monter `/` de l'hôte dedans et y écrire en `root` : `docker run -v /:/hote …` suffit. Appartenir au groupe `docker`, c'est donc posséder la machine. N'y mettez que des comptes à qui vous confieriez `sudo`.

28.4 Premiers pas

Lançons un serveur web :

```
gary@serveur:~$ docker run -d --name web -p 8080:80 nginx:alpine
[…]
Status: Downloaded newer image for nginx:alpine
ca5238994cba2a7fbe452db47073dd1dd59a72cc68c4c5d424e085da322db20e
gary@serveur:~$ docker ps --format 'table {{.Names}}\t{{.Image}}\t{{.Status}}\t{{.Ports}}'
NAMES     IMAGE          STATUS                  PORTS
web       nginx:alpine   Up Less than a second   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp
gary@serveur:~$ curl -sI localhost:8080 | head -n 2
HTTP/1.1 200 OK
Server: nginx/1.31.6
```

Option	Rôle
`-d`	Lancer en arrière-plan
`--name web`	Donner un nom, plutôt que le nom tiré au hasard

`docker ps` liste les conteneurs en marche. Sa sortie complète compte sept colonnes ; l'option `--format` n'en garde ici que quatre. | `-p 8080:80` | Relier le port 8080 de l'hôte au port 80 du conteneur |

Ce Nginx n'a rien à voir avec celui du chapitre 23 : c'est une autre version, sur un autre système. Entrons voir :

```
gary@serveur:~$ docker exec web cat /etc/os-release | head -n 2
NAME="Alpine Linux"
ID=alpine
gary@serveur:~$ docker exec web ps
PID   USER     TIME  COMMAND
    1 root      0:00 nginx: master process nginx -g daemon off;
   30 nginx     0:00 nginx: worker process
   31 nginx     0:00 nginx: worker process
   44 root      0:00 ps
```

`docker exec` lance une commande dans un conteneur en marche ; `docker exec -it web sh` y ouvrirait un shell. À l'intérieur, Nginx est le processus 1, et Alpine ne voit que lui. Vu de l'hôte, c'est un processus ordinaire, avec un autre numéro, et le profil AppArmor que Docker applique à tous les conteneurs :

```
gary@serveur:~$ ps -o user,pid,cmd -C nginx | grep 'master process nginx -g'
root      211602 nginx: master process nginx -g daemon off;
gary@serveur:~$ sudo cat /proc/$(docker inspect -f '{{.State.Pid}}' web)/attr/current
docker-default (enforce)
```

Les journaux sont ce que le programme écrit sur sa sortie, et `docker stats` montre sa consommation :

```
gary@serveur:~$ docker logs web | tail -n 2
/docker-entrypoint.sh: Configuration complete; ready for start up
172.17.0.1 - - [05/Oct/2026:01:19:39 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/8.5.0" "-"
gary@serveur:~$ docker stats --no-stream \
    --format 'table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}'
NAME      CPU %     MEM USAGE / LIMIT
web       0.00%     3.391MiB / 1.423GiB
```

Trois mégaoctets et demi pour un serveur web. Arrêtons-le, puis supprimons-le :

```
gary@serveur:~$ docker stop web
web
gary@serveur:~$ docker ps -a --format 'table {{.Names}}\t{{.Image}}\t{{.Status}}'
NAMES     IMAGE          STATUS
web       nginx:alpine   Exited (0) Less than a second ago
gary@serveur:~$ docker rm -f web
web
```

Un conteneur arrêté existe encore : `docker ps -a`, qui liste aussi les conteneurs arrêtés, le montre, et `docker start` le relancerait. `docker rm` le supprime, avec tout ce qui avait été écrit dedans. `-f` arrête et supprime d'un coup.

Commande	Effet
`docker run`	Créer et lancer un conteneur
`docker ps`, `docker ps -a`	Les conteneurs en marche, ou tous
`docker logs -f nom`	Suivre les journaux
`docker exec -it nom sh`	Ouvrir un shell à l'intérieur
`docker stop`, `docker start`	Arrêter, relancer
`docker rm`, `docker rmi`	Supprimer un conteneur, une image
`docker images`	Les images présentes
`docker system df`	La place occupée

28.5 Docker et le pare-feu

Le pare-feu du chapitre 21 n'autorise que SSH, HTTP et HTTPS. Publions deux conteneurs, l'un sur toutes les adresses, l'autre sur `127.0.0.1` seulement :

```
gary@serveur:~$ docker run -d --name ouvert -p 8081:80 nginx:alpine
[…]
gary@serveur:~$ docker run -d --name local -p 127.0.0.1:8082:80 nginx:alpine
[…]
gary@serveur:~$ sudo ufw status | grep -E '8081|8082' ; echo "aucune règle"
aucune règle
```

Aucune règle n'ouvre ces ports. Et pourtant, depuis le poste :

```
gary@vostro:~$ curl -s --max-time 5 http://192.168.1.50:8081 | grep -o '<title>.*</title>'
<title>Welcome to nginx!</title>
gary@vostro:~$ curl -s --max-time 5 http://192.168.1.50:8082; echo "code $?"
code 28
```

SUR LE TERRAIN · Docker passe à côté d'UFW

Pour publier un port, Docker écrit ses propres règles dans la table de traduction d'adresses du noyau :

```
gary@serveur:~$ sudo iptables -t nat -L DOCKER -n | grep -E 'dpt:808[12]'
DNAT       6    --  0.0.0.0/0            0.0.0.0/0            tcp dpt:8081 to:
172.17.0.2:80
DNAT       6    --  0.0.0.0/0            127.0.0.1            tcp dpt:8082 to:
172.17.0.3:80
```

Le paquet destiné au port 8081 est détourné vers le conteneur avant d'atteindre les règles d'UFW, qui ne le voient jamais. Un port publié avec `-p 8081:80` est donc ouvert au monde, quoi que dise `ufw status`. La règle à suivre : publier sur `127.0.0.1`, et laisser le relais Nginx du chapitre 23 faire face au réseau. Le port 8082 ne répond qu'à la machine elle-même.

28.6 Les volumes

Ce qu'un conteneur écrit disparaît avec lui. Une base de données, des fichiers envoyés par les utilisateurs doivent vivre ailleurs : dans un **volume**, que Docker range dans `/var/lib/docker/volumes`.

Uptime Kuma, l'outil de surveillance cité au chapitre 26, garde sa configuration dans `/app/data`. Confions ce dossier à un volume :

```
gary@serveur:~$ docker volume create donnees-kuma
donnees-kuma
gary@serveur:~$ docker run -d --name kuma -p 127.0.0.1:3001:3001 \
    -v donnees-kuma:/app/data --restart unless-stopped louislam/uptime-kuma:1
[…]
gary@serveur:~$ docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}'
```

```
NAMES     STATUS                    PORTS
kuma      Up 44 seconds (healthy)   127.0.0.1:3001->3001/tcp
gary@serveur:~$ sudo ls /var/lib/docker/volumes/donnees-kuma/_data
docker-tls  kuma.db  kuma.db-shm  kuma.db-wal  screenshots  upload
```

Supprimons le conteneur : le volume reste, et un nouveau conteneur retrouve tout.

```
gary@serveur:~$ docker rm -f kuma
kuma
gary@serveur:~$ docker volume ls
DRIVER    VOLUME NAME
local     donnees-kuma
[…]
gary@serveur:~$ docker run -d --name kuma -p 127.0.0.1:3001:3001 \
    -v donnees-kuma:/app/data --restart unless-stopped louislam/uptime-kuma:1
[…]
gary@serveur:~$ sudo ls /var/lib/docker/volumes/donnees-kuma/_data
docker-tls  kuma.db  kuma.db-shm  kuma.db-wal  screenshots  upload
```

`--restart unless-stopped` relance le conteneur après une panne ou un redémarrage du serveur, sauf si on l'a arrêté soi-même.

On peut aussi monter un dossier de l'hôte, ce qu'on appelle un **montage lié**. Le suffixe `:ro` l'offre en lecture seule :

```
gary@serveur:~$ mkdir -p ~/site
gary@serveur:~$ echo '<h1>Servi depuis le disque du serveur</h1>' > ~/site/index.html
gary@serveur:~$ docker run -d --name statique -p 127.0.0.1:8083:80 \
    -v ~/site:/usr/share/nginx/html:ro nginx:alpine
[…]
gary@serveur:~$ curl -s localhost:8083
<h1>Servi depuis le disque du serveur</h1>
gary@serveur:~$ echo '<h1>Modifié à chaud</h1>' > ~/site/index.html
gary@serveur:~$ curl -s localhost:8083
<h1>Modifié à chaud</h1>
gary@serveur:~$ docker exec statique sh -c 'echo essai > /usr/share/nginx/html/x.html'
sh: can't create /usr/share/nginx/html/x.html: Read-only file system
```

	Volume	Montage lié
Écriture	`-v nom:/chemin`	`-v /dossier/hôte:/chemin`
Rangement	Géré par Docker	Où vous voulez
Usage	Les données d'une application	Des fichiers que vous éditez, une configuration

28.7 Les réseaux

Docker crée trois réseaux à l'installation :

```
gary@serveur:~$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
4a7ccaa0e7d9   bridge    bridge    local
ffffbe3aa681   host      host      local
bd9cde101965   none      null      local
```

Réseau	Effet
`bridge`	Le réseau par défaut : chaque conteneur a son adresse privée, en 172.17.0.0/16
`host`	Le conteneur utilise directement le réseau de l'hôte, sans isolation
`none`	Aucun réseau

Pour qu'une application parle à sa base de données, on crée un réseau à elle. Sur un tel réseau, chaque conteneur est joignable par son nom, ce que le réseau par défaut ne permet pas :

```
gary@serveur:~$ docker network create appli
[…]
gary@serveur:~$ docker run -d --name base --network appli \
    -e POSTGRES_PASSWORD=essai postgres:16-alpine
[…]
gary@serveur:~$ docker run --rm --network appli alpine ping -c 1 -W 2 base | head -n 2
PING base (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.044 ms
gary@serveur:~$ docker run --rm alpine ping -c 1 -W 2 base
ping: bad address 'base'
```

Le premier conteneur, sur le réseau `appli`, trouve `base`. Le second, sur le réseau par défaut, ne le connaît pas. `--rm` supprime le conteneur dès qu'il a fini. La base n'a publié aucun port : elle n'est joignable que par les conteneurs de son réseau, ce qui est exactement ce qu'on veut.

28.8 Construire une image

Une image se fabrique à partir d'un **Dockerfile**, une suite d'instructions. Mettons en boîte l'application `tableau` du chapitre 23 :

```
FROM python:3.12-slim

RUN pip install --no-cache-dir gunicorn==23.0.0
```

```
WORKDIR /app
COPY tableau.py .

USER nobody
EXPOSE 8000

HEALTHCHECK --interval=30s --timeout=3s \
    CMD python3 -c "import urllib.request; urllib.request.urlopen('http://127.0.0.1:8000/')"

CMD ["gunicorn", "--bind", "0.0.0.0:8000", "--workers", "2", "tableau:application"]
```

Instruction	Rôle
`FROM`	L'image de départ : ici Python 3.12 sur une Debian réduite
`RUN`	Une commande exécutée pendant la construction
`WORKDIR`	Le dossier de travail pour la suite
`COPY`	Copier un fichier du dossier de construction dans l'image
`USER`	Le compte sous lequel tournera le programme
`EXPOSE`	Documente le port écouté, sans rien publier
`HEALTHCHECK`	Une commande qui dit si le service va bien
`CMD`	La commande lancée au démarrage du conteneur

Un fichier `.dockerignore` liste ce qui ne doit pas être envoyé à la construction, à la manière du `.gitignore` du chapitre 27. Construisons :

```
gary@serveur:~/tableau$ ls -A
.dockerignore  Dockerfile  tableau.py
gary@serveur:~/tableau$ docker build -t tableau:1.0 .
[…]
#5 [1/4] FROM docker.io/library/python:3.12-slim@sha256:02108f5d322d…
#6 [2/4] RUN pip install --no-cache-dir gunicorn==23.0.0
#7 [3/4] WORKDIR /app
#8 [4/4] COPY tableau.py .
#9 naming to docker.io/library/tableau:1.0 done
gary@serveur:~/tableau$ docker images tableau
IMAGE         ID             DISK USAGE   CONTENT SIZE   EXTRA
tableau:1.0   55e7a708efe4        204MB         50.2MB
```

Chaque instruction produit une **couche**. Docker les garde en cache : si l'on modifie `tableau.py`, seule la dernière couche est reconstruite. D'où l'ordre du Dockerfile, du plus stable au plus changeant.

L'application lit les relevés du service `veilleur`. On lui prête leur dossier en lecture seule :

```
gary@serveur:~$ docker run -d --name tableau -p 127.0.0.1:8001:8000 \
    -v /var/lib/veilleur:/var/lib/veilleur:ro tableau:1.0
[…]
gary@serveur:~$ docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}'
NAMES     STATUS                    PORTS
tableau   Up 40 seconds (healthy)   127.0.0.1:8001->8000/tcp
gary@serveur:~$ curl -s localhost:8001 | head -n 3
Charge du serveur, cinq derniers relevés
2026-10-05 01:25:01  0.36
2026-10-05 01:25:11  0.38
gary@serveur:~$ docker exec tableau id
uid=65534(nobody) gid=65534(nogroup) groups=65534(nogroup)
gary@serveur:~$ docker exec tableau touch /var/lib/veilleur/essai
touch: cannot touch '/var/lib/veilleur/essai': Read-only file system
```

Le contrôle de santé est passé : `healthy`. Le programme tourne sous `nobody`, et ne peut rien écrire dans les relevés.

> **ASTUCE · « address already in use »**
>
> Le premier essai a été lancé avec `-p 127.0.0.1:8000:8000`, et Docker a refusé : `failed to bind host port 127.0.0.1:8000/tcp: address already in use`. Le service `tableau` du chapitre 23 occupait déjà ce port. Avant de publier, `sudo ss -ltnp | grep :8000` dit qui l'utilise.

Pour partager l'image, on la pousse vers un registre : `docker login`, `docker tag`, puis `docker push`. Docker Hub, GitHub et GitLab en offrent un, et le paquet `docker-registry` permet d'héberger le sien.

28.9 Docker Compose

Une application réelle compte plusieurs conteneurs, avec leurs volumes, leur réseau et leurs réglages. Les décrire par des `docker run` à rallonge devient vite illisible. **Docker Compose** les décrit dans un fichier, `compose.yaml`, et les lance d'une commande. Voici WordPress avec sa base MariaDB :

```
services:
  base:
    image: mariadb:11.4
    restart: unless-stopped
    environment:
```

```
      MARIADB_DATABASE: wordpress
      MARIADB_USER: wordpress
      MARIADB_PASSWORD: ${MDP_BASE}
      MARIADB_RANDOM_ROOT_PASSWORD: "1"
    volumes:
      - base:/var/lib/mysql
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--innodb_initialized"]
      interval: 10s
      timeout: 5s
      retries: 5

  wordpress:
    image: wordpress:6-apache
    restart: unless-stopped
    depends_on:
      base:
        condition: service_healthy
    environment:
      WORDPRESS_DB_HOST: base
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: ${MDP_BASE}
      WORDPRESS_DB_NAME: wordpress
    ports:
      - "127.0.0.1:8080:80"
    volumes:
      - site:/var/www/html

volumes:
  base:
  site:
```

Clé	Rôle
`services`	Les conteneurs. Chaque nom de service devient un nom sur le réseau
`image`	L'image, avec une étiquette précise plutôt que `latest`
`environment`	Les variables passées au conteneur
`volumes`	Les volumes montés, déclarés en bas du fichier
`healthcheck`	Le contrôle de santé
`depends_on`	L'ordre de démarrage. Avec `service_healthy`, WordPress attend que la base soit prête
`restart`	La politique de redémarrage

`${MDP_BASE}` est lu dans le fichier `.env`, à côté de `compose.yaml`. Le mot de passe n'apparaît ainsi jamais dans le fichier qu'on partage ou qu'on place dans Git :

```
gary@serveur:~/wordpress$ printf 'MDP_BASE=%s\n' "$(openssl rand -base64 18)" > .env
gary@serveur:~/wordpress$ chmod 600 .env
gary@serveur:~/wordpress$ docker compose config --services
base
wordpress
gary@serveur:~/wordpress$ docker compose up -d
 Image mariadb:11.4 Pulled
 Image wordpress:6-apache Pulled
 Network wordpress_default Created
 Volume wordpress_base Created
 Volume wordpress_site Created
 Container wordpress-base-1 Started
 Container wordpress-base-1 Healthy
 Container wordpress-wordpress-1 Started
```

Compose a créé un réseau propre à l'application, les deux volumes, puis a lancé la base, attendu qu'elle soit saine, et seulement alors lancé WordPress.

```
gary@serveur:~/wordpress$ docker compose ps \
    --format 'table {{.Service}}\t{{.Status}}\t{{.Ports}}'
SERVICE     STATUS                    PORTS
base        Up 41 seconds (healthy)   3306/tcp
wordpress   Up 30 seconds             127.0.0.1:8080->80/tcp
gary@serveur:~/wordpress$ curl -sI localhost:8080 | grep -E '^HTTP|^Location'
HTTP/1.1 302 Found
Location: http://localhost:8080/wp-admin/install.php
```

La base n'est publiée nulle part : seul WordPress la voit. WordPress répond sur `127.0.0.1:8080` et propose son installation.

Commande	Effet
`docker compose up -d`	Créer et lancer, ou mettre à jour ce qui a changé
`docker compose ps`	L'état des services
`docker compose logs -f service`	Les journaux
`docker compose exec service commande`	Une commande dans un service
`docker compose pull`	Télécharger les nouvelles versions des images
`docker compose down`	Arrêter et supprimer les conteneurs et le réseau, en gardant les volumes

Derrière Nginx

Il reste à exposer le site, en HTTPS. C'est le travail du Nginx de l'hôte, exactement comme pour l'application du chapitre 23 :

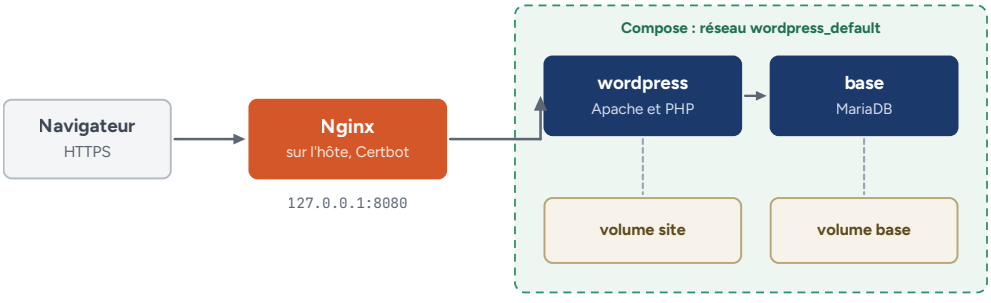


Figure 28.2 — WordPress derrière le Nginx de l'hôte : seul Nginx fait face au réseau.

```
server {
    listen 80;
    listen [::]:80;
    server_name wp.maison.lan;

    client_max_body_size 64m;

    location / {
        proxy_pass http://127.0.0.1:8080;
        include proxy_params;
    }
}
```

`client_max_body_size` autorise l'envoi d'images et d'extensions jusqu'à 64 Mo. Puis Certbot, comme au chapitre 23 :

```
gary@serveur:~$ sudo ln -s /etc/nginx/sites-available/wp /etc/nginx/sites-enabled/
gary@serveur:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
gary@serveur:~$ sudo systemctl reload nginx
gary@serveur:~$ sudo certbot --nginx -d wp.maison.lan
[…]
Successfully deployed certificate for wp.maison.lan to /etc/nginx/sites-enabled/wp
Congratulations! You have successfully enabled HTTPS on https://wp.maison.lan
```

Depuis le poste :

```
gary@vostro:~$ curl -sI http://wp.maison.lan | grep -E '^HTTP|^Location'
HTTP/1.1 301 Moved Permanently
Location: https://wp.maison.lan/
gary@vostro:~$ curl -s https://wp.maison.lan/wp-admin/install.php | \
    grep -o '<title>[^<]*</title>'
<title>WordPress &rsaquo; Installation</title>
```

Il ne reste qu'à terminer l'installation dans le navigateur. `proxy_params`, vu au chapitre 23, transmet l'en-tête `X-Forwarded-Proto`, grâce auquel WordPress sait qu'il est servi en HTTPS.

Sauvegarder et mettre à jour

Les données vivent dans les volumes. La base se sauvegarde par un export, comme au chapitre 24, lancé dans son conteneur :

```
gary@serveur:~/wordpress$ docker compose exec base sh -c \
    'mariadb-dump -u wordpress -p"$MARIADB_PASSWORD" wordpress' > wordpress.sql
gary@serveur:~/wordpress$ ls -lh wordpress.sql
-rw-rw-r-- 1 gary gary 100K Oct  5 01:29 wordpress.sql
gary@serveur:~/wordpress$ grep -c '^CREATE TABLE' wordpress.sql
12
```

Les guillemets simples comptent : `$MARIADB_PASSWORD` est lu dans le conteneur, où Compose l'a défini, et non sur l'hôte. Le volume `site`, qui contient les extensions et les images envoyées, se sauvegarde comme un dossier : ajoutez `/var/lib/docker/volumes/wordpress_site/_data` à la sauvegarde restic du chapitre 16.

Mettre à jour, c'est télécharger les nouvelles images et recréer les conteneurs qui en ont besoin :

```
gary@serveur:~/wordpress$ docker compose pull
 Image mariadb:11.4 Pulled
 Image wordpress:6-apache Pulled
gary@serveur:~/wordpress$ docker compose up -d
 Container wordpress-wordpress-1 Running
 Container wordpress-base-1 Running
```

Rien n'avait changé ce jour-là : les deux conteneurs sont restés en place. Après une nouvelle version, Compose les aurait recréés, en gardant les volumes. Les anciennes images restent sur le disque jusqu'à `docker image prune`.

> **ATTENTION · Les images ne se mettent pas à jour toutes seules**
>
> `unattended-upgrades`, vu au chapitre 20, corrige les paquets de l'hôte, pas le contenu des images. Une image vieille de six mois contient les failles de six mois. Prévoyez un `docker compose pull` régulier, et choisissez des étiquettes qui suivent les correctifs : `mariadb:11.4` reçoit les corrections de la 11.4, sans changer de version majeure.

28.10 Traefik, le relais qui lit les étiquettes

Le chapitre 23 a présenté Traefik, ce relais qui découvre seul les applications à servir. C'est avec Docker qu'il prend tout son sens. Il interroge le service Docker, et lit sur chaque conteneur des **étiquettes** qui lui disent quoi faire :

```
services:
  traefik:
    image: traefik:v3.7
    command:
      - --providers.docker=true
      - --providers.docker.exposedbydefault=false
      - --entrypoints.web.address=:80
    ports:
      - "127.0.0.1:8090:80"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro

  bonjour:
    image: traefik/whoami
    labels:
      - traefik.enable=true
      - traefik.http.routers.bonjour.rule=Host(`bonjour.maison.lan`)
```

Le service `bonjour` n'a aucun port publié. Son étiquette dit à Traefik : envoie-moi les demandes adressées à `bonjour.maison.lan`.

```
gary@serveur:~/traefik$ docker compose up -d
[…]
gary@serveur:~/traefik$ curl -s -H 'Host: bonjour.maison.lan' localhost:8090 | head -n 2
Hostname: 4c76a2ca364f
IP: 127.0.0.1
gary@serveur:~/traefik$ curl -sI -H 'Host: autre.maison.lan' localhost:8090 | head -n 1
HTTP/1.1 404 Not Found
```

Ajouter une application, c'est lancer un conteneur avec les bonnes étiquettes : Traefik la découvre, sans qu'on touche à sa configuration ni qu'on le recharge. Il sait aus-

si obtenir lui-même ses certificats auprès de Let's Encrypt. C'est pourquoi on le trouve devant tant de serveurs qui hébergent des dizaines de conteneurs.

En échange, Traefik a besoin de la prise de Docker, `/var/run/docker.sock`. Le `:ro` du montage n'y change rien : la prise donne accès à toute l'API de Docker, donc à la machine. Un Traefik compromis est un serveur compromis. Pour quelques applications, le Nginx de l'hôte reste plus simple et plus sûr.

28.11 De bonnes habitudes

Habitude	Pourquoi
Publier sur `127.0.0.1`	Docker passe à côté d'UFW
Fixer les étiquettes des images	`latest` change sans prévenir
Préférer les images officielles, et les petites	Moins de logiciels, moins de failles
Faire tourner le programme sous un compte ordinaire	`USER` dans le Dockerfile
Monter en lecture seule ce qui n'a pas à être écrit	Le suffixe `:ro`
Donner un contrôle de santé à chaque service	Compose et Traefik s'en servent
Garder les secrets dans `.env`, hors de Git	Un secret enregistré est publié, chapitre 27
Sauvegarder les volumes et exporter les bases	Le conteneur se recrée, les données non

`docker system df` dit la place que tout cela occupe :

```
gary@serveur:~$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          8         3         2.099GB   341.2MB (16%)
Containers      3         3         69.63kB   0B (0%)
Local Volumes   2         2         244.8MB   0B (0%)
Build Cache     10        0         204.4MB   24.58kB
```

Deux gigaoctets d'images après un après-midi d'essais. `docker system prune` supprime les conteneurs arrêtés, les réseaux inutilisés et les images orphelines. Il ne touche pas aux volumes sans l'option `--volumes`, à manier avec prudence.

Pour aller plus loin

Outil	Ce qu'il apporte
Podman (`apt install podman`)	Les mêmes commandes que Docker, sans service qui tourne en `root`. Un utilisateur ordinaire lance ses propres conteneurs
LXC, **Incus**	Des conteneurs qui ressemblent à des machines entières, avec systemd et SSH, pour ceux qui veulent une petite machine légère
Kubernetes	Répartit des conteneurs sur des dizaines de machines, les relance, les met à l'échelle. Indispensable à grande échelle, démesuré pour un serveur

Podman accepte la plupart des commandes de ce chapitre telles quelles : `alias docker=podman` suffit souvent. Kubernetes se découvre sans grand serveur avec k3s ou MicroK8s, deux distributions légères.

En résumé

- Un conteneur est un groupe de processus isolés par le noyau. Il partage le noyau de l'hôte.
- Docker s'installe depuis son dépôt officiel. Le groupe `docker` équivaut à `root`.
- `docker run -d --name … -p …`, `docker ps`, `docker logs`, `docker exec`, `docker rm -f`.
- Un port publié passe à côté d'UFW : publiez sur `127.0.0.1` et laissez Nginx faire face au réseau.
- Les données vivent dans des volumes. Un montage lié prête un dossier de l'hôte, en lecture seule avec `:ro`.
- Sur un réseau créé pour elle, une application joint ses conteneurs par leur nom.
- Un Dockerfile décrit une image, couche par couche. `USER` et `HEALTHCHECK` en font une image soignée.
- Docker Compose décrit une application entière dans `compose.yaml`, les secrets dans `.env`.
- Traefik route vers les conteneurs d'après leurs étiquettes, au prix d'un accès à la prise de Docker.

- Mettre à jour : `docker compose pull` puis `docker compose up -d`, régulièrement.

Atelier — WordPress en conteneurs

Objectif : déployer WordPress et MariaDB avec Docker Compose, derrière le Nginx de l'hôte, en HTTPS, avec une sauvegarde de la base. Comptez une heure et demie.

Docker

1. Installez Docker Engine depuis le dépôt officiel. Ajoutez votre compte au groupe `docker` et ouvrez une nouvelle session.
2. Lancez `hello-world`, puis un `nginx:alpine` publié sur le port 8081. Vérifiez depuis le poste qu'il est joignable malgré UFW. Supprimez-le.

L'application

3. Dans `~/wordpress`, écrivez le `compose.yaml` du chapitre et un `.env` qui contient un mot de passe tiré au hasard. Protégez `.env`.
4. Lancez l'application. Vérifiez que la base est `healthy`, que WordPress répond sur `127.0.0.1:8080`, et que le port 3306 n'est publié nulle part.

L'accès

5. Ajoutez un nom `wp.maison.lan` à votre DNS local, et un site Nginx qui relaie vers `127.0.0.1:8080`.
6. Obtenez un certificat avec Certbot. Terminez l'installation de WordPress depuis le navigateur du poste.

La durée

7. Écrivez un script qui exporte la base dans un fichier daté. Lancez-le, et vérifiez que le fichier contient les tables de WordPress.
8. Supprimez l'application avec `docker compose down`, puis relancez-la. Vérifiez que votre site et votre compte ont survécu.

Vérifier

- `https://wp.maison.lan` affiche votre site, avec un certificat valide pour le poste.
- `sudo ss -ltn` ne montre aucun port de Docker ouvert sur une autre adresse que `127.0.0.1`.

- `docker compose down` suivi de `docker compose up -d` ne perd rien.

Exercices

Exercice 1 — Où sont les données ?

Un collègue a lancé `docker run -d --name base -e MARIADB_ROOT_PASSWORD=… mariadb:11.4`, a rempli la base, puis a fait `docker rm -f base` pour changer un réglage. Qu'a-t-il perdu ? Comment aurait-il dû lancer le conteneur ? Comment savoir si un conteneur utilise un volume ?

Exercice 2 — Le pare-feu

Sur votre serveur, lancez un conteneur publié avec `-p 8081:80`. Vérifiez qu'il est joignable depuis le poste. Relancez-le pour qu'il ne le soit plus, sans toucher à UFW. Quelle commande vous montre les règles que Docker a écrites ?

Exercice 3 — Une image

Écrivez un Dockerfile pour le script `sauvegarde.sh` du chapitre 10, à partir de `alpine`. Le conteneur doit sauvegarder un dossier monté en lecture seule vers un dossier monté en écriture. Construisez l'image, et vérifiez qu'elle tourne sous un compte ordinaire.

Exercice 4 — Compose

Ajoutez au `compose.yaml` de WordPress un service `adminer`, une interface web pour la base, publiée sur `127.0.0.1:8085` seulement. Joignez-le depuis le poste par un tunnel SSH, comme au chapitre 26.

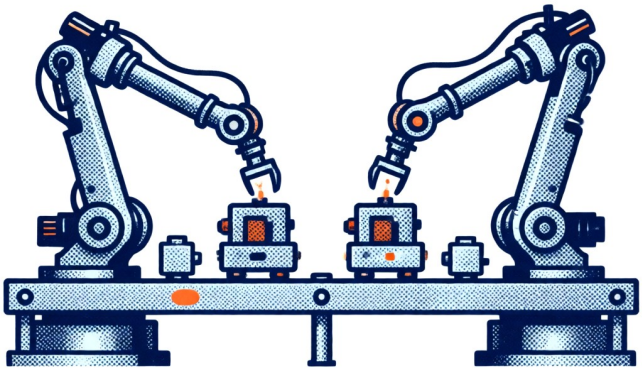
Exercice 5 — Uptime Kuma

Déployez Uptime Kuma avec Docker Compose, derrière Nginx, sur `kuma.maison.lan`. Faites-lui surveiller `https://wp.maison.lan` et `https://blog.maison.lan`. Arrêtez WordPress et observez la réaction. Pourquoi cette surveillance serait-elle plus utile sur une autre machine que le serveur ?

Pour aller plus loin

Sur la machine. `docker help` et `docker compose help` listent toutes les commandes. `docker inspect` décrit un conteneur, une image, un réseau ou un volume dans le moindre détail. `man docker-run` et les pages voisines sont installées avec le paquet.

En ligne. La documentation officielle, `docs.docker.com`, est complète et à jour, en particulier la référence de Compose et le guide des bonnes pratiques pour écrire un Dockerfile. La page des images officielles sur Docker Hub dit, pour chacune, les variables qu'elle accepte : c'est là qu'on apprend que l'image MariaDB attend `MARIADB_DATABASE`. Le projet `ufw-docker`, sur GitHub, propose des règles qui font cohabiter Docker et UFW pour ceux qui doivent publier des ports directement.



CHAPITRE 29

Automatiser avec Ansible

Le chapitre 20 a durci un serveur en une heure, réglage après réglage. Le deuxième serveur prendra la même heure, avec le risque d'oublier une étape. Ansible écrit cette heure de travail dans des fichiers, une fois pour toutes, et l'applique à une machine comme à cent. Relancé, il ne refait que ce qui manque.

DANS CE CHAPITRE

- Comprendre ce qu'apporte l'infrastructure décrite par du code.
- Installer Ansible sur un poste et décrire ses machines dans un inventaire.
- Lancer des commandes ponctuelles sur plusieurs machines.
- Écrire un playbook avec les modules courants, des variables et des modèles.
- Déclencher une action seulement quand un fichier change.
- Ranger un travail dans un rôle réutilisable.
- Chiffrer les secrets avec Ansible Vault.
- Vérifier qu'un playbook est idempotent.

> **AVANT DE COMMENCER**
>
> Les chapitres 19, 20, 21 et 27. Le poste `vostro` se connecte aux serveurs avec sa clé SSH. Une seconde machine, `neuf`, vient d'être installée : Ubuntu Server 24.04 vierge, à l'adresse 192.168.1.51, avec le compte `gary` et la clé du poste.

29.1 Décrire plutôt que faire

Un serveur configuré à la main est une œuvre unique. Personne ne sait exactement ce qu'on y a changé, ni dans quel ordre. Le jour où il faut le reconstruire, ou en monter un second identique, on recommence en espérant ne rien oublier.

L'**infrastructure décrite par du code** (*Infrastructure as Code*) renverse la démarche. On écrit dans des fichiers l'état voulu des machines : tels paquets installés, tel fichier avec tel contenu, tel service arrêté. Un outil compare cet état à la réalité et corrige les écarts. Les fichiers se rangent dans Git, comme au chapitre 27 : chaque changement a une date, un auteur et une raison, et l'on peut relire, comparer, revenir en arrière.

Ansible est l'outil le plus répandu pour cela. Il a deux particularités. Il n'installe rien sur les machines qu'il gère : il s'y connecte par SSH et y fait exécuter ses tâches par Python, présent sur toute Ubuntu. Et ses fichiers sont écrits en YAML, lisibles par qui ne programme pas.

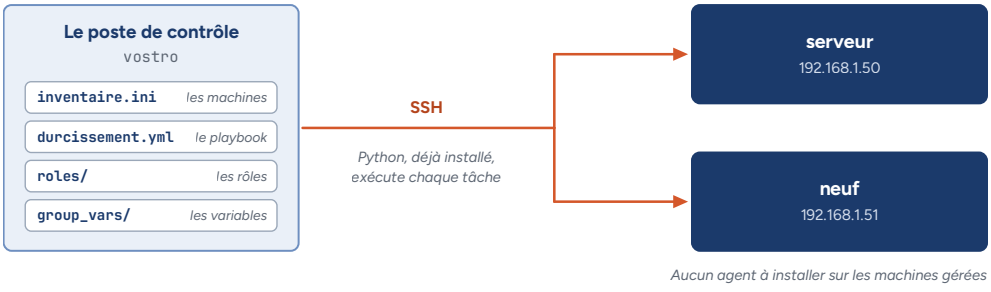


Figure 29.1 — Ansible travaille depuis un poste de contrôle, par SSH.

REPÈRE HISTORIQUE · Sans agent

Ansible a été écrit en 2012 par Michael DeHaan, qui avait travaillé sur les outils d'installation de Red Hat. Les outils du moment, Puppet et Chef, demandaient d'installer un programme permanent sur chaque machine et un serveur central pour les commander. DeHaan voulait n'avoir besoin que de SSH. Red Hat a racheté Ansible en 2015. Le nom vient de la science-fiction : l'ansible y est un appareil qui communique instantanément à travers l'espace.

Outil	Depuis	Fonctionnement	Langage
Puppet	2005	Un agent sur chaque machine, un serveur central	Un langage à lui
Chef	2009	Un agent sur chaque machine, un serveur central	Ruby
Salt	2011	Un agent, ou SSH	YAML
Ansible	2012	SSH, sans agent	YAML

Les quatre font le même métier. Pour quelques serveurs, l'absence d'agent fait la différence : il n'y a rien à installer, rien à maintenir, rien de plus qui écoute sur le réseau.

29.2 Installer Ansible

Ansible s'installe sur le **poste de contrôle**, la machine d'où l'on commande, et nulle part ailleurs :

```
gary@vostro:~$ sudo apt install ansible
[…]
gary@vostro:~$ ansible --version | head -n 1
ansible [core 2.16.3]
gary@vostro:~$ ansible-galaxy collection list 2>/dev/null | \
    grep -E '^(community.general|ansible.posix) '
ansible.posix                           1.5.4
community.general                       8.3.0
```

Le paquet `ansible` d'Ubuntu contient le moteur, `ansible-core`, et des **collections** de modules prêts à l'emploi. Les deux listées ici serviront dans ce chapitre.

29.3 L'inventaire

On travaille dans un dossier de projet, qui se rangera dans Git. Le fichier `ansible.cfg` y fixe les réglages, ici le nom de l'inventaire :

```
[defaults]
inventory = inventaire.ini
```

L'**inventaire** énumère les machines, regroupées :

```
[serveurs]
serveur ansible_host=192.168.1.50
neuf    ansible_host=192.168.1.51

[serveurs:vars]
ansible_user=gary
```

Le groupe `serveurs` contient deux machines. `ansible_host` donne leur adresse, et la section `[serveurs:vars]` fixe le compte de connexion pour tout le groupe. Ansible connaît aussi deux groupes implicites : `all`, toutes les machines, et `ungrouped`, celles qui ne sont dans aucun groupe.

```
gary@vostro:~/ansible$ ansible-inventory --graph
@all:
  |--@ungrouped:
  |--@serveurs:
  |  |--serveur
  |  |--neuf
```

Ansible se connecte comme `ssh` le ferait : il vérifie l'empreinte de chaque machine. Pour une machine neuve, on se connecte une première fois à la main, en vérifiant l'empreinte comme au chapitre 19 :

```
gary@vostro:~/ansible$ ssh 192.168.1.51 hostname
The authenticity of host '192.168.1.51 (192.168.1.51)' can't be established.
ED25519 key fingerprint is SHA256:roMNEsI439MjTIglFjCHSpJu6FKJEKAqBjfs54ZGqgM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.51' (ED25519) to the list of known hosts.
[…]
neuf
```

29.4 Des commandes ponctuelles

La commande `ansible` lance un module sur un groupe de machines. Le module `ping` vérifie qu'Ansible peut se connecter et trouver Python :

```
gary@vostro:~/ansible$ ansible all -m ping
neuf | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
serveur | SUCCESS => {
[…]
```

Sans `-m`, Ansible lance une commande :

```
gary@vostro:~/ansible$ ansible serveurs -a "uptime"
neuf | CHANGED | rc=0 >>
 01:44:40 up 6 min,  1 user,  load average: 0.08, 0.02, 0.01
serveur | CHANGED | rc=0 >>
 01:44:40 up  3:24,  2 users,  load average: 0.00, 0.00, 0.05
```

Les deux machines ont répondu en même temps. Ansible travaille en parallèle, cinq machines à la fois par défaut. `neuf` vient de démarrer, `serveur` tourne depuis plus de trois heures.

Avant de travailler, Ansible relève sur chaque machine des dizaines d'informations, les **faits** : le système, la mémoire, les interfaces, les disques. Le module `setup` les affiche :

```
gary@vostro:~/ansible$ ansible neuf -m setup -a 'filter=ansible_distribution*'
neuf | SUCCESS => {
    "ansible_facts": {
        "ansible_distribution": "Ubuntu",
        […]
        "ansible_distribution_release": "noble",
        "ansible_distribution_version": "24.04",
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false
}
```

Devenir root

Les commandes tournent sous le compte de connexion. Pour administrer, il faut `sudo`, qu'Ansible appelle **become** :

```
gary@vostro:~/ansible$ ansible neuf -a "cat /etc/shadow"
neuf | FAILED | rc=1 >>
cat: /etc/shadow: Permission deniednon-zero return code
gary@vostro:~/ansible$ ansible neuf -b -K -a "head -n 1 /etc/shadow"
BECOME password:
neuf | CHANGED | rc=0 >>
root:*:20722:0:99999:7:::
```

`-b` demande de passer par `sudo`, `-K` fait demander le mot de passe de `sudo`, une fois pour toutes les machines. Installons Lynis sur la machine neuve, et mesurons-la avant d'y toucher :

```
gary@vostro:~/ansible$ ansible neuf -b -K -m apt -a "name=lynis update_cache=true" | \
    head -n 3
BECOME password:
neuf | CHANGED => {
[…]
gary@vostro:~/ansible$ ansible neuf -b -K -a "lynis audit system --quick --no-colors" | \
    grep -E 'Hardening index|Tests performed'
BECOME password:
  Hardening index : 59 [###########        ]
  Tests performed : 255
```

Une Ubuntu fraîchement installée obtient 59. Gardons ce chiffre.

29.5 Le premier playbook

Les commandes ponctuelles dépannent. Le vrai travail s'écrit dans un **playbook**, un fichier YAML qui énumère des **tâches**. Chaque tâche appelle un **module** avec ses paramètres :

```yaml
- name: Préparer les serveurs
  hosts: serveurs
  become: true

  tasks:
    - name: Installer quelques outils
      ansible.builtin.apt:
        name: [htop, tree, ncdu]
        state: present
        update_cache: true
        cache_valid_time: 3600
```

```
    - name: Écrire le message d'accueil
      ansible.builtin.template:
        src: templates/motd.j2
        dest: /etc/motd
        mode: "0644"
```

Ligne	Sens
`name`	Ce que fait le jeu, ou la tâche. C'est ce qui s'affiche
`hosts`	Les machines visées : un groupe de l'inventaire
`become: true`	Toutes les tâches passent par `sudo`
`ansible.builtin.apt`	Le module qui gère les paquets
`cache_valid_time`	Ne relancer `apt update` que si la dernière date de plus d'une heure

La seconde tâche fabrique `/etc/motd`, le message affiché à la connexion, à partir d'un **modèle** :

```
{{ ansible_hostname }} — Ubuntu {{ ansible_distribution_version }}
{{ ansible_processor_vcpus }} processeur(s), {{ (ansible_memtotal_mb / 1024)
| round(1) }} Go de mémoire
Machine gérée par Ansible : les modifications à la main seront écrasées.
```

Les doubles accolades sont remplacées par la valeur des faits de chaque machine. Ce langage de modèles s'appelle **Jinja2**. Il sait aussi calculer et transformer : `| round(1)` est un **filtre**, qui arrondit ici la mémoire convertie en gigaoctets.

```
gary@vostro:~/ansible$ ansible-playbook premier.yml -K
BECOME password:

PLAY [Préparer les serveurs] ***************************************************

TASK [Gathering Facts] *********************************************************
ok: [neuf]
ok: [serveur]

TASK [Installer quelques outils] ***********************************************
changed: [neuf]
changed: [serveur]

TASK [Écrire le message d'accueil] *********************************************
changed: [neuf]
changed: [serveur]

PLAY RECAP *********************************************************************
neuf                       : ok=3    changed=2    unreachable=0    failed=0    […]
serveur                    : ok=3    changed=2    unreachable=0    failed=0    […]
```

`ok` signifie que l'état était déjà le bon, `changed` qu'Ansible a dû le modifier. Le bilan final compte les deux par machine. Relançons exactement le même playbook :

```
gary@vostro:~/ansible$ ansible-playbook premier.yml -K
[…]
TASK [Installer quelques outils] ***********************************************
ok: [neuf]
ok: [serveur]

TASK [Écrire le message d'accueil] **********************************************
ok: [serveur]
ok: [neuf]

PLAY RECAP *********************************************************************
neuf                       : ok=3    changed=0    unreachable=0    failed=0    […]
serveur                    : ok=3    changed=0    unreachable=0    failed=0    […]
```

Rien n'a changé : les paquets sont là, le fichier a le bon contenu. C'est la propriété la plus importante d'Ansible, l'**idempotence** : appliquer deux fois la même chose donne le même résultat qu'une seule. Un playbook se relance donc sans crainte, après une modification, sur une machine neuve ou pour vérifier qu'une autre n'a pas dérivé.

```
gary@vostro:~/ansible$ ssh 192.168.1.51 cat /etc/motd
neuf — Ubuntu 24.04
1 processeur(s), 0.9 Go de mémoire
Machine gérée par Ansible : les modifications à la main seront écrasées.
```

29.6 Les modules essentiels

Ansible compte des milliers de modules. Une poignée suffit à l'essentiel de l'administration :

Module	Rôle	Équivalent à la main
`apt`	Installer, retirer, mettre à jour des paquets	`apt install`
`copy`	Déposer un fichier, avec ses droits	`cp`, `chmod`, `chown`
`template`	Déposer un fichier fabriqué à partir d'un modèle	Un fichier édité pour chaque machine
`lineinfile`	Garantir la présence d'une ligne dans un fichier	`nano` sur une ligne

Module	Rôle	Équivalent à la main
`file`	Créer un dossier, un lien, régler des droits	`mkdir`, `ln -s`, `chmod`
`user`, `group`	Gérer les comptes	`useradd`, `usermod`, `groupadd`
`service`, `systemd_service`	Démarrer, arrêter, activer un service	`systemctl`
`command`, `shell`	Lancer une commande	La commande elle-même

Les noms complets portent leur collection : `ansible.builtin.copy` pour les modules livrés avec Ansible, `ansible.posix.sysctl` ou `community.general.ufw` pour les autres. Le nom court fonctionne souvent, le nom complet évite toute ambiguïté.

`command` et `shell` sont les derniers recours. Un module sait vérifier l'état avant d'agir ; une commande, non. Ansible la lance chaque fois et la déclare `CHANGED`, comme on l'a vu pour `uptime`. Quand on ne peut pas s'en passer, `creates: fichier` évite de la relancer si le fichier existe déjà, et `changed_when` dit à Ansible quand considérer qu'elle a changé quelque chose.

29.7 Un rôle : le durcissement du chapitre 20

À la fin du chapitre 20, vous teniez la liste des réglages appliqués au serveur et leur raison. Transformons-la en automatisation, et appliquons-la à la machine neuve.

La structure d'un rôle

Un playbook qui grossit devient illisible. On le découpe en **rôles**, des dossiers à la structure fixe qu'Ansible sait lire :

```
gary@vostro:~/ansible$ tree roles
roles
`-- durcissement
    |-- defaults
    |   `-- main.yml
    |-- files
    |   |-- 40-banniere.conf
    |   |-- 50-politique.conf
    |   |-- 52unattended-upgrades-local
    |   |-- comptes.rules
    |   |-- deploiement
    |   |-- issue.net
    |   `-- usb-storage.conf
    |-- handlers
```

```
    |   `-- main.yml
    |-- tasks
    |   `-- main.yml
    `-- templates
        `-- 10-durcissement.conf.j2

7 directories, 11 files
```

Dossier	Contenu
`tasks`	Les tâches, dans `main.yml`
`files`	Les fichiers que `copy` dépose tels quels
`templates`	Les modèles Jinja2 que `template` remplit
`defaults`	Les valeurs par défaut des variables
`handlers`	Les actions déclenchées par un changement

Les fichiers de `files` sont exactement ceux du chapitre 20. Le playbook, lui, se réduit à trois lignes utiles :

```yaml
- name: Durcir les serveurs
  hosts: serveurs
  become: true
  roles:
    - durcissement
```

Les tâches

```yaml
# Le durcissement du chapitre 20, plus SSH (chapitre 19) et le pare-feu (chapitre 21)

- name: Installer les paquets de sécurité
  ansible.builtin.apt:
    name: [unattended-upgrades, libpam-pwquality, auditd, ufw]
    state: present
    update_cache: true
    cache_valid_time: 3600

- name: Redémarrer seul après une mise à jour qui l'exige
  ansible.builtin.copy:
    src: 52unattended-upgrades-local
    dest: /etc/apt/apt.conf.d/52unattended-upgrades-local
    mode: "0644"

- name: Créer le dossier des réglages de pwquality
  ansible.builtin.file:
    path: /etc/security/pwquality.conf.d
    state: directory
    mode: "0755"
```

```
- name: Imposer des mots de passe de douze caractères
  ansible.builtin.copy:
    src: 50-politique.conf
    dest: /etc/security/pwquality.conf.d/50-politique.conf
    mode: "0644"

- name: Créer le compte de déploiement
  ansible.builtin.user:
    name: deploiement
    shell: /bin/bash

- name: Lui permettre de relancer cron, et rien d'autre
  ansible.builtin.copy:
    src: deploiement
    dest: /etc/sudoers.d/deploiement
    mode: "0440"
    validate: visudo -cf %s

- name: Recenser les services
  ansible.builtin.service_facts:

- name: Arrêter les services inutiles
  ansible.builtin.systemd_service:
    name: "{{ item }}"
    state: stopped
    enabled: false
  loop: "{{ services_inutiles }}"
  when: item in ansible_facts.services

- name: Poser les réglages du noyau
  ansible.posix.sysctl:
    name: "{{ item.key }}"
    value: "{{ item.value }}"
    sysctl_file: /etc/sysctl.d/60-durcissement.conf
  loop: "{{ reglages_noyau | dict2items }}"

- name: Empêcher UFW de remettre log_martians à 0
  ansible.builtin.lineinfile:
    path: /etc/ufw/sysctl.conf
    regexp: "^net/ipv4/conf/{{ item }}/log_martians="
    line: "net/ipv4/conf/{{ item }}/log_martians=1"
  loop: [all, default]

- name: Interdire les clés USB
  ansible.builtin.copy:
    src: usb-storage.conf
    dest: /etc/modprobe.d/usb-storage.conf
    mode: "0644"

- name: Créer le groupe autorisé à se connecter
  ansible.builtin.group:
    name: "{{ groupe_ssh }}"

- name: Y placer les administrateurs
  ansible.builtin.user:
    name: "{{ item }}"
    groups: "{{ groupe_ssh }}"
    append: true
```

```
    loop: "{{ admins }}"

  - name: Écrire la bannière
    ansible.builtin.copy:
      src: issue.net
      dest: /etc/issue.net
      mode: "0644"

  - name: Régler SSH
    ansible.builtin.template:
      src: 10-durcissement.conf.j2
      dest: /etc/ssh/sshd_config.d/10-durcissement.conf
      mode: "0644"
      validate: /usr/sbin/sshd -t -f %s
    notify: Recharger ssh

  - name: Afficher la bannière à la connexion
    ansible.builtin.copy:
      src: 40-banniere.conf
      dest: /etc/ssh/sshd_config.d/40-banniere.conf
      mode: "0644"
    notify: Recharger ssh

  - name: Surveiller les fichiers des comptes
    ansible.builtin.copy:
      src: comptes.rules
      dest: /etc/audit/rules.d/comptes.rules
      mode: "0640"
    notify: Charger les règles d'audit

  - name: Autoriser SSH, avec limitation
    community.general.ufw:
      rule: limit
      name: OpenSSH

  - name: Activer le pare-feu, tout refuser par défaut
    community.general.ufw:
      state: enabled
      policy: deny
```

On reconnaît le chapitre 20, étape par étape, et quelques idées nouvelles.

Valider avant d'écrire. `validate` fait vérifier le fichier avant de le mettre en place : `visudo -cf` pour sudo, `sshd -t` pour SSH. Un fichier faux n'atteint jamais sa destination. C'est la protection du chapitre 20 contre un `sudoers` cassé, appliquée d'office.

Boucler. `loop` répète une tâche pour chaque élément d'une liste. Le filtre `dict2items` transforme le dictionnaire des réglages du noyau en liste de paires `key` et `value`.

Conditionner. `service_facts` relève les services de la machine, et `when` ne fait arrêter un service que s'il existe. Le même rôle s'applique ainsi à des machines différentes.

L'ordre compte. Le groupe `acces-ssh` est créé et `gary` y est placé avant que SSH ne soit réglé avec `AllowGroups acces-ssh`. Dans l'ordre inverse, Ansible s'enfermerait dehors, et vous avec. De même, SSH est autorisé dans le pare-feu avant que celui-ci ne soit activé.

Les variables

Les valeurs propres à chaque parc ne s'écrivent pas dans les tâches. Le rôle déclare les siennes dans `defaults/main.yml` :

```
# Valeurs par défaut du rôle, à remplacer dans group_vars ou host_vars
groupe_ssh: acces-ssh
admins: []
ssh_essais_max: 3

services_inutiles:
  - ModemManager.service
  - multipathd.service

reglages_noyau:
  net.ipv4.conf.all.accept_redirects: 0
  net.ipv4.conf.default.accept_redirects: 0
  net.ipv6.conf.all.accept_redirects: 0
  net.ipv6.conf.default.accept_redirects: 0
  net.ipv4.conf.all.send_redirects: 0
  net.ipv4.conf.default.send_redirects: 0
  net.ipv4.conf.all.rp_filter: 1
  net.ipv4.conf.default.rp_filter: 1
  net.ipv4.conf.all.log_martians: 1
```

Et le projet les complète ou les remplace dans `group_vars`, un fichier par groupe de l'inventaire :

```
groupe_ssh: acces-ssh
admins: [gary]
```

Ansible cherche une variable à de nombreux endroits, du moins prioritaire au plus prioritaire. Retenez l'ordre de ceux que vous utiliserez :

Endroit	Priorité
`roles/…/defaults/main.yml`	La plus faible : les valeurs à remplacer
`group_vars/groupe/`	Pour tout un groupe
`host_vars/machine/`	Pour une machine

Endroit	Priorité
`-e variable=valeur` sur la ligne de commande	La plus forte

Les variables servent aussi dans les modèles. Celui de SSH reprend le fichier du chapitre 19 :

```
# Géré par Ansible, rôle durcissement
PasswordAuthentication no
PermitRootLogin no
AllowGroups {{ groupe_ssh }}
MaxAuthTries {{ ssh_essais_max }}
X11Forwarding no
```

Les handlers

Il faut recharger SSH quand sa configuration change, et seulement dans ce cas. C'est le rôle des **handlers** : une tâche qui modifie quelque chose les prévient par `notify`, et ils s'exécutent une seule fois, à la fin, même si plusieurs tâches les ont prévenus.

```
- name: Recharger ssh
  ansible.builtin.service:
    name: ssh
    state: reloaded

- name: Charger les règles d'audit
  ansible.builtin.command: augenrules --load
```

Le nom écrit dans `notify` doit être exactement celui du handler.

Essayer à blanc

Avant de toucher au serveur en service, demandons à Ansible ce qu'il ferait. `--check` simule sans rien modifier, `--diff` montre les différences dans les fichiers, `--limit` restreint à une machine :

```
gary@vostro:~/ansible$ ansible-playbook durcissement.yml -K --check --diff --limit serveur
[…]
TASK [durcissement : Poser les réglages du noyau] ***********************************
ok: [serveur] => (item={'key': 'net.ipv4.conf.all.accept_redirects', 'value': 0})
[…]
changed: [serveur] => (item={'key': 'net.ipv4.conf.all.log_martians', 'value': 1})
[…]
TASK [durcissement : Régler SSH] ****************************************************
--- before: /etc/ssh/sshd_config.d/10-durcissement.conf
+++ after: /home/gary/.ansible/tmp/ansible-local-76923wsvrexg/…/10-durcissement.conf.j2
```

```
@@ -1,4 +1,4 @@
-# Connexion par clé uniquement, jamais en root, réservée à un groupe
+# Géré par Ansible, rôle durcissement
 PasswordAuthentication no
 PermitRootLogin no
 AllowGroups acces-ssh

changed: [serveur]
[…]
RUNNING HANDLER [durcissement : Recharger ssh] ***************************************
changed: [serveur]

PLAY RECAP ********************************************************************************
serveur                    : ok=19   changed=3    unreachable=0    failed=0    […]
```

Le serveur a été durci à la main au chapitre 20 : presque tout est déjà `ok`. Le fichier de SSH ne diffère que par son commentaire. Un réglage du noyau, en revanche, n'a pas la valeur attendue. Nous y reviendrons.

Appliquer

```
gary@vostro:~/ansible$ ansible-playbook durcissement.yml -K
[…]
TASK [durcissement : Installer les paquets de sécurité] ******************************
ok: [serveur]
changed: [neuf]
[…]
TASK [durcissement : Imposer des mots de passe de douze caractères] ******************
ok: [serveur]
fatal: [neuf]: FAILED! => {"changed": false, "checksum": "ac78ac2fdc121f2f23c3a0230a…",
  "msg": "Destination directory /etc/security/pwquality.conf.d does not exist"}
[…]
PLAY RECAP ********************************************************************************
neuf                       : ok=3    changed=2    unreachable=0    failed=1    […]
serveur                    : ok=19   changed=3    unreachable=0    failed=0    […]
```

Sur `neuf`, Ansible s'est arrêté net à la quatrième tâche, sans toucher au reste ; sur `serveur`, il a continué. Une erreur sur une machine n'interrompt pas les autres.

SUR LE TERRAIN · Ce que la machine neuve a révélé

Le dossier `/etc/security/pwquality.conf.d` n'existe pas après l'installation de `libpam-pwquality`. Sur le serveur, il avait été créé pendant les essais du chapitre 20, sans que le texte le dise : le lecteur qui suivait le chapitre à la lettre aurait buté au même endroit. Le chapitre 20 a été corrigé, et le rôle a reçu une tâche de plus, avant la copie :

```
- name: Créer le dossier des réglages de pwquality
  ansible.builtin.file:
    path: /etc/security/pwquality.conf.d
    state: directory
    mode: "0755"
```

C'est l'un des grands mérites de l'automatisation. Une machine neuve ne cache rien : tout ce qui manque à la description apparaît au premier passage.

Relancer, et ne rien changer

Le rôle corrigé est relancé. `neuf` reçoit tout le durcissement, et ses handlers s'exécutent à la fin :

```
gary@vostro:~/ansible$ ansible-playbook durcissement.yml -K
[…]
RUNNING HANDLER [durcissement : Recharger ssh] ************************************
changed: [neuf]

RUNNING HANDLER [durcissement : Charger les règles d'audit] ***************************
changed: [neuf]

PLAY RECAP ************************************************************************
neuf                       : ok=22   changed=18   unreachable=0    failed=0    […]
serveur                    : ok=20   changed=1    unreachable=0    failed=0    […]
```

Puis une seconde fois, pour vérifier l'idempotence :

```
gary@vostro:~/ansible$ ansible-playbook durcissement.yml -K | tail -n 4
BECOME password:
PLAY RECAP ************************************************************************
neuf                       : ok=20   changed=0    unreachable=0    failed=0    […]
serveur                    : ok=20   changed=0    unreachable=0    failed=0    […]
```

Zéro changement sur les deux machines. Ce n'était pas le cas avant la correction suivante.

SUR LE TERRAIN · Le réglage qui revenait toujours

Lors des premiers passages, la tâche des réglages du noyau annonçait sur le serveur `changed` pour `log_martians`, à chaque fois. Un playbook idempotent ne change rien la seconde fois : quelque chose remettait donc la valeur à 0 entre deux passages.

```
gary@serveur:~$ grep -n martians /etc/ufw/sysctl.conf
29:net/ipv4/conf/all/log_martians=0
30:net/ipv4/conf/default/log_martians=0
```

UFW, installé au chapitre 21, applique ses propres réglages du noyau à chaque démarrage et à chaque rechargement. Ils l'emportent sur `/etc/sysctl.d/`. Le réglage posé au chapitre 20 était donc annulé depuis le chapitre 21, sans que rien ne le signale. Le rôle corrige désormais aussi ce fichier, avec `lineinfile` :

```
- name: Empêcher UFW de remettre log_martians à 0
  ansible.builtin.lineinfile:
    path: /etc/ufw/sysctl.conf
    regexp: "^net/ipv4/conf/{{ item }}/log_martians="
    line: "net/ipv4/conf/{{ item }}/log_martians=1"
  loop: [all, default]
```

`regexp` désigne la ligne à remplacer, `line` son nouveau contenu. Si aucune ligne ne correspond, `line` est ajoutée à la fin. Un `changed` qui revient à chaque passage n'est jamais anodin : il signale que deux choses se disputent la machine.

Vérifier

Ansible dit qu'il a appliqué le rôle. Vérifions sur `neuf`, réglage par réglage :

```
gary@vostro:~/ansible$ ssh 192.168.1.51 true
Serveur de travail, usage restreint.
Les connexions sont inscrites au journal.
gary@vostro:~/ansible$ ansible neuf -a \
    "sysctl net.ipv4.conf.all.log_martians net.ipv4.conf.all.accept_redirects"
neuf | CHANGED | rc=0 >>
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.all.accept_redirects = 0
gary@vostro:~/ansible$ ansible neuf -b -K -a "ufw status"
BECOME password:
neuf | CHANGED | rc=0 >>
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    LIMIT       Anywhere
OpenSSH (v6)               LIMIT       Anywhere (v6)
gary@vostro:~/ansible$ ansible neuf -b -K -a "auditctl -l"
```

```
BECOME password:
neuf | CHANGED | rc=0 >>
-w /etc/passwd -p wa -k comptes
-w /etc/shadow -p wa -k comptes
-w /etc/sudoers.d -p wa -k comptes
gary@vostro:~/ansible$ ansible neuf -b -K -a "sudo -l -U deploiement" | tail -n 2
BECOME password:
User deploiement may run the following commands on neuf:
    (root) NOPASSWD: /usr/bin/systemctl restart cron
```

La bannière, le noyau, le pare-feu, l'audit, sudo : tout y est. Et Lynis :

```
gary@vostro:~/ansible$ ansible neuf -b -K -a "lynis audit system --quick --no-colors" | \
    grep -E 'Hardening index|Tests performed'
BECOME password:
  Hardening index : 65 [#############       ]
  Tests performed : 258
```

De 59 à 65, en une commande. Le serveur du chapitre 20 avait atteint 70 : il avait reçu en plus toutes les mises à jour en attente et des options de montage sur `/srv/donnees`, une partition que `neuf` n'a pas. Les mises à jour s'ajoutent sans peine au rôle, avec `ansible.builtin.apt: upgrade: safe`, et l'exercice 4 s'en charge.

29.8 Les secrets avec Ansible Vault

Un projet Ansible se range dans Git, et un secret enregistré dans Git est publié, comme l'a montré le chapitre 27. Les mots de passe se chiffrent donc avec **Ansible Vault**. Créons le mot de passe d'un compte :

```
gary@vostro:~/ansible$ ansible-vault create group_vars/serveurs/secret.yml
New Vault password:
Confirm New Vault password:
```

Ansible demande un mot de passe pour le coffre, puis ouvre l'éditeur. On y écrit la variable, `mdp_alice: …`, on enregistre, et le fichier est chiffré :

```
gary@vostro:~/ansible$ head -n 3 group_vars/serveurs/secret.yml
$ANSIBLE_VAULT;1.1;AES256
38653933663838613436303764336632343838643130636230393831356237336632636465386462
34616165343263333731346532343831633234313561326 20a383130356461633831336332353937
```

Ce fichier-là peut entrer dans Git. `ansible-vault view` l'affiche en clair, `ansible-vault edit` le modifie. Le playbook qui s'en sert ne sait pas que la variable vient d'un coffre :

```yaml
- name: Créer les comptes des utilisateurs
  hosts: serveurs
  become: true

  tasks:
    - name: Créer le compte d'Alice
      ansible.builtin.user:
        name: alice
        shell: /bin/bash
        password: "{{ mdp_alice | password_hash('sha512') }}"
        update_password: on_create
```

Le filtre `password_hash` transforme le mot de passe en empreinte, la forme que `/etc/shadow` attend. Sans le mot de passe du coffre, Ansible refuse de travailler :

```
gary@vostro:~/ansible$ ansible-playbook comptes.yml -K
BECOME password:

PLAY [Créer les comptes des utilisateurs] **********************************************
ERROR! Attempting to decrypt but no vault secrets found
gary@vostro:~/ansible$ ansible-playbook comptes.yml -K --ask-vault-pass
BECOME password:
Vault password:
[…]
TASK [Créer le compte d'Alice] *********************************************************
changed: [serveur]
changed: [neuf]
[…]
```

Un second passage ne change rien : `update_password: on_create` ne pose le mot de passe qu'à la création du compte. Sans cette ligne, `password_hash` tirerait à chaque passage un nouveau sel, l'empreinte changerait, et la tâche serait `changed` à chaque fois.

Le coffre se lit dès qu'Ansible charge les variables du groupe, y compris pour une commande ponctuelle : `ansible serveurs -a "id alice"` réclame lui aussi `--ask-vault-pass`. Pour ne pas taper le mot de passe à chaque fois, `--vault-password-file` le lit dans un fichier, protégé par `chmod 600` et bien sûr absent du dépôt.

29.9 L'idempotence, en pratique

L'idempotence ne vient pas toute seule. Elle dépend de la façon d'écrire les tâches :

Plutôt que	Écrire
`command: apt install -y nginx`	`apt: name=nginx`
`shell: echo "ligne" >> fichier`	`lineinfile`
`command: systemctl restart ssh` après chaque copie	Un handler prévenu par `notify`
`command: mkdir -p /srv/x`	`file: path=/srv/x state=directory`
Une commande inévitable	`creates`, ou `changed_when`

Le test est simple : relancez le playbook. Toute tâche `changed` au second passage mérite qu'on comprenne pourquoi.

En résumé

- Ansible décrit l'état voulu des machines dans des fichiers YAML, et l'applique par SSH, sans agent.
- L'inventaire liste les machines par groupes. `ansible groupe -m module` lance une action ponctuelle.
- `-b` fait passer par `sudo`, `-K` en demande le mot de passe.
- Un playbook enchaîne des tâches, chacune appelant un module : `apt`, `copy`, `template`, `lineinfile`, `file`, `user`, `service`.
- Les modèles Jinja2 adaptent un fichier à chaque machine, avec ses faits et vos variables.
- `notify` prévient un handler, qui ne s'exécute qu'une fois, à la fin.
- Un rôle range tâches, fichiers, modèles, variables et handlers dans une structure fixe.
- `--check --diff` montre ce qui changerait. `validate` refuse un fichier invalide.
- Ansible Vault chiffre les secrets ; le fichier chiffré peut entrer dans Git.
- Un playbook idempotent ne change rien au second passage. Un `changed` qui revient toujours cache un problème.

Atelier — Reproduire le durcissement

Objectif : écrire un rôle qui applique le durcissement du chapitre 20 à une machine neuve, vérifier son effet avec Lynis, et ranger le projet dans Git. Comptez deux heures.

Le terrain

1. Installez une machine virtuelle Ubuntu Server 24.04 vierge, avec votre clé SSH. Prenez-en un instantané.
2. Sur le poste, installez Ansible. Créez un dossier de projet, un `ansible.cfg`, un inventaire avec votre serveur et la machine neuve.
3. Vérifiez l'empreinte de la machine neuve, puis `ansible all -m ping`.
4. Installez Lynis sur la machine neuve avec une commande ponctuelle, et notez son indice.

Le rôle

5. Créez la structure du rôle `durcissement`. Copiez dans `files` les fichiers du chapitre 20.
6. Écrivez les tâches, en reprenant la liste que vous teniez à la fin du chapitre 20. Utilisez `validate` pour sudo et SSH.
7. Placez les valeurs propres à votre parc dans `group_vars`.
8. Lancez le rôle avec `--check --diff` sur votre serveur. Lisez chaque différence annoncée.

L'application

9. Appliquez le rôle aux deux machines. Corrigez ce qui échoue, et relancez jusqu'à zéro échec.
10. Relancez une dernière fois : le bilan doit afficher `changed=0` partout. Sinon, cherchez la cause.
11. Mesurez la machine neuve avec Lynis.

Le rangement

12. Faites du projet un dépôt Git, avec un `.gitignore` qui exclut tout fichier de mot de passe du coffre. Enregistrez.

Vérifier

- Depuis le poste, la connexion à la machine neuve affiche la bannière.

- `ansible-playbook durcissement.yml -K` relancé n'annonce aucun changement.
- Restaurez l'instantané de la machine neuve, relancez le rôle : elle retrouve le même indice Lynis.

Exercices

Exercice 1 — Lire un bilan

```
neuf      : ok=12   changed=4    unreachable=0    failed=1    skipped=2
serveur   : ok=20   changed=0    unreachable=0    failed=0    skipped=0
web1      : ok=0    changed=0    unreachable=1    failed=0    skipped=0
```

Pour chaque machine, que s'est-il passé ? Que faites-vous ensuite ?

Exercice 2 — Commandes ponctuelles

Avec une seule commande `ansible` chacune :

1. affichez la place libre sur `/` de toutes les machines ;
2. installez `ncdu` sur le groupe `serveurs` ;
3. redémarrez le service `cron` sur `neuf` ;
4. affichez la quantité de mémoire de chaque machine, à partir des faits.

Exercice 3 — Un modèle

Écrivez un modèle `issue.net.j2` qui affiche le nom de la machine et le nom de l'organisation, tiré d'une variable `organisation` définie dans `group_vars`. Remplacez la copie de `issue.net` par ce modèle dans le rôle. Vérifiez le résultat sur les deux machines.

Exercice 4 — Les mises à jour

Ajoutez au rôle une tâche qui applique les mises à jour en attente, puis une tâche qui redémarre la machine si `/var/run/reboot-required` existe. Le module `ansible.builtin.stat` teste l'existence d'un fichier, `ansible.builtin.reboot` redémarre et attend le retour. Pourquoi faut-il être prudent en lançant ce rôle sur toutes les machines à la fois ? Cherchez ce que fait le mot-clé `serial`.

Exercice 5 — Idempotence

Cette tâche est-elle idempotente ? Réécrivez-la pour qu'elle le soit.

```
- name: Ajouter le dépôt de sauvegarde au journal
  ansible.builtin.shell: echo "dépôt : /srv/sauvegardes" >> /etc/motd
```

Pour aller plus loin

Sur la machine. `ansible-doc apt` affiche la documentation complète d'un module, avec des exemples ; `ansible-doc -l` les liste tous. `ansible-playbook --syntax-check` vérifie un playbook sans rien lancer. Le paquet `ansible-lint` relève les mauvaises pratiques, dont les tâches non idempotentes.

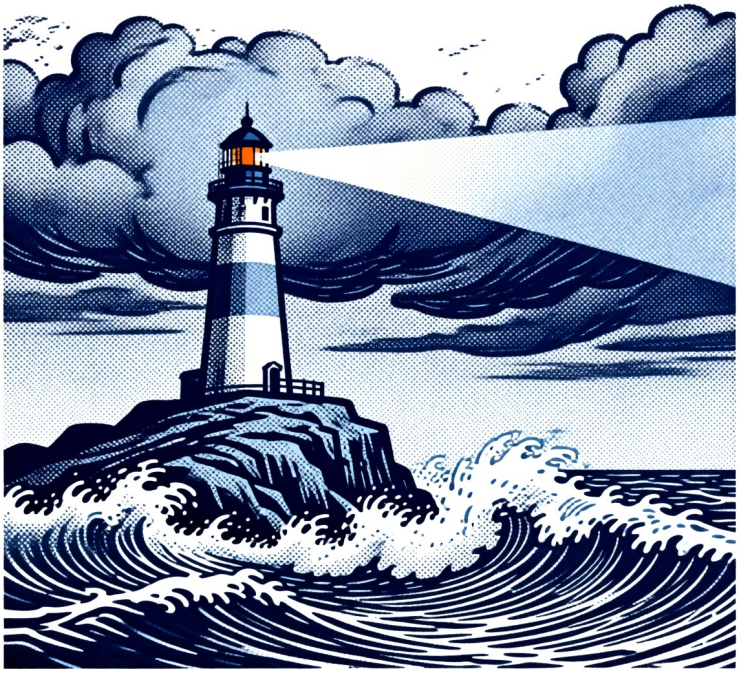
Paquets et outils. Ansible Galaxy (`ansible-galaxy`) partage des rôles et des collections écrits par d'autres ; lisez-les avant de les lancer sur vos machines. **AWX**, version libre d'Ansible Automation Platform, et **Semaphore**, plus léger, donnent une interface web pour lancer les playbooks, les planifier et garder l'historique de chaque exécution. **Molecule** teste un rôle dans des conteneurs jetables avant de l'appliquer à de vraies machines.

En ligne. La documentation officielle, `docs.ansible.com`, propose un guide de démarrage et la référence de chaque module. Son chapitre sur la priorité des variables détaille les vingt et quelques endroits où Ansible les cherche. Le projet `ansible-lockdown` publie des rôles qui appliquent les référentiels CIS cités au chapitre 20 ; lisez-les comme des exemples de rôles bien écrits.

PARTIE X

Diagnostic, performance, récupération

Savoir quoi faire quand ça va mal : une méthode pour diagnostiquer, des gestes pour réparer.





CHAPITRE 30

Diagnostiquer un incident

Un jour, le site ne répond plus, ou répond lentement, ou le disque est plein sans raison apparente. La tentation est de tout essayer en même temps. Ce chapitre propose une démarche, puis l'applique à des pannes réelles, provoquées sur le serveur du livre ou rencontrées pendant son écriture.

DANS CE CHAPITRE

- Mener un diagnostic avec méthode plutôt qu'au hasard.
- Trier un incident : l'impact, ce qui a changé, le symptôme et la cause.
- Lire les journaux du noyau et des services, y compris la trace d'un manque de mémoire.
- Mesurer la latence, les pertes et le débit du réseau.
- Observer ce que fait un programme avec strace, ltrace et lsof.
- Résoudre des pannes courantes : service qui ne démarre pas, disque plein, fork bomb, force brute, nom qui ne se résout plus.
- Rédiger un retour d'expérience.

> **AVANT DE COMMENCER**
>
> Les chapitres 12, 13, 17, 21 et 26. Les outils s'installent avec `sudo apt install strace ltrace lsof mtr-tiny iperf3`. Les pannes de ce chapitre se provoquent sur une machine d'essai, jamais sur un serveur en service.

30.1 Une méthode

Face à une panne, on a envie d'agir : redémarrer, réinstaller, modifier trois fichiers à la fois. Parfois cela marche, et l'on ne sait pas pourquoi. Souvent cela ajoute une deuxième panne à la première. Le diagnostic gagne à suivre la démarche de toute enquête.

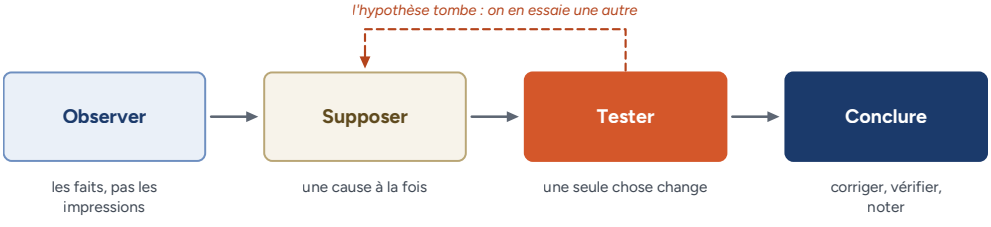


Figure 30.1 — La boucle du diagnostic.

Observer. Relever les faits : le message d'erreur exact, l'heure, ce qui marche encore, ce qui ne marche plus. « Le site est lent » est une impression. « La page d'accueil met huit secondes depuis 14 h, les autres pages sont normales » est un fait.

Supposer. Formuler une cause possible, une seule, qui explique tous les faits observés.

Tester. Chercher la mesure qui confirmera ou détruira l'hypothèse. Ne changer qu'une chose à la fois : sinon, on ne saura pas laquelle a agi.

Conclure. Corriger, vérifier que le symptôme a disparu, et noter ce qu'on a appris. Si l'hypothèse tombe, on en formule une autre, avec les faits nouveaux que le test a apportés.

> **À RETENIR · Notez au fil de l'eau**
>
> Ouvrez un fichier dès le début de l'incident, et notez-y l'heure de chaque constat et de chaque action. Ces notes servent pendant la panne, quand on ne sait plus ce qu'on a déjà essayé, et après, pour le retour d'expérience.

30.2 Le triage

Avant de chercher la cause, trois questions.

Quel est l'impact ? Un service arrêté pour tous les clients n'appelle pas la même réponse qu'une lenteur sur une page d'administration. L'impact décide de l'urgence, et parfois d'une solution provisoire avant la vraie réparation.

Qu'est-ce qui a changé ? La plupart des pannes suivent un changement : une mise à jour, un fichier modifié, un déploiement. C'est la première piste à suivre :

```
gary@serveur:~$ grep -E " (install|upgrade|remove) " /var/log/dpkg.log | tail -n 4
2026-10-05 01:49:33 install ncdu:amd64 <none> 1.19-0.1
2026-10-05 02:14:22 install libiperf0:amd64 <none> 3.16-1build2
2026-10-05 02:14:22 install iperf3:amd64 <none> 3.16-1build2
2026-10-05 02:14:22 install ltrace:amd64 <none> 0.7.3-6.4ubuntu3
gary@serveur:~$ sudo git -C /etc log --oneline | head -n 4
0ed2270 committing changes in /etc made by "apt-get install -y -q ltrace iperf3"
ab92673 saving uncommitted changes in /etc prior to apt run
ef9b898 saving uncommitted changes in /etc prior to apt run
a27bb29 committing changes in /etc made by "apt-get install -y docker-ce docker-ce-cli […]"
gary@serveur:~$ last -n 4
gary     pts/0        192.168.1.1      Mon Oct  5 02:14   still logged in
[…]
```

Le journal de `dpkg` dit ce qui a été installé et quand. L'historique d'etckeeper, mis en place au chapitre 27, dit ce qui a changé dans `/etc`. `last` dit qui s'est connecté.

Symptôme ou cause ? « Le site ne répond plus » est un symptôme. « Nginx ne démarre pas » en est un autre, plus précis. « Un autre programme occupe le port 80 » est une cause. On descend de l'un à l'autre, et l'on ne déclare l'incident résolu qu'en ayant atteint la cause.

30.3 Les journaux

Deux commandes donnent l'état général :

```
gary@serveur:~$ uptime
 02:14:43 up  3:54,  2 users,  load average: 0.13, 0.06, 0.04
gary@serveur:~$ systemctl --failed
  UNIT LOAD ACTIVE SUB DESCRIPTION

0 loaded units listed.
```

Puis le journal, filtré sur les erreurs depuis le démarrage, comme au chapitre 13 :

```
gary@serveur:~$ journalctl -p err -b --since "-3h" --no-pager | tail -n 6
[…]
Oct 05 00:05:53 serveur systemd[1]: Failed to start sauvegarde-bases.service - Export des
  bases PostgreSQL.
Oct 05 00:07:07 serveur mariadb-server.postinst[150010]: 2026-10-05  0:07:07 0 [Warning]
  You need to use --log-bin […]
Oct 05 00:15:16 serveur blkmapd[195918]: open pipe file /run/rpc_pipefs/nfs/blocklayout
  failed: No such file or directory
Oct 05 00:22:48 serveur sshd[198630]: error: /dev/pts/0: No such file or directory
```

Tout n'y est pas grave : le message de `blkmapd`, un composant de NFS, revient à chaque démarrage sans conséquence. L'échec de `sauvegarde-bases.service`, lui, mérite qu'on regarde : `journalctl -u sauvegarde-bases --since 00:05` en donnera le détail. Une erreur se juge à ce qu'elle empêche, pas à sa couleur.

Un serveur qui a redémarré seul cache souvent la réponse dans le démarrage précédent :

```
gary@serveur:~$ journalctl --list-boots | tail -n 3
 -2 92b085a6f359485db2d4d44af1115d00 Sun 2026-10-04 21:57:43 UTC Sun 2026-10-04 22:08:57 UTC
 -1 9bd4021d828f4b93a0cc6434cc13977b Sun 2026-10-04 22:11:04 UTC Sun 2026-10-04 22:17:43 UTC
  0 0af9a591d96843a8878a6252d8967d15 Sun 2026-10-04 22:19:51 UTC Mon 2026-10-05 02:14:43 UTC
```

`journalctl -b -1 -n 50` affiche la fin du démarrage précédent : les derniers mots du système avant l'arrêt. Un arrêt propre se termine par les messages de systemd qui arrête les services ; un journal qui s'interrompt net signale une coupure de courant ou un blocage complet.

Les messages du noyau ont leur propre commande, `dmesg`, et la même option de filtrage :

```
gary@serveur:~$ sudo dmesg -T --level=err,warn | tail -n 2
[Mon Oct  5 01:20:48 2026] [UFW BLOCK] IN=enp0s3 OUT=
MAC=52:54:00:12:34:56:… SRC=192.168.1.1
DST=192.168.1.50 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=58766 DF PROTO=TCP
SPT=40244 DPT=8082 […]
```

On y trouve le matériel qui flanche, les disques qui renvoient des erreurs, les processus tués faute de mémoire et, ici, les paquets refusés par UFW. Enfin, certains programmes écrivent encore dans leurs propres fichiers, sous `/var/log` : Nginx, MariaDB, Fail2ban. `ls -lt /var/log` montre ceux qui ont bougé récemment.

30.4 La performance

Le chapitre 26 a présenté les outils de mesure. En diagnostic, on les prend dans l'ordre, du plus large au plus précis :

Question	Commande	Ce qui doit alerter
La machine est-elle chargée ?	`uptime`	Une charge durablement supérieure au nombre de processeurs
Qui consomme le processeur ?	`top`, trié par `%CPU`	Un processus à 100 % qui ne devrait pas l'être
Manque-t-on de mémoire ?	`free -h`, `vmstat 2`	`available` faible, colonnes `si` et `so` non nulles
Le disque suit-il ?	`iostat -dxs 2`, `iotop`	`await` à des dizaines de millisecondes, `wa` élevé dans `top`
Qu'en était-il cette nuit ?	`sar`, `sar -r`, `sar -d`	Le moment où la courbe a changé

Une charge élevée avec un processeur peu occupé est un indice précieux : les processus attendent autre chose que le processeur, le plus souvent le disque.

La trace d'un manque de mémoire

Quand la mémoire manque, le noyau choisit un processus et le tue. Provoquons-le, sans danger pour le reste de la machine, en enfermant un programme dans une limite de 200 Mo, comme au chapitre 12 :

```
gary@serveur:~$ sudo systemd-run --scope -p MemoryMax=200M -p MemorySwapMax=0 \
    python3 -c 'a = bytearray(500 * 1024 * 1024)'
Running as unit: run-ra4fb3b3d5b6f4591a99cf1e730831a1f.scope; invocation ID: 73f44a56…
Killed
gary@serveur:~$ echo $?
137
```

Le programme demandait 500 Mo. Il n'a affiché aucune erreur : il a été tué, et son code de retour, 137, vaut 128 plus 9, le numéro du signal `KILL`. Le noyau, lui, a tout noté :

```
gary@serveur:~$ journalctl -k -n 40 --no-pager | \
    grep -E 'Memory cgroup out of memory|oom-kill:'
Oct 05 02:14:47 serveur kernel: oom-kill:constraint=CONSTRAINT_MEMCG,nodemask=(null),cpuse
  t=run-ra4f…
Oct 05 02:14:47 serveur kernel: Memory cgroup out of memory: Killed process 237991 (python3)
  total-vm:530344kB, anon-rss:204288kB, file-rss:6400kB, shmem-rss:0kB, UID:0 pgtables:476kB
  oom_score_adj:0
```

Élément	Sens
`constraint=CONSTRAINT_MEMCG`	La limite atteinte est celle d'un groupe de contrôle, pas la mémoire de la machine
`Killed process 237991 (python3)`	La victime
`anon-rss:204288kB`	La mémoire qu'elle occupait vraiment : les 200 Mo autorisés
`oom_score_adj`	Le réglage qui rend un processus plus ou moins exposé

Quand c'est la machine entière qui manque de mémoire, la ligne dit `Out of memory: Killed process` et `constraint=CONSTRAINT_NONE`. Le noyau choisit alors le processus qui occupe le plus de mémoire, ce qui tombe souvent sur la base de données. Un service qui « s'arrête tout seul la nuit » doit faire chercher cette ligne : `journalctl -k | grep -i 'killed process'`.

30.5 Le réseau

Trois mesures : la latence, les pertes, le débit. Depuis le poste, au repos :

```
gary@vostro:~$ ping -c 20 -i 0.2 -q 192.168.1.50
PING 192.168.1.50 (192.168.1.50) 56(84) bytes of data.

--- 192.168.1.50 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 3941ms
rtt min/avg/max/mdev = 0.349/0.540/0.910/0.128 ms
```

Une demi-milliseconde, aucune perte. Dégradons volontairement la ligne sur le serveur : `tc`, le régleur de trafic du noyau, sait ajouter un retard et des pertes avec le module `netem`.

```
gary@serveur:~$ sudo tc qdisc add dev enp0s3 root netem delay 100ms loss 10%
gary@vostro:~$ ping -c 30 -i 0.2 -q 192.168.1.50
[…]
30 packets transmitted, 26 received, 13.3333% packet loss, time 5832ms
rtt min/avg/max/mdev = 100.607/100.895/101.102/0.126 ms
gary@vostro:~$ mtr -r -n -c 30 192.168.1.50
Start: 2026-10-05T02:17:51+0000
HOST: vostro                      Loss%   Snt   Last   Avg  Best  Wrst StDev
  1.|-- 192.168.1.50               6.7%    30  101.0 101.0 100.7 101.3   0.2
gary@serveur:~$ sudo tc qdisc del dev enp0s3 root
```

Le retard est là, exact, et les pertes oscillent autour des 10 % demandés. `mtr`, vu au chapitre 17, montre la même chose étape par étape ; sur un vrai trajet, il dit à quel endroit les pertes commencent. Une perte qui n'apparaît qu'à une étape intermédiaire, et pas aux suivantes, vient souvent d'un routeur qui répond peu aux sondes, pas d'un vrai problème.

Ces chiffres expliquent beaucoup de lenteurs. Une page qui demande dix allers-retours prend une seconde de plus avec 100 ms de retard. Une perte de quelques pour cent fait chuter le débit des transferts, car TCP ralentit à chaque paquet perdu.

Le débit se mesure avec `iperf3`, lancé en serveur sur une machine (`iperf3 -s`) et en client sur l'autre :

```
gary@serveur:~$ iperf3 -c 192.168.1.1 -t 5
Connecting to host 192.168.1.1, port 5201
[…]
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-5.00   sec  26.3 GBytes  45.1 Gbits/sec    0             sender
[  5]   0.00-5.00   sec  26.3 GBytes  45.1 Gbits/sec                  receiver
```

45 Gbit/s : entre une machine virtuelle et son hôte, les paquets ne quittent jamais la mémoire. Sur un vrai réseau à 1 Gbit/s, on attend environ 940 Mbit/s. La colonne `Retr` compte les retransmissions : nulle ici, elle grimpe quand le réseau perd des paquets.

30.6 Ce que fait un programme

strace

Un programme échoue avec un message vague, ou n'en affiche aucun. `strace` montre ses **appels système** : chaque fois qu'il demande quelque chose au noyau, ouvrir un fichier, se connecter, lire, écrire. Voici ce que fait `curl` pour joindre le blog, limité aux ouvertures de fichiers et aux connexions :

```
gary@serveur:~$ strace -f -e trace=openat,connect \
    curl -s -o /dev/null https://blog.maison.lan 2>&1 | \
    grep -E 'resolv|hosts|certs|connect\(' | grep -v ENOENT
[pid 242435] openat(AT_FDCWD, "/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 7
[pid 242435] openat(AT_FDCWD, "/etc/hosts", O_RDONLY|O_CLOEXEC) = 7
[pid 242435] connect(7, {sa_family=AF_INET, sin_port=htons(53),
  sin_addr=inet_addr("127.0.0.53")}, 16) = 0
connect(5, {sa_family=AF_INET, sin_port=htons(443), sin_addr=inet_addr("192.168.1.50")},
  16) = -1 EINPROGRESS
openat(AT_FDCWD, "/etc/ssl/certs/ca-certificates.crt", O_RDONLY) = 6
[…]
```

En cinq lignes, toute l'histoire : `curl` lit la configuration du DNS, consulte `/etc/hosts`, interroge le résolveur local de systemd-resolved sur le port 53, se connecte au serveur sur le port 443, puis charge les autorités de certification auxquelles il fait confiance. Si `curl` refusait le certificat, on saurait ainsi quel fichier il consulte. L'option `-f` suit aussi les fils d'exécution : la résolution du nom se fait dans un fil séparé, marqué `[pid 242435]`, qu'on ne verrait pas sans elle.

Option	Effet
`-e trace=openat,connect`	Seulement ces appels
`-e trace=%file,%network`	Une famille d'appels
`-f`	Suivre les fils et les processus enfants
`-p PID`	S'attacher à un processus déjà lancé
`-o fichier`	Écrire la trace dans un fichier
`-c`	Compter les appels et leur durée, sans les détailler

`-p` permet d'observer un service en marche. Attachons-nous aux processus de Nginx pendant une requête :

```
gary@serveur:~$ sudo timeout 4 strace -f -p $(pgrep -d, -f 'nginx: worker') \
    -e trace=openat,sendfile -o /tmp/nginx.trace 2>/dev/null &
gary@serveur:~$ curl -s -o /dev/null https://blog.maison.lan/
gary@serveur:~$ grep openat /tmp/nginx.trace | grep -v ENOENT
238182 openat(AT_FDCWD, "/var/www/blog/index.html", O_RDONLY|O_NONBLOCK) = 15
```

Nginx a ouvert exactement le fichier attendu. Une page qui renvoie une erreur 404 alors que le fichier semble là se diagnostique ainsi : la trace montre le chemin réellement cherché.

Une erreur se lit en fin de ligne : `= -1 ENOENT (No such file or directory)`, `= -1 EACCES (Permission denied)`. Beaucoup de `ENOENT` sont normaux, un pro-

gramme cherchant souvent un fichier à plusieurs endroits. C'est le dernier échec avant l'arrêt qui compte.

> **ATTENTION · strace ralentit ce qu'il observe**
>
> Chaque appel système passe par strace. Un programme tracé peut tourner dix fois plus lentement. Sur un serveur en service, limitez la trace à quelques secondes et à quelques appels.

ltrace

`ltrace` fait de même pour les appels aux bibliothèques. Voici les variables d'environnement que consulte `curl` :

```
gary@serveur:~$ ltrace -e getenv curl -s -o /dev/null https://blog.maison.lan 2>&1 | \
    head -n 6
libcurl.so.4->getenv("SSLKEYLOGFILE")            = nil
libcurl.so.4->getenv("CURL_HOME")                = nil
libcurl.so.4->getenv("XDG_CONFIG_HOME")          = nil
libcurl.so.4->getenv("HOME")                     = "/home/gary"
[…]
```

Il est utile pour comprendre d'où vient un réglage, mais beaucoup moins fiable que strace sur les programmes modernes. Dans le doute, commencez par strace.

lsof

`lsof` liste les fichiers ouverts, et sous Linux presque tout est fichier : les connexions réseau aussi.

Commande	Ce qu'elle montre
`sudo lsof -i :443`	Qui utilise le port 443
`sudo lsof -nP -i TCP -sTCP:ESTABLISHED`	Les connexions établies
`sudo lsof -p PID`	Tout ce qu'un processus a ouvert
`sudo lsof -u compte`	Tout ce qu'un compte a ouvert
`sudo lsof +D /dossier`	Qui utilise un dossier
`sudo lsof +L1`	Les fichiers supprimés mais encore ouverts

```
gary@serveur:~$ sudo lsof -i :443 -sTCP:LISTEN
COMMAND    PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nginx   238180     root   11u  IPv4 508426      0t0  TCP *:https (LISTEN)
```

```
nginx   238180     root   12u  IPv6 508427      0t0  TCP *:https (LISTEN)
nginx   238181 www-data   11u  IPv4 508426      0t0  TCP *:https (LISTEN)
[…]
gary@serveur:~$ sudo lsof -p $(pgrep -o -f 'nginx: master') | grep -E '\.log$' | head -n 3
nginx   238180 root    2w   REG              252,0      566 530210 /var/log/nginx/error.log
nginx   238180 root    3w   REG              252,0      566 530210 /var/log/nginx/error.log
nginx   238180 root    4w   REG              252,0     2408 530209 /var/log/nginx/access.log
```

La dernière commande répond à une question fréquente : dans quel fichier ce service écrit-il ses journaux ?

30.7 Des pannes, de bout en bout

Un service qui refuse de démarrer

Nginx est relancé, et refuse :

```
gary@serveur:~$ sudo systemctl start nginx
Job for nginx.service failed because the control process exited with error code.
See "systemctl status nginx.service" and "journalctl -xeu nginx.service" for details.
gary@serveur:~$ journalctl -u nginx -n 6 --no-pager
Oct 05 02:15:07 serveur nginx[238137]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98:
  Address already in use)
Oct 05 02:15:07 serveur nginx[238137]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98:
  Address already in use)
Oct 05 02:15:08 serveur nginx[238137]: nginx: [emerg] still could not bind()
[…]
```

Le message de systemd renvoie au journal, et le journal dit tout : le port 80 est déjà pris. Par qui ?

```
gary@serveur:~$ sudo ss -ltnp 'sport = :80'
State      Recv-Q     Send-Q         Local Address:Port           Peer
Address:Port     Process
LISTEN     0          5                    0.0.0.0:80                  0.0.0.0:*
  users:(("python3",pid=238131,fd=3))
gary@serveur:~$ ps -o user,pid,lstart,cmd -p $(pgrep -f 'http.server 80')
USER         PID                  STARTED CMD
root      238128 Mon Oct  5 02:15:02 2026 sudo nohup python3 -m http.server 80 --
bind 0.0.0.0
root      238130 Mon Oct  5 02:15:02 2026 sudo nohup python3 -m http.server 80 --
bind 0.0.0.0
root      238131 Mon Oct  5 02:15:02 2026 python3 -m http.server 80 --bind 0.0.0.0
```

Un petit serveur de test, lancé avec `sudo` cinq secondes avant le redémarrage de Nginx, et oublié. On l'arrête, et Nginx démarre :

```
gary@serveur:~$ sudo kill $(pgrep -f 'http.server 80')
gary@serveur:~$ sudo systemctl start nginx
gary@serveur:~$ systemctl is-active nginx
active
```

Pour un service qui refuse de démarrer, l'ordre est toujours le même : `systemctl status`, puis `journalctl -u service`, puis la vérification de la configuration quand le service en offre une : `nginx -t`, `sshd -t`, `apachectl configtest`. Neuf fois sur dix, la cause est écrite en clair dans le journal.

Un disque plein qui semble vide

Une application écrit un très gros journal sur `/srv/donnees`. Le disque est plein. Quelqu'un supprime le fichier, mais la place ne revient pas :

```
gary@serveur:~$ df -h /srv/donnees
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       5.9G  5.7G   82M  99% /srv/donnees
gary@serveur:~$ sudo rm /srv/donnees/journal-appli.log
gary@serveur:~$ df -h /srv/donnees
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       5.9G  5.7G   82M  99% /srv/donnees
gary@serveur:~$ sudo du -sh /srv/donnees
40K     /srv/donnees
```

`df` dit 5,7 Go occupés, `du` en trouve 40 Ko. L'écart a une explication classique : un fichier supprimé n'est effacé qu'une fois fermé par tous les programmes qui l'avaient ouvert. Son nom a disparu, ses données restent. `lsof +L1` liste ces fichiers, qui n'ont plus aucun nom :

```
gary@serveur:~$ sudo lsof +L1
COMMAND     PID        USER   FD   TYPE DEVICE   SIZE/OFF NLINK   NODE NAME
[…]
tail     238191        root    3r   REG    8,1 6081740800     0     13
  /srv/donnees/journal-appli.log (deleted)
```

La liste contient d'ordinaire plusieurs petits fichiers temporaires, que les programmes suppriment volontairement en les gardant ouverts. La ligne qui compte est celle de six gigaoctets. Le même processus empêche d'ailleurs de démonter le disque, et `fuser` le désigne aussi :

```
gary@serveur:~$ sudo umount /srv/donnees
umount: /srv/donnees: target is busy.
gary@serveur:~$ sudo fuser -vm /srv/donnees
```

```
                     USER        PID ACCESS COMMAND
/srv/donnees:        root     kernel mount /srv/donnees
                     root      238191 f.... tail
gary@serveur:~$ sudo kill 238191
gary@serveur:~$ df -h /srv/donnees
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       5.9G   48K  5.8G   1% /srv/donnees
```

Sur un vrai serveur, le processus qui garde le fichier est le service lui-même. Plutôt que de le tuer, on le redémarre, ou on lui demande de rouvrir ses journaux : c'est ce que fait `logrotate` avec `postrotate`. Pour vider un journal sans le supprimer, `sudo truncate -s 0 fichier` libère la place immédiatement, même s'il est ouvert.

Pour trouver ce qui remplit un disque, `sudo du -xh --max-depth=1 / | sort -h` donne la taille de chaque dossier, et `ncdu` permet de descendre dans l'arborescence.

Une fork bomb

`:(){ :|:& };:` est la plus célèbre ligne de shell malveillante. Elle définit une fonction nommée `:` qui s'appelle deux fois elle-même en arrière-plan, et la lance. Le nombre de processus double sans fin, jusqu'à saturer la machine. Ubuntu s'en protège en limitant le nombre de tâches de chaque utilisateur :

```
gary@serveur:~$ systemctl show user-1000.slice -p TasksMax
TasksMax=3641
```

Assez pour gêner, pas pour tout bloquer. Essayons-la, enfermée dans une limite de 60 tâches :

```
gary@serveur:~$ systemd-run --user --scope --unit=bombe -p TasksMax=60 \
    bash -c ':(){ :|:& };:' > /dev/null 2>&1 &
gary@serveur:~$ systemctl --user status bombe.scope --no-pager | head -n 6
● bombe.scope - /usr/bin/bash -c ":(){ :|:& };:"
     Loaded: loaded (/run/user/1000/systemd/transient/bombe.scope; transient)
  Transient: yes
     Active: active (running) since Mon 2026-10-05 02:15:16 UTC; 5s ago
      Tasks: 60 (limit: 60)
     Memory: 13.4M (peak: 16.2M)
gary@serveur:~$ uptime
 02:15:22 up  3:55,  2 users,  load average: 0.15, 0.07, 0.04
gary@serveur:~$ systemctl --user stop bombe.scope
```

La bombe a atteint sa limite et s'y heurte, sans conséquence pour le reste. Arrêter le groupe tue tous ses processus d'un coup, ce que `kill` aurait du mal à faire, la

bombe recréant des processus aussi vite qu'on les tue. C'est tout l'intérêt des groupes de contrôle vus au chapitre 12.

Une force brute sur SSH

Une machine du réseau, `neuf`, essaie de se connecter au serveur sous huit noms de comptes courants :

```
gary@neuf:~$ for u in admin root test oracle ubuntu deploy postgres git; do
>     ssh -o BatchMode=yes -o ConnectTimeout=3 $u@192.168.1.50 true 2>&1 | tail -n 1
> done
admin@192.168.1.50: Permission denied (publickey).
root@192.168.1.50: Permission denied (publickey).
test@192.168.1.50: Permission denied (publickey).
oracle@192.168.1.50: Permission denied (publickey).
ubuntu@192.168.1.50: Permission denied (publickey).
ssh: connect to host 192.168.1.50 port 22: Connection refused
ssh: connect to host 192.168.1.50 port 22: Connection refused
ssh: connect to host 192.168.1.50 port 22: Connection refused
```

Au sixième essai, le serveur ne répond plus. Que s'est-il passé de son côté ?

```
gary@serveur:~$ journalctl -u ssh --since "-10min" --no-pager | grep 'Invalid user' | \
    awk '{print $8, $10}' | sort | uniq -c
      1 admin 192.168.1.51
      1 oracle 192.168.1.51
      1 test 192.168.1.51
      1 ubuntu 192.168.1.51
gary@serveur:~$ sudo journalctl -k --since "-10min" --no-pager | grep -c 'UFW LIMIT BLOCK'
3
gary@serveur:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     10
|  `- Journal matches:  _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned:     2
   `- Banned IP list:   192.168.1.51
```

Les deux protections du chapitre 21 ont joué. La règle `limit` d'UFW a refusé les trois dernières connexions, arrivées trop vite. Fail2ban a compté les échecs et banni l'adresse. Le compte `root` n'apparaît pas parmi les comptes inconnus : il existe, et sa connexion est refusée par une autre règle. La chaîne `awk | sort | uniq -c` est le réflexe à garder pour résumer n'importe quel journal : qui, combien de fois.

Sur un serveur exposé à Internet, ces tentatives sont permanentes et ne sont pas un incident. Elles en deviennent un si une tentative réussit : cherchez alors `Accepted` dans le journal de SSH, et vérifiez chaque connexion acceptée.

```
gary@serveur:~$ sudo fail2ban-client set sshd unbanip 192.168.1.51
1
```

Un nom qui ne se résout plus

Cette panne-là n'a pas été provoquée. Pendant l'écriture de ce chapitre, une commande banale a échoué sur le serveur :

```
gary@serveur:~$ curl -sS -o /dev/null https://blog.maison.lan
curl: (6) Could not resolve host: blog.maison.lan
```

Le site répondait pourtant depuis le poste. Observons, du plus général au plus précis :

```
gary@serveur:~$ getent hosts blog.maison.lan; echo "code $?"
code 2
gary@serveur:~$ getent hosts ubuntu.com
2620:2d:4000:1::26 ubuntu.com
[…]
gary@serveur:~$ resolvectl status enp0s3 | grep -E 'Current DNS|DNS Servers|DNS Domain'
Current DNS Server: 1.1.1.1
       DNS Servers: 192.168.1.1 1.1.1.1
        DNS Domain: maison.lan
```

Les noms d'Internet se résolvent, pas les noms locaux. Et le serveur DNS en cours d'usage est 1.1.1.1, pas le routeur de la maison. Hypothèse : 1.1.1.1 ne connaît pas `maison.lan`. Testons-la en interrogeant chaque serveur directement :

```
gary@serveur:~$ dig +short @192.168.1.1 blog.maison.lan
192.168.1.50
gary@serveur:~$ dig @1.1.1.1 blog.maison.lan | grep status
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 58324
```

Confirmée. Reste à comprendre pourquoi systemd-resolved utilisait 1.1.1.1. La configuration du chapitre 18 déclare les deux serveurs côte à côte, et resolved les tient pour interchangeables. Une heure plus tôt, le serveur DNS local avait été relancé pour déclarer une nouvelle machine ; pendant ces quelques secondes, resolved était passé au second serveur, et il y était resté, sans rien écrire au journal. La correction est de ne déclarer que le serveur local, qui transmet lui-même les autres demandes vers Internet :

```
gary@serveur:~$ sudo sed -i \
    's/addresses: \[192.168.1.1, 1.1.1.1\]/addresses: [192.168.1.1]/' \
    /etc/netplan/60-adresse-fixe.yaml
gary@serveur:~$ sudo netplan apply
```

```
gary@serveur:~$ resolvectl status enp0s3 | grep -E 'Current DNS|DNS Servers'
       DNS Servers: 192.168.1.1
gary@serveur:~$ curl -s -o /dev/null -w '%{http_code}\n' https://blog.maison.lan
200
gary@serveur:~$ getent hosts ubuntu.com
2620:2d:4000:1::28 ubuntu.com
[…]
```

Le chapitre 18 porte désormais un avertissement. Le cheminement mérite d'être relu : un code d'erreur précis, la couche concernée, une hypothèse, un test qui la confirme, la cause, puis une correction qui la supprime au lieu d'en effacer l'effet. Redémarrer systemd-resolved aurait fait disparaître le symptôme jusqu'à la prochaine coupure.

30.8 Le retour d'expérience

Une fois l'incident clos, on l'écrit. Pas pour chercher un coupable : pour que la même panne ne revienne pas, ou soit réglée en cinq minutes la prochaine fois. Le format est court, et toujours le même. Voici celui de la panne précédente :

```
# Incident du 5 octobre 2026 : les noms locaux ne se résolvent plus sur serveur

## Résumé
De 01 h 38 à 02 h 20 (UTC), le serveur ne résolvait plus les noms en
maison.lan. Les sites restaient joignables depuis les postes, mais les
programmes du serveur qui appelaient blog.maison.lan échouaient.

## Impact
Aucun pour les visiteurs. Les vérifications lancées depuis le serveur
lui-même échouaient (curl : code 6).

## Chronologie
- 01:38  Le serveur DNS local (192.168.1.1) est relancé pour déclarer une
         nouvelle machine. Coupure de quelques secondes.
- 01:38  systemd-resolved, faute de réponse, passe au second serveur
         déclaré, 1.1.1.1. Il y reste ensuite. Rien n'est journalisé.
- 02:16  curl https://blog.maison.lan échoue sur le serveur (code 6).
- 02:18  resolvectl status : « Current DNS Server: 1.1.1.1 ».
         dig @192.168.1.1 répond, dig @1.1.1.1 répond NXDOMAIN.
- 02:20  Le second serveur est retiré de la configuration netplan,
         netplan apply. curl répond 200.

## Cause
La configuration réseau déclarait deux serveurs DNS de nature différente :
le serveur local, qui connaît maison.lan, et un serveur public, qui ne le
connaît pas. systemd-resolved les tient pour équivalents et garde celui
qui a répondu en dernier.

## Ce qui a bien fonctionné
```

```
- La démarche : le symptôme (code 6) a mené à la résolution, puis à
  resolvectl, sans détour.

## Actions
- [x] Ne déclarer que le serveur local, qui transmet lui-même vers Internet.
- [x] Avertissement ajouté au chapitre 18.
- [ ] Ajouter à la supervision une vérification d'un nom local depuis le
      serveur.
```

Partie	Contenu
Résumé	Trois lignes, lisibles par quelqu'un qui n'a rien suivi
Impact	Qui a été gêné, combien de temps
Chronologie	Les heures et les faits, tirés des notes prises pendant l'incident
Cause	La cause profonde, pas le dernier symptôme
Actions	Des tâches précises, chacune avec quelqu'un pour s'en charger

La dernière action, non cochée, est la plus importante : une vérification qui aurait signalé le problème au lieu d'attendre qu'on tombe dessus. C'est le lien avec la supervision du chapitre 26.

En résumé

- Observer, supposer, tester une seule chose, conclure. Noter tout, avec l'heure.
- Trier d'abord : l'impact, ce qui a changé (`dpkg.log`, etckeeper, `last`), le symptôme et la cause.
- `journalctl -p err -b`, `journalctl -b -1`, `dmesg -T --level=err,warn`, puis les journaux propres aux services.
- Un processus tué faute de mémoire laisse une ligne `Killed process` dans le journal du noyau. Son code de retour est 137.
- `ping` et `mtr` mesurent latence et pertes, `iperf3` le débit. `tc netem` simule un mauvais réseau.
- `strace` montre les appels système, `lsof` les fichiers et connexions ouverts.
- Un service qui ne démarre pas : `systemctl status`, `journalctl -u`, le test de configuration.
- `df` plein et `du` vide : un fichier supprimé mais ouvert, que `lsof +L1` désigne.

- Les groupes de contrôle contiennent une fork bomb. UFW et Fail2ban contiennent une force brute.
- Un incident clos s'écrit : chronologie, cause profonde, actions.

Atelier — Des pannes à résoudre

Objectif : retrouver et réparer des pannes inconnues sur une machine d'essai, en appliquant la méthode. Comptez une heure.

Le script `pannes.sh` introduit au hasard l'une de quatre pannes, et ne dit que le symptôme :

```
#!/bin/bash
#
# pannes.sh — introduit une panne au hasard, pour s'entraîner au diagnostic
# Usage : sudo pannes.sh            choisit et provoque une panne
#         sudo pannes.sh --reponse  dit laquelle (après avoir cherché !)
#         sudo pannes.sh --reparer  remet tout en état
# À n'utiliser que sur une machine d'essai.

set -euo pipefail
[ "$(id -u)" -eq 0 ] || { echo "À lancer avec sudo." >&2; exit 1; }

memo=/root/.panne
site=/var/www/blog/index.html

reparer() {
    systemctl stop intrus.service remplisseur.service 2>/dev/null || true
    systemctl reset-failed intrus.service remplisseur.service 2>/dev/null || true
    resolvectl revert enp0s3 2>/dev/null || true
    chmod 644 "$site"
    systemctl restart nginx
    rm -f "$memo"
    echo "Tout est remis en état."
}

case "${1:-}" in
    --reponse) cat "$memo" 2>/dev/null || echo "Aucune panne en cours."; exit 0 ;;
    --reparer) reparer; exit 0 ;;
esac

[ -f "$memo" ] && { echo "Une panne est déjà en cours. Trouvez-la, ou lancez --reparer."
>&2; exit 1; }

numero=${PANNE:-$(( RANDOM % 4 + 1 ))}   # PANNE=2 pour choisir
case $numero in
    1)  # Un autre programme prend le port 80, puis Nginx est relancé
        systemctl stop nginx
        systemd-run --quiet --unit=intrus \
            python3 -m http.server 80 --bind 127.0.0.1 --directory /var/www/html >/dev/null
        sleep 1
        systemctl start nginx 2>/dev/null || true
        reponse="Un programme (intrus.service) occupe le port 80 ; Nginx ne peut plus
démarrer."
```

```
        symptome="Le blog ne répond plus correctement depuis un redémarrage de Nginx." ;;
    2)  # Le serveur DNS de l'interface est remplacé par un serveur public
        resolvectl dns enp0s3 1.1.1.1
        reponse="Le DNS de enp0s3 a été remplacé par 1.1.1.1, qui ignore maison.lan."
        symptome="Le serveur ne trouve plus les machines du réseau local par leur nom." ;;
    3)  # Un processus remplit /srv/donnees avec un fichier qu'il a déjà supprimé
        systemd-run --quiet --unit=remplisseur bash -c \
            'fallocate -l 5800M /srv/donnees/.cache; exec 3</srv/donnees/.cache; rm /srv/
donnees/.cache; sleep infinity'
        reponse="remplisseur.service garde ouvert un fichier supprimé de 5,8 Go dans /srv/
donnees."
        symptome="/srv/donnees est plein, mais on n'y trouve presque rien." ;;
    4)  # La page d'accueil du blog devient illisible pour Nginx
        chmod 000 "$site"
        reponse="La page $site n'est plus lisible (droits 000)."
        symptome="Le blog répond, mais par une erreur." ;;
esac

echo "$reponse" > "$memo"
chmod 600 "$memo"
echo "Une panne a été introduite. Ce qu'on vous signale :"
echo "  « $symptome »"
```

Il suppose une machine organisée comme celle du livre : le site `blog.maison.lan` servi par Nginx depuis `/var/www/blog`, un disque monté sur `/srv/donnees`, l'interface `enp0s3`. Adaptez ces trois points à la vôtre.

Préparer

1. Prenez un instantané de la machine virtuelle.
2. Installez le script : `sudo install -m 755 pannes.sh /usr/local/sbin/`.
3. Ouvrez un fichier de notes.

Pour chaque panne

4. Lancez `sudo pannes.sh` et lisez le symptôme signalé.
5. Observez, puis notez une hypothèse et le test qui la vérifiera. Notez l'heure.
6. Testez. Si l'hypothèse tombe, notez pourquoi et passez à la suivante.
7. Une fois la cause trouvée, réparez-la vous-même, sans `--reparer`, et vérifiez que le symptôme a disparu.
8. Comparez avec `sudo pannes.sh --reponse`, puis lancez `sudo pannes.sh --reparer` pour repartir d'une machine saine.

Recommencez jusqu'à avoir rencontré les quatre pannes. La variable `PANNE` permet de choisir : `sudo PANNE=3 pannes.sh`.

Conclure

9. Choisissez la panne qui vous a pris le plus de temps, et rédigez son retour d'expérience au format du chapitre.

Vérifier

- Pour chaque panne, vos notes contiennent au moins une hypothèse écartée ou confirmée par un test.
- Après réparation, `curl -s -o /dev/null -w '%{http_code}\n' https://blog.maison.lan` affiche 200 depuis le serveur, et `df -h /srv/donnees` montre le disque presque vide.

Exercices

Exercice 1 — Symptôme ou cause

Classez chacune de ces phrases en symptôme ou en cause, et pour chaque symptôme proposez une cause possible et la commande qui la vérifierait.

1. « Le site affiche une erreur 502. »
2. « PostgreSQL a été tué par le noyau faute de mémoire. »
3. « Les sauvegardes de cette nuit n'ont pas été faites. »
4. « Le certificat a expiré hier. »
5. « La machine répond lentement en SSH. »

Exercice 2 — Lire une trace

```
Oct 12 03:14:07 web kernel: Out of memory: Killed process 1823 (mariadbd)
  total-vm:2541320kB, anon-rss:1236880kB, file-rss:0kB, shmem-rss:0kB, UID:110
  pgtables:2848kB oom_score_adj:0
```

Que s'est-il passé ? Combien de mémoire le processus occupait-il ? Était-ce une limite de groupe de contrôle ou la machine entière ? Quelles commandes lancez-vous ensuite, et quelles solutions envisagez-vous ?

Exercice 3 — strace

Avec strace, découvrez quels fichiers de configuration lisent `ssh` (le client) et `git` au démarrage. Lequel de ces fichiers changeriez-vous pour modifier leur comportement pour vous seul ?

Exercice 4 — Le disque qui se remplit

Écrivez la suite de commandes qui, sur une machine dont la racine est pleine à 100 %, trouve en moins de deux minutes le dossier responsable. Puis dites ce que vous feriez s'il s'agit de `/var/log/journal`, de `/var/lib/docker` ou de `/var/cache/apt`.

Exercice 5 — Un réseau dégradé

Sur votre machine d'essai, ajoutez 200 ms de retard avec `tc netem`. Mesurez le temps de chargement d'une page avec `curl -w '%{time_total}\n'`, avant et après. Combien d'allers-retours la page demande-t-elle ? Retirez la règle ensuite.

Pour aller plus loin

Sur la machine. `man strace` et `man lsof` sont longues mais riches d'exemples. `man tc-netem` décrit tout ce que le module sait simuler : retard variable, pertes en rafale, paquets dupliqués ou désordonnés. `systemd-analyze blame` dit quels services ralentissent le démarrage.

Paquets. `sysdig` et les outils `bpfcc-tools` (`opensnoop`, `execsnoop`, `tcpconnect`) observent toute la machine à la fois, avec beaucoup moins de ralentissement que strace. `atop` enregistre l'état de la machine toutes les dix minutes, pour examiner après coup ce qui consommait quoi au moment d'une panne.

En ligne. Brendan Gregg, ingénieur spécialiste des performances, publie sur `brendangregg.com` la méthode USE (utilisation, saturation, erreurs), une liste de contrôle pour chaque ressource de la machine, et des schémas qui situent chaque outil d'observation. Le livre *Site Reliability Engineering*, publié gratuitement par Google sur `sre.google/books`, consacre des chapitres entiers au diagnostic et aux retours d'expérience sans recherche de coupable.



CHAPITRE 31

Récupérer et sauver

Le serveur ne démarre plus. Le mot de passe de l'administrateur est perdu. Un disque renvoie des erreurs, un volume a été supprimé par erreur. Ce chapitre traite ces situations une par une, toutes provoquées sur le serveur du livre et réparées pour de bon. La première règle, avant chacune : ne rien aggraver.

DANS CE CHAPITRE

- Démarrer à la main depuis la ligne de commande de GRUB, puis régénérer sa configuration.
- Reprendre la main sur une machine dont on a perdu le mot de passe.
- Réparer le démarrage depuis un système de secours, avec chroot.
- Réparer un système de fichiers, même quand son superbloc est détruit.
- Tester un disque et en sauver le contenu malgré des secteurs illisibles.
- Récupérer des fichiers effacés, un volume LVM supprimé, un RAID dégradé.
- Savoir quand s'arrêter et demander de l'aide.

> **AVANT DE COMMENCER**
>
> Les chapitres 3, 11, 14, 15 et 16. Une clé USB d'installation d'Ubuntu 24.04, ou son image ISO pour une machine virtuelle. Pour les essais : une machine virtuelle et un instantané pris avant de commencer. Les outils de ce chapitre s'installent avec `sudo apt install testdisk gddrescue`.

31.1 Avant de toucher à quoi que ce soit

Une panne de démarrage ou de disque donne envie d'agir vite. C'est le moment où l'on perd des données qui étaient encore là. Quatre règles :

- **Copier avant de réparer.** Si les données comptent et que le disque est suspect, on le clone d'abord, et l'on travaille sur la copie.
- **Lire seulement.** Tant qu'on ne sait pas, on monte en lecture seule et l'on lance les outils dans leur mode qui ne modifie rien : `e2fsck -n`, `badblocks` sans option d'écriture.
- **Une chose à la fois**, comme au chapitre 30, en notant chaque commande.
- **Savoir ce que valent les sauvegardes.** Si celles du chapitre 16 sont récentes et vérifiées, réinstaller et restaurer est souvent plus rapide et plus sûr que réparer.

31.2 GRUB sans configuration

Le fichier `/boot/grub/grub.cfg` décrit à GRUB ce qu'il doit démarrer. Supprimons-le, puis redémarrons :

```
gary@serveur:~$ ls -l /boot/grub/grub.cfg
-rw------- 1 root root 7233 Oct  4 21:43 /boot/grub/grub.cfg
gary@serveur:~$ sudo rm /boot/grub/grub.cfg
gary@serveur:~$ sudo systemctl reboot
```

GRUB démarre, ne trouve pas sa configuration et s'arrête à sa ligne de commande :

```
                          GNU GRUB  version 2.12

   Minimal BASH-like line editing is supported. For the first word, TAB lists possible
   command completions. Anywhere else TAB lists possible device or file completions. To
   enable less(1)-like paging, "set pager=1".


grub>
```

On ne fait pas qu'y lire : on peut y démarrer le système à la main. `ls` liste les disques et les partitions que GRUB voit. GRUB compte les disques à partir de zéro, et les partitions à partir de un : `(hd1,gpt2)` est la deuxième partition du deuxième disque. On cherche celle qui contient le noyau, `/boot`, puis on lui donne le noyau, la racine, et l'image de démarrage :

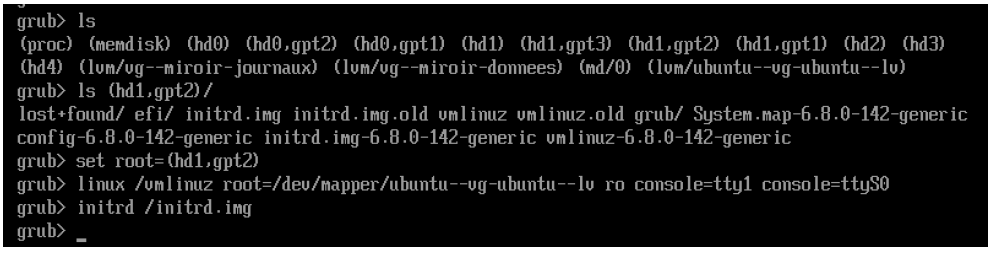


Figure 31.1 — Un démarrage à la main depuis la ligne de commande de GRUB.

Commande GRUB	Rôle
`ls`	Les disques, partitions, volumes LVM et grappes RAID que GRUB reconnaît
`ls (hd1,gpt2)/`	Le contenu d'une partition : on y cherche `vmlinuz` et `initrd.img`
`set root=(hd1,gpt2)`	La partition où GRUB lira les fichiers suivants
`linux /vmlinuz root=… ro`	Le noyau, avec l'emplacement de la racine du système
`initrd /initrd.img`	L'image qui monte la racine, LVM et chiffrement compris
`boot`	Démarrer

La racine se désigne comme dans `/etc/fstab`. Sur le serveur, c'est le volume LVM `/dev/mapper/ubuntu--vg-ubuntu--lv` ; sans LVM, ce serait `/dev/sda2` ou `UUID=…`. La touche Tab complète les noms, ce qui évite de deviner. Une fois le système démarré, on reconstruit la configuration :

```
gary@serveur:~$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
gary@serveur:~$ sudo update-grub
Sourcing file `/etc/default/grub'
Sourcing file `/etc/default/grub.d/50-cloudimg-settings.cfg'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.8.0-142-generic
Found initrd image: /boot/initrd.img-6.8.0-142-generic
[…]
```

```
done
gary@serveur:~$ ls -l /boot/grub/grub.cfg
-rw------- 1 root root 7233 Oct  5 02:35 /boot/grub/grub.cfg
```

`/proc/cmdline` confirme que le système a démarré avec la ligne tapée à la main. Au redémarrage suivant, GRUB retrouve son menu.

31.3 Le mot de passe perdu

Ubuntu n'a pas de mot de passe `root` : on administre par `sudo`. Le mot de passe perdu est donc celui de l'administrateur, et avec lui l'accès à `sudo`. Il se remplace à condition d'être devant la machine, ou devant sa console, celle d'une machine virtuelle ou celle que fournit l'hébergeur d'un serveur loué.

Afficher le menu de GRUB

Sur Ubuntu, le menu de GRUB est caché. Pour l'afficher, on appuie sur **Échap** pendant le démarrage, juste après l'écran du constructeur, sur une machine UEFI ; on garde **Maj** enfoncée sur une machine ancienne. Trop d'appuis sur Échap mènent à la ligne de commande de GRUB, vue plus haut : `normal` ramène alors au menu.

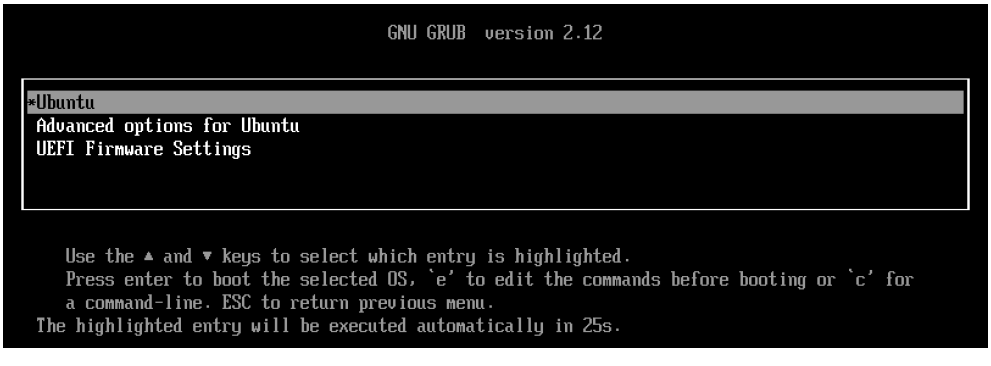


Figure 31.2 — Le menu de GRUB.

`Advanced options for Ubuntu` propose les noyaux précédents, utiles quand une mise à jour du noyau empêche de démarrer, et un mode de dépannage pour chacun. La touche `e` permet de modifier l'entrée choisie, pour ce démarrage seulement.

Démarrer sur un shell

Dans l'éditeur, on descend jusqu'à la ligne qui commence par `linux`, on va à sa fin (`Ctrl-e` ou Fin) et l'on ajoute `init=/bin/bash`. Au lieu de lancer systemd, le noyau lancera un shell, en `root`, sans rien demander.

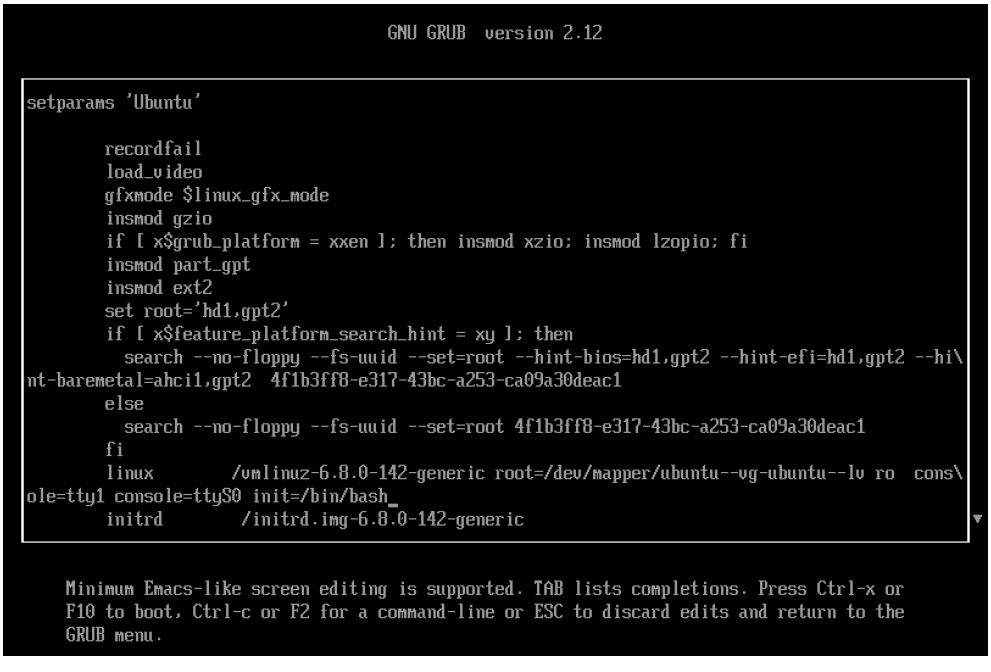


Figure 31.3 — L'entrée de démarrage en cours de modification, avec init=/bin/bash ajouté à la ligne linux.

`Ctrl-x` démarre. Quelques secondes plus tard :

```
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@(none):/# mount | grep ' / '
/dev/mapper/ubuntu--vg-ubuntu--lv on / type ext4 (ro,relatime)
```

Un shell `root`, sans nom de machine ni services : rien d'autre ne tourne. La racine est montée en lecture seule. On la remonte en écriture, on change le mot de passe, puis l'on repasse en lecture seule avant de redémarrer :

```
root@(none):/# mount -o remount,rw /
root@(none):/# passwd gary
New password:
Retype new password:
passwd: password updated successfully
```

```
root@(none):/# sync
root@(none):/# mount -o remount,ro /
root@(none):/# reboot -f
```

`reboot -f` est nécessaire : sans systemd, `reboot` ne trouverait personne pour arrêter proprement les services. Après le redémarrage, la session s'ouvre avec le nouveau mot de passe.

> **ATTENTION · Qui a la console a la machine**
>
> Cette manipulation ne demande aucun mot de passe. Toute personne qui peut redémarrer la machine devant son écran peut en faire autant. Pour un serveur dans un lieu accessible, ou un portable, deux protections : un mot de passe sur GRUB, qui interdit de modifier les entrées (`grub-mkpasswd-pbkdf2`), et surtout le chiffrement du disque du chapitre 15, sans lequel n'importe quel système de secours lit aussi tout le contenu.

31.4 Démarrer sur un système de secours

Quand GRUB lui-même a disparu, il faut démarrer sur autre chose. La clé d'installation d'Ubuntu sert de système de secours. Ubuntu Desktop propose « Essayer Ubuntu », qui donne un bureau et un terminal. Ubuntu Server propose, dans le menu « Help » de l'installateur, « Enter shell ». Dans les deux cas, on obtient un système complet, qui ne touche aux disques que si on le lui demande.

La panne

Effaçons le contenu de la partition EFI, où se trouvent le chargeur de démarrage et GRUB :

```
gary@serveur:~$ sudo ls -R /boot/efi/EFI
/boot/efi/EFI:
BOOT  ubuntu

/boot/efi/EFI/BOOT:
BOOTX64.EFI  fbx64.efi  grubx64.efi  mmx64.efi

/boot/efi/EFI/ubuntu:
BOOTX64.CSV  grub.cfg  grubx64.efi  mmx64.efi  shimx64.efi
gary@serveur:~$ sudo rm -r /boot/efi/EFI
gary@serveur:~$ sudo systemctl reboot
```

Le micrologiciel ne trouve plus rien à démarrer sur le disque, et essaie le réseau :

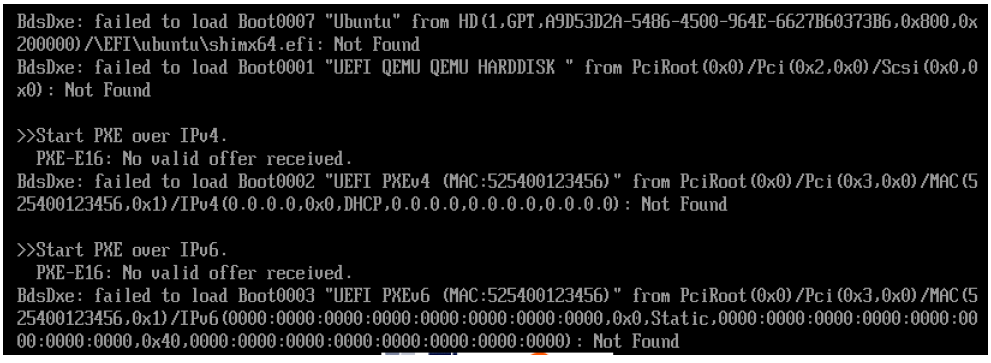


Figure 31.4 — Le micrologiciel ne trouve plus le chargeur de démarrage, puis tente un démarrage par le réseau.

`Not Found` sur le fichier `shimx64.efi` : l'entrée « Ubuntu » existe toujours dans la mémoire du micrologiciel, mais le fichier qu'elle désigne a disparu. Les tentatives `PXE` sont le démarrage par le réseau, qui échoue faute de serveur pour y répondre.

Retrouver les disques

On démarre sur la clé, et l'on ouvre un terminal. Le serveur du livre a de quoi compliquer la tâche : sa racine est dans LVM, ses données en RAID et dans un volume chiffré, tout ce qu'a construit le chapitre 15.

```
ubuntu@ubuntu:~$ lsblk -o NAME,SIZE,TYPE,FSTYPE
NAME                      SIZE TYPE  FSTYPE
sda                        10G disk
├─sda1                      6G part  ext4
└─sda2                      4G part  xfs
sdb                        25G disk
├─sdb1                      1G part  vfat
├─sdb2                      2G part  ext4
└─sdb3                     22G part  LVM2_member
  └─ubuntu--vg-ubuntu--lv  16G lvm   ext4
sdc                         5G disk  linux_raid_member
└─md127                     5G raid1 LVM2_member
[…]
sde                         5G disk  crypto_LUKS
[…]
```

La colonne `FSTYPE` dit tout : `sdb1` est la partition EFI, `sdb2` est `/boot`, `sdb3` porte la racine dans LVM. Le système de secours a déjà reconnu la grappe RAID, qu'il a nommée `md127`, et activé les volumes LVM. On le vérifie, et l'on ouvre le volume chiffré :

```
ubuntu@ubuntu:~$ sudo mdadm --assemble --scan
mdadm: No arrays found in config file or automatically
ubuntu@ubuntu:~$ cat /proc/mdstat
Personalities : [raid1] [raid0] [raid6] [raid5] [raid4] [raid10]
md127 : active (auto-read-only) raid1 sdd[1] sdc[2]
      5237760 blocks super 1.2 [2/2] [UU]
ubuntu@ubuntu:~$ sudo vgchange -ay
  2 logical volume(s) in volume group "vg-miroir" now active
  1 logical volume(s) in volume group "ubuntu-vg" now active
ubuntu@ubuntu:~$ sudo cryptsetup open /dev/sde coffre
Enter passphrase for /dev/sde:
```

`mdadm --assemble --scan` ne trouve rien à assembler parce que c'est déjà fait : un système récent reconnaît les grappes d'office. Sur un système plus ancien, ou si une grappe manque, cette commande les assemble. Le nom change d'un système à l'autre (`md0` sur le serveur, `md127` ici) : c'est sans importance, LVM et `/etc/fstab` désignent leurs volumes par nom ou par identifiant.

Le chroot

Avant de réparer, on peut vérifier le système de fichiers de la racine, puisqu'il n'est pas monté :

```
ubuntu@ubuntu:~$ sudo e2fsck -f /dev/ubuntu-vg/ubuntu-lv
e2fsck 1.47.0 (5-Feb-2023)
Pass 1: Checking inodes, blocks, and sizes
[…]
Pass 5: Checking group summary information
/dev/ubuntu-vg/ubuntu-lv: 148105/1048576 files (0.3% non-contiguous), 2284095/4194304 blocks
```

Puis on reconstitue l'arborescence du serveur sous `/mnt`, avec les systèmes de fichiers virtuels dont les outils ont besoin, et l'on y entre avec `chroot` : à partir de là, les commandes s'exécutent dans le système du serveur, avec ses programmes et ses fichiers de configuration.

```
ubuntu@ubuntu:~$ sudo mount /dev/ubuntu-vg/ubuntu-lv /mnt
ubuntu@ubuntu:~$ sudo mount /dev/sdb2 /mnt/boot
ubuntu@ubuntu:~$ sudo mount /dev/sdb1 /mnt/boot/efi
ubuntu@ubuntu:~$ for d in /dev /dev/pts /proc /sys /run; do
>     sudo mount --bind $d /mnt$d
> done
ubuntu@ubuntu:~$ sudo chroot /mnt
root@ubuntu:/# cat /etc/hostname
serveur
root@ubuntu:/# mount -t efivarfs efivarfs /sys/firmware/efi/efivars
root@ubuntu:/# grub-install --target=x86_64-efi --efi-directory=/boot/efi \
>     --bootloader-id=ubuntu
```

```
Installing for x86_64-efi platform.
Installation finished. No error reported.
root@ubuntu:/# update-grub
[…]
root@ubuntu:/# ls -R /boot/efi/EFI
/boot/efi/EFI:
BOOT  ubuntu

/boot/efi/EFI/BOOT:
BOOTX64.EFI  fbx64.efi  mmx64.efi

/boot/efi/EFI/ubuntu:
BOOTX64.CSV  grub.cfg  grubx64.efi  mmx64.efi  shimx64.efi
root@ubuntu:/# umount /sys/firmware/efi/efivars
root@ubuntu:/# exit
```

L'ordre des montages compte : la racine d'abord, puis `/boot`, puis `/boot/efi` à l'intérieur. `efivarfs` donne à `grub-install` l'accès aux variables du micrologiciel, pour y enregistrer l'entrée de démarrage. Ubuntu réinstalle aussi `shimx64.efi`, le petit chargeur signé qui permet de démarrer avec le *Secure Boot* activé.

On démonte tout, dans l'ordre inverse, et l'on ferme ce qu'on a ouvert :

```
ubuntu@ubuntu:~$ for d in /run /sys /proc /dev/pts /dev; do sudo umount /mnt$d; done
ubuntu@ubuntu:~$ sudo umount /mnt/boot/efi /mnt/boot /mnt
ubuntu@ubuntu:~$ sudo cryptsetup close coffre
ubuntu@ubuntu:~$ sudo vgchange -an ubuntu-vg vg-miroir
```

On retire la clé, on redémarre : le micrologiciel retrouve son fichier, GRUB son menu, le serveur démarre.

SUR LE TERRAIN · Le démontage qui a coupé la branche

Beaucoup de guides montent les systèmes virtuels d'un coup, avec `mount --rbind`, et démontent avec `umount -lR /mnt`. Pendant les essais de ce chapitre, cette combinaison a retiré au système de secours lui-même ses terminaux : juste après, `sudo` répondait `unable to allocate pty: No such device`, et il a fallu l'éteindre de force. Sur Ubuntu, les montages sont partagés : démonter la copie de `/dev/pts` démonte aussi l'original. La méthode ci-dessus, un `--bind` simple par dossier, démonté un par un, n'a pas cet effet. Si vous utilisez `--rbind`, ajoutez `--make-rslave` à chaque montage.

31.5 Un système de fichiers abîmé

Une coupure de courant, un disque qui flanche, et un système de fichiers peut devenir incohérent. Au démarrage, Ubuntu le vérifie et le répare seul quand c'est simple. Sinon, le démarrage s'arrête en mode de secours, et c'est à vous.

`fsck` ne se lance jamais sur un système de fichiers monté en écriture : il le détruirait. Pour la racine, on passe par le système de secours, comme ci-dessus. Pour une partition de données, il suffit de la démonter.

Essayons le cas le plus impressionnant sur une image jetable, pour ne rien risquer : le **superbloc**, le bloc qui décrit tout le système de fichiers, est effacé.

```
gary@serveur:~/essais$ truncate -s 300M disque.img
gary@serveur:~/essais$ mkfs.ext4 -q -L essai disque.img
gary@serveur:~/essais$ sudo mount -o loop disque.img /mnt/essai
gary@serveur:~/essais$ sudo cp -r /etc/ssh /mnt/essai/
gary@serveur:~/essais$ sudo umount /mnt/essai
gary@serveur:~/essais$ dd if=/dev/zero of=disque.img bs=1024 seek=1 count=1 \
    conv=notrunc status=none
gary@serveur:~/essais$ sudo mount -o loop disque.img /mnt/essai
mount: /mnt/essai: wrong fs type, bad option, bad superblock on /dev/loop0,
  missing codepage or helper program, or other error.
```

Le superbloc d'ext4 se trouve au premier kilo-octet ; `dd` vient de le remplacer par des zéros. Le système de fichiers n'est plus reconnu. Mais ext4 garde des copies du superbloc, et `mke2fs -n` dit où, sans rien écrire :

```
gary@serveur:~/essais$ mke2fs -n disque.img | tail -n 3
mke2fs 1.47.0 (5-Feb-2023)
Superblock backups stored on blocks:
        32768
```

> **DANGER · Le `-n`**
>
> Sans l'option `-n`, `mke2fs` formaterait le disque et effacerait tout. Vérifiez deux fois avant d'appuyer sur Entrée. Pour un système de fichiers déjà formaté, `sudo dumpe2fs /dev/sdX1 | grep -i superblock` donne la même liste sans ce risque, tant que le superbloc principal est lisible.

`e2fsck -n` regarde sans rien corriger, et dit l'ampleur des dégâts. Puis on répare en lui indiquant la copie, et en acceptant les corrections avec `-y` :

```
gary@serveur:~/essais$ e2fsck -fy -b 32768 disque.img
e2fsck 1.47.0 (5-Feb-2023)
Pass 1: Checking inodes, blocks, and sizes
[…]
Free blocks count wrong for group #0 (27917, counted=27914).
Fix? yes
[…]
essai: ***** FILE SYSTEM WAS MODIFIED *****
essai: 28/76800 files (0.0% non-contiguous), 9154/76800 blocks
gary@serveur:~/essais$ sudo mount -o loop disque.img /mnt/essai
gary@serveur:~/essais$ ls /mnt/essai/ssh | head -n 3
moduli
ssh_config
ssh_config.d
```

Les compteurs de la copie dataient de la création : `e2fsck` les recalcule. Les fichiers sont intacts. Sur un vrai disque, les fichiers dont la place n'a pas pu être retrouvée atterrissent dans `lost+found`, sous un numéro au lieu de leur nom.

Système de fichiers	Vérification	Remarque
ext4	`e2fsck -f`	`-n` pour regarder, `-y` pour tout accepter, `-b` pour un superbloc de secours
XFS	`xfs_repair`	`-n` pour regarder. Il refuse de travailler si le journal n'est pas vide : monter puis démonter le vide
Btrfs	`btrfs check`	Réparation (`--repair`) en dernier recours, après une copie
vfat (EFI)	`fsck.vfat -a`	

31.6 Un disque qui flanche

Tester

Un disque fatigué renvoie des erreurs de lecture sur certains secteurs. Le noyau les note dans son journal, et `smartctl -a /dev/sda`, du paquet `smartmontools`, interroge l'état que le disque tient sur lui-même : secteurs réalloués, secteurs en attente, heures de fonctionnement.

Pour l'essai, simulons un disque abîmé. Le device-mapper du noyau, qu'utilisent LVM et le chiffrement, sait aussi présenter une zone qui répond par une erreur à toute lecture. On crée une image de 64 Mo dont seize secteurs, au milieu, sont illisibles :

```
gary@serveur:~/essais$ truncate -s 64M fatigue.img
gary@serveur:~/essais$ LOOP=$(sudo losetup --show -f fatigue.img); echo $LOOP
/dev/loop0
gary@serveur:~/essais$ printf '0 60000 linear %s 0\n60000 16 error\n%s\n' $LOOP \
    "60016 71056 linear $LOOP 60016" | sudo dmsetup create fatigue
```

`badblocks` lit tout le disque, sans rien écrire, et liste les secteurs qui ne répondent pas :

```
gary@serveur:~/essais$ sudo badblocks -sv -b 512 /dev/mapper/fatigue
Checking blocks 0 to 131071
Checking for bad blocks (read-only test): 60000
60001
[…]
60015
done
Pass completed, 16 bad blocks found. (16/0/0 errors)
gary@serveur:~/essais$ sudo dd if=/dev/mapper/fatigue of=/dev/null bs=1M status=none; \
    echo "code $?"
dd: error reading '/dev/mapper/fatigue': Input/output error
code 1
gary@serveur:~/essais$ sudo journalctl -k -n 1 --no-pager
Oct 05 08:19:50 serveur kernel: Buffer I/O error on dev dm-4, logical block 7501,
  async page read
```

Les seize secteurs, exactement. Et `dd`, qui sert à copier un disque, s'arrête à la première erreur. Les disques modernes remplacent d'eux-mêmes un secteur défaillant par un secteur de réserve ; quand ces erreurs apparaissent, la réserve s'épuise. Un disque qui commence à perdre des secteurs ne guérit pas : on sauve son contenu, et on le remplace.

Sauver avec ddrescue

`ddrescue`, du paquet `gddrescue`, copie un disque en sautant ce qu'il ne peut pas lire, puis revient sur les zones difficiles. Il tient une **carte** de ce qui a été sauvé, ce qui permet d'interrompre et de reprendre :

```
gary@serveur:~/essais$ sudo ddrescue -n /dev/mapper/fatigue copie.img copie.map
GNU ddrescue 1.27
Press Ctrl-C to interrupt
[…]
pct rescued:   99.98%, read errors:        3,  remaining time:          1s
Finished
gary@serveur:~/essais$ sudo ddrescue -r3 /dev/mapper/fatigue copie.img copie.map | tail -n 3
pct rescued:   99.98%, read errors:       62,  remaining time:         n/a
                            time since last successful read:         n/a
Finished
gary@serveur:~/essais$ grep -v '^#' copie.map
0x01D4DE00     +               3
```

```
0x00000000  0x01D4C000  +
0x01D4C000  0x00002000  -
0x01D4E000  0x022B2000  +
```

Le premier passage, `-n`, copie tout ce qui se lit facilement, sans s'acharner. Le second, `-r3`, réessaie trois fois les zones en échec. La carte se lit simplement : de 0 à `0x01D4C000`, sauvé (`+`) ; puis `0x2000` octets, soit 8 Ko, perdus (`-`) ; puis le reste, sauvé. Les seize secteurs de 512 octets, et rien d'autre.

On travaille ensuite sur `copie.img` : `e2fsck` dessus, puis montage en lecture seule, comme au paragraphe précédent. Sur un vrai disque, la copie se fait vers un autre disque au moins aussi grand, et sur une machine qui n'est pas celle qu'on sauve.

31.7 Des fichiers effacés

Un fichier supprimé n'est pas effacé tout de suite : sa place est marquée libre, et ses données restent jusqu'à ce qu'autre chose les recouvre. La première règle est donc d'arrêter d'écrire sur ce disque, et de le démonter.

PhotoRec, livré avec TestDisk, cherche dans tout le disque les signatures des types de fichiers qu'il connaît : PDF, images, archives, documents bureautiques, plus de quatre cents formats. Faisons l'essai sur une image, avec quelques fichiers compressés :

```
gary@serveur:~/essais$ sudo mount -o loop cle.img /mnt/essai
gary@serveur:~/essais$ sudo cp /usr/share/doc/bash/*.gz /mnt/essai/
gary@serveur:~/essais$ ls /mnt/essai
COMPAT.gz  NEWS.gz   README.Debian.gz    README.gz                    changelog.Debian.gz
INTRO.gz   POSIX.gz  README.commands.gz  README.md.bash_completion.gz  lost+found
gary@serveur:~/essais$ sync
gary@serveur:~/essais$ sudo rm /mnt/essai/*.gz
gary@serveur:~/essais$ sudo umount /mnt/essai
gary@serveur:~/essais$ sudo photorec /log /d recup /cmd cle.img \
    partition_none,fileopt,everything,disable,gz,enable,wholespace,search > /dev/null
gary@serveur:~/essais$ ls recup.1
f0196608.gz  f0196640.gz  f0196728.gz  f0196744.gz  f0196768.gz
f0196632.gz  f0196712.gz  f0196736.gz  f0196752.gz  report.xml
gary@serveur:~/essais$ zcat recup.1/f0196608.gz | head -n 2
Compatibility with previous versions
====================================
```

Les neuf fichiers sont revenus, mais sans leurs noms : PhotoRec ne lit pas les répertoires, il reconnaît des contenus. Il faut ensuite les ouvrir pour savoir lequel est lequel. Lancé sans `/cmd`, `photorec` présente des menus qui guident pas à pas : choix du disque, de la partition, des types de fichiers, du dossier où écrire. Ce dossier doit être sur un autre disque que celui qu'on fouille.

> **SUR LE TERRAIN · Le fichier qui n'avait jamais existé**
>
> Un premier essai n'a rien retrouvé : les fichiers avaient été copiés puis supprimés aussitôt, sans `sync`. ext4 retarde l'écriture des données, et ces fichiers n'avaient jamais atteint le disque. Il n'y avait rien à récupérer. À l'inverse, un fichier enregistré depuis longtemps puis effacé a de bonnes chances de revenir, si rien n'a été écrit par-dessus.

`extundelete` essaie de retrouver aussi les noms, à partir du journal d'ext4. Sur la même image, il a restauré cinq fichiers sur neuf lors d'un premier essai, sans leurs noms, et aucun lors du second. ext4 efface les informations de placement des fichiers supprimés, ce qui limite beaucoup ce genre d'outil. Comptez sur PhotoRec.

TestDisk, l'autre outil du paquet, retrouve des partitions perdues : une table de partitions effacée ou réécrite par erreur. Il parcourt le disque à la recherche des débuts de systèmes de fichiers, propose une table reconstruite, et ne l'écrit qu'après confirmation. Lancé par `sudo testdisk /dev/sdX`, il se pilote par menus.

31.8 Un volume LVM supprimé

LVM garde une copie de ses métadonnées avant chaque modification, dans `/etc/lvm/archive`. Un volume supprimé par erreur peut ainsi revenir. Créons-en un, écrivons-y, supprimons-le :

```
gary@serveur:~$ sudo lvcreate -q -L 200M -n essai ubuntu-vg
  Logical volume "essai" created.
gary@serveur:~$ sudo mkfs.ext4 -q /dev/ubuntu-vg/essai
gary@serveur:~$ sudo mount /dev/ubuntu-vg/essai /mnt/essai
gary@serveur:~$ echo "à ne pas perdre" | sudo tee /mnt/essai/important.txt
à ne pas perdre
gary@serveur:~$ sudo umount /mnt/essai
gary@serveur:~$ sudo lvremove -y ubuntu-vg/essai
  Logical volume "essai" successfully removed.
```

L'archive la plus récente a été écrite juste avant la suppression, et le dit :

```
gary@serveur:~$ ARCHIVE=$(sudo ls -t /etc/lvm/archive/ | grep ubuntu-vg | head -n 1)
gary@serveur:~$ echo $ARCHIVE
ubuntu-vg_00005-1111839040.vg
gary@serveur:~$ sudo grep description /etc/lvm/archive/$ARCHIVE
description = "Created *before* executing 'lvremove -y ubuntu-vg/essai'"
gary@serveur:~$ sudo vgcfgrestore -f /etc/lvm/archive/$ARCHIVE ubuntu-vg
  Volume group ubuntu-vg has active volume: ubuntu-lv.
  WARNING: Found 1 active volume(s) in volume group "ubuntu-vg".
  Restoring VG with active LVs, may cause mismatch with its metadata.
```

```
Do you really want to proceed with restore of volume group "ubuntu-vg",
  while 1 volume(s) are active? [y/n]: y
  Restored volume group ubuntu-vg.
gary@serveur:~$ sudo lvchange -ay ubuntu-vg/essai
gary@serveur:~$ sudo mount /dev/ubuntu-vg/essai /mnt/essai
mount: /mnt/essai: wrong fs type, bad option, bad superblock on
  /dev/mapper/ubuntu--vg-essai, […]
```

Le volume est revenu, à sa place. Son contenu, non.

SUR LE TERRAIN · Ubuntu efface vraiment

```
gary@serveur:~$ sudo lvmconfig devices/issue_discards
issue_discards=1
```

Ubuntu règle LVM pour qu'il signale au disque, à chaque suppression de volume, que les blocs libérés ne servent plus. C'est utile à un SSD, qui garde ainsi ses performances. Mais un SSD, ou un disque virtuel, qui reçoit ce signal efface réellement les blocs : `vgcfgrestore` rend la description du volume, plus ses données. Sur ce serveur virtuel, la même manipulation refaite avec `lvremove --config 'devices/issue_discards=0'` a rendu le fichier intact :

```
gary@serveur:~$ cat /mnt/essai/important.txt
à ne pas perdre
```

Sur un disque qui accepte ce signal, une suppression de volume est donc définitive. Seule la sauvegarde la répare.

`vgcfgrestore` reste utile dans les autres cas : des métadonnées LVM abîmées, un volume réduit ou étendu par erreur, un disque classique sans discard. Quand c'est un disque entier qui manque à un groupe de volumes, `vgchange -ay --activationmode partial` active ce qui peut l'être, pour en copier le contenu.

31.9 Un RAID dégradé

Le miroir du chapitre 15 est fait pour survivre à la perte d'un disque. Simulons la panne de l'un des deux :

```
gary@serveur:~$ sudo mdadm /dev/md0 --fail /dev/sdd
mdadm: set /dev/sdd faulty in /dev/md0
gary@serveur:~$ cat /proc/mdstat
Personalities : [raid1] [raid0] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdc[2] sdd[1](F)
      5237760 blocks super 1.2 [2/1] [U_]
gary@serveur:~$ sudo mdadm --detail /dev/md0 | grep -E 'State :|faulty|active sync'
```

```
             State : clean, degraded
       2       8       32        0      active sync   /dev/sdc
       1       8       48        -      faulty   /dev/sdd
gary@serveur:~$ df -h /srv/miroir/donnees | tail -n 1
/dev/mapper/vg--miroir-donnees  2.9G   33M  2.7G   2% /srv/miroir/donnees
```

`[2/1] [U_]` : deux disques prévus, un seul en service. `(F)` marque le disque en panne. Les données restent accessibles, servies par le disque restant. Le serveur fonctionne, mais sans filet. On retire le disque défaillant, on le remplace physiquement, puis on ajoute le nouveau ; ici, on remet le même :

```
gary@serveur:~$ sudo mdadm /dev/md0 --remove /dev/sdd
mdadm: hot removed /dev/sdd from /dev/md0
gary@serveur:~$ sudo mdadm /dev/md0 --add /dev/sdd
mdadm: added /dev/sdd
gary@serveur:~$ cat /proc/mdstat
[…]
md0 : active raid1 sdd[3] sdc[2]
      5237760 blocks super 1.2 [2/1] [U_]
      [>....................]  recovery =  3.8% (201856/5237760) finish=0.4min
  speed=201856K/sec
```

La reconstruction recopie tout le disque restant sur le nouveau. Une demi-minute ici ; des heures pour un disque de plusieurs téraoctets, pendant lesquelles le disque restant travaille à plein. C'est le moment le plus risqué de la vie d'un RAID : si le second disque lâche pendant la reconstruction, tout est perdu. D'où, encore, les sauvegardes.

```
gary@serveur:~$ cat /proc/mdstat
[…]
md0 : active raid1 sdd[3] sdc[2]
      5237760 blocks super 1.2 [2/2] [UU]
```

Un RAID dégradé sans que personne ne le sache est un piège : il ne protège plus rien. Le service `mdmonitor`, installé avec `mdadm`, envoie un courriel à l'adresse réglée par `MAILADDR` dans `/etc/mdadm/mdadm.conf`. La supervision du chapitre 26 peut aussi le surveiller : Node Exporter publie l'état des grappes.

31.10 Quand appeler à l'aide

Certaines situations dépassent ce qu'un administrateur doit tenter seul.

- **Un disque qui fait du bruit, chauffe, ou disparaît par intermittence** : chaque mise sous tension l'abîme davantage. Si les données n'ont pas de sauvegarde,

éteignez-le et confiez-le à une société de récupération de données. Elles ouvrent les disques en salle blanche ; c'est cher, et souvent efficace.

- **Un RAID à plusieurs disques en échec** : ne tentez pas `mdadm --create` sur les disques existants, une erreur d'ordre détruit tout. Clonez chaque disque avec `ddrescue`, et travaillez sur les copies.
- **Une compromission** : si vous pensez que la machine a été piratée, ne réparez pas. Isolez-la du réseau, gardez une image de ses disques pour l'enquête, et reconstruisez à partir d'une installation propre et des sauvegardes.

Pour le reste, les forums d'Ubuntu (`discourse.ubuntu.com`), le site `askubuntu.com` et les listes des projets répondent vite, à condition de poser la question avec les faits : la version, le message exact, ce qui a été essayé. Pour un serveur d'entreprise, le support commercial de Canonical, Ubuntu Pro, ou celui de votre hébergeur donne un interlocuteur engagé sur des délais.

En résumé

- Avant de réparer : copier si c'est précieux, lire seulement tant qu'on ne sait pas, une chose à la fois.
- À l'invite `grub>` : `ls`, `set root`, `linux … root=…`, `initrd`, `boot`, puis `update-grub` une fois démarré.
- Mot de passe perdu : Échap au démarrage, `e`, `init=/bin/bash`, `mount -o remount,rw /`, `passwd`, `reboot -f`. Le chiffrement du disque protège de qui a la console.
- Depuis un système de secours : `lsblk`, `mdadm --assemble --scan`, `vgchange -ay`, `cryptsetup open`, montages dans l'ordre, `--bind` un par un, `chroot`, `grub-install`, `update-grub`, puis tout démonter dans l'ordre inverse.
- `e2fsck` jamais sur un système monté en écriture. Un superbloc détruit se répare avec `-b` et l'une des copies.
- `badblocks` teste en lecture, `ddrescue` sauve un disque malade et tient la carte de ce qu'il a sauvé.
- PhotoRec retrouve les fichiers effacés par leur contenu, sans leurs noms. Arrêter d'écrire sur le disque d'abord.
- `vgcfgrestore` rend un volume LVM, mais pas ses données si le disque a reçu un discard, ce qu'Ubuntu fait par défaut.

- Un RAID dégradé fonctionne sans filet : `--fail`, `--remove`, `--add`, et une surveillance pour le savoir.

Atelier — Trois pannes

Objectif : provoquer trois pannes sur une machine virtuelle, et les réparer sans réinstaller. Comptez une heure et demie.

Préparer

1. Prenez un instantané de la machine virtuelle. Téléchargez l'image ISO d'Ubuntu 24.04 et ajoutez-la comme lecteur optique de la machine.
2. Notez la disposition des disques : `lsblk -f`, `cat /etc/fstab`, `cat /proc/cmdline`. Gardez ces notes hors de la machine.

Panne 1 — GRUB sans configuration

3. Supprimez `/boot/grub/grub.cfg` et redémarrez.
4. À l'invite `grub>`, retrouvez la partition qui contient le noyau, et démarrez à la main.
5. Régénérez la configuration et vérifiez que le menu revient.

Panne 2 — Le mot de passe perdu

6. Changez votre mot de passe pour une valeur aléatoire que vous ne notez pas : `echo "$USER:$(openssl rand -hex 16)" | sudo chpasswd`. Vérifiez qu'un `sudo -k; sudo true` échoue désormais.
7. Redémarrez, affichez le menu de GRUB, et retrouvez l'accès par `init=/bin/bash`.

Panne 3 — Un système de fichiers détruit

8. Créez une image de 300 Mo, formatez-la en ext4, copiez-y des fichiers, démontez-la.
9. Effacez son superbloc principal avec `dd`, et constatez que le montage échoue.
10. Retrouvez une copie du superbloc, réparez avec `e2fsck`, vérifiez vos fichiers.

Pour aller plus loin

11. Démarrez la machine virtuelle sur l'image ISO, choisissez l'essai ou le shell, et entrez dans votre système par `chroot`. Lancez `update-grub` depuis le chroot, puis démontez proprement.

Vérifier

- Après chaque panne, la machine démarre normalement, et `systemctl --failed` ne liste rien.
- Vous avez noté, pour chaque panne, les commandes qui l'ont réparée.

Exercices

Exercice 1 — L'invite de GRUB

Sur une machine sans LVM, la racine est sur `/dev/sda2` et `/boot` n'a pas de partition séparée. Écrivez les commandes à taper à l'invite `grub>` pour démarrer. Où se trouvent alors `vmlinuz` et `initrd.img` ?

Exercice 2 — Lire lsblk

Depuis un système de secours, `lsblk -f` affiche une partition `vfat` de 512 Mo, une partition `crypto_LUKS` de 237 Go, et rien d'autre. Que contient probablement chaque partition ? Quelles commandes lancez-vous, et dans quel ordre, pour atteindre les fichiers de l'utilisateur ?

Exercice 3 — La carte de ddrescue

Voici la carte laissée par `ddrescue` après une copie interrompue. Que reste-t-il à faire ? Que se passe-t-il si l'on relance la même commande ?

```
0x00000000  0x40000000  +
0x40000000  0x00010000  -
0x40010000  0x1BFFF0000  ?
```

Exercice 4 — Le discard

Sur votre machine, vérifiez la valeur de `issue_discards` et dites si vos disques acceptent le discard (`lsblk -D`). Créez un volume LVM, écrivez-y un fichier, supprimez-le et tentez la restauration. Qu'en concluez-vous pour vos volumes réels ?

Exercice 5 — Le plan de secours

Rédigez une fiche d'une page, à imprimer et à ranger près du serveur, qui dit : où est la clé de secours, comment afficher le menu de GRUB, la disposition des

disques, l'emplacement des phrases de passe et des sauvegardes, et les numéros à appeler. Qu'est-ce qui ne doit surtout pas y figurer ?

Pour aller plus loin

Sur la machine. `man e2fsck`, `man xfs_repair` et `man ddrescue` détaillent chaque option. `info grub` contient le manuel complet de GRUB, dont la liste de ses commandes. `man lvm.conf` explique `issue_discards` et l'archivage des métadonnées.

Paquets. `smartmontools` surveille la santé des disques et peut prévenir avant la panne (`smartd`). `boot-repair`, distribué hors des dépôts officiels, automatise la réparation de GRUB depuis un système de secours ; lisez ce qu'il propose avant de l'accepter.

En ligne. Le site de CGSecurity, `cgsecurity.org`, documente TestDisk et PhotoRec pas à pas, avec des captures. Le manuel de GNU ddrescue, sur `gnu.org/software/ddrescue`, explique la stratégie de ses passages et le format de sa carte. La page `help.ubuntu.com/community/Grub2` de la documentation communautaire d'Ubuntu décrit les réparations de GRUB les plus courantes.

PARTIE XI

Projets guidés

Deux projets complets qui mobilisent tout le livre : un serveur personnel auto-hébergé et un poste de travail professionnel.





CHAPITRE 32

Projet : un serveur personnel

Vos fichiers, vos mots de passe, vos photos et votre code vivent chez des prestataires dont vous louez l'accès. Ce projet les ramène sur une machine à vous. Il n'apporte presque rien de nouveau : il assemble, de l'installation à la sauvegarde, ce que les chapitres précédents ont construit pièce par pièce.

DANS CE CHAPITRE

- Choisir le matériel d'un serveur personnel.
- Installer Ubuntu Server sans intervention, avec autoinstall.
- Durcir la machine et installer Docker avec Ansible.
- Servir cinq applications derrière un seul relais HTTPS.
- Sauvegarder chaque nuit vers un disque externe, et vérifier la restauration.
- Surveiller les services et y accéder à distance par WireGuard.

> **AVANT DE COMMENCER**
>
> Tout le livre, et surtout les chapitres 2, 16, 23, 26, 28 et 29. Un nom de domaine et une adresse publique si le serveur doit être joignable depuis Internet. Comptez une journée, en plusieurs fois.

32.1 Le projet

À la fin de ce chapitre, une machine nommée `perso` rendra cinq services :

Service	Remplace	Adresse
Nextcloud	Google Drive, Dropbox, iCloud : fichiers, agenda, contacts	`cloud.maison.lan`
Vaultwarden	Bitwarden, 1Password : mots de passe	`mdp.maison.lan`
Immich	Google Photos : photos et vidéos du téléphone	`photos.maison.lan`
Forgejo	GitHub : dépôts Git, chapitre 27	`git.maison.lan`
Uptime Kuma	Un service de surveillance en ligne, chapitre 26	`etat.maison.lan`

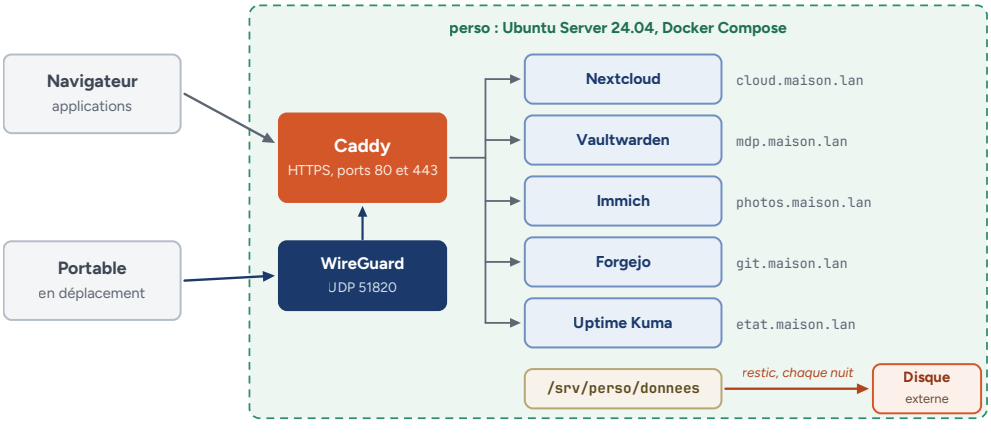


Figure 32.1 — Le serveur personnel : un relais HTTPS, cinq services, une sauvegarde chaque nuit.

Chaque service tourne dans des conteneurs, décrits dans un seul fichier Docker Compose. Caddy, vu au chapitre 23, reçoit toutes les connexions, obtient les certificats et transmet chaque demande au bon service. Les données de tous les services

sont rangées sous un même dossier, `/srv/perso/donnees`, que la sauvegarde emporte chaque nuit. Pour y accéder hors de chez soi, sans exposer tout le reste, WireGuard ouvre un tunnel chiffré.

Chaque étape renvoie au chapitre qui l'explique. Ici, on assemble.

32.2 Le matériel

	Serveur loué (VPS)	Mini-PC à la maison	Raspberry Pi 5
Coût	5 à 15 € par mois	150 à 400 € une fois	100 à 150 € avec boîtier et alimentation
Mémoire	2 à 8 Go	8 à 64 Go	4 à 16 Go
Stockage	Cher au-delà de 100 Go	Plusieurs disques, plusieurs téraoctets	Un SSD en USB ou en NVMe
Joignable depuis Internet	Oui, adresse fixe	Selon votre box et votre fournisseur	Idem
Vos données	Chez l'hébergeur	Chez vous	Chez vous
Pannes de courant, de réseau	Rares	Les vôtres	Les vôtres

Pour des photos et des fichiers, le volume décide souvent : un téraoctet coûte cher chez un hébergeur, presque rien sur un disque à la maison. Pour un service qui doit répondre en permanence, le serveur loué l'emporte. Beaucoup combinent les deux : la machine à la maison garde les données, et un petit serveur loué héberge ce qui doit toujours répondre, ou sert de point d'entrée.

Pour ce projet, comptez au moins 4 Go de mémoire, 8 si Immich doit analyser les photos pour y reconnaître les visages et les objets. Un Raspberry Pi 5 de 8 Go convient, avec Ubuntu Server pour Raspberry Pi, qui s'installe en copiant une image sur une carte ou un SSD avec Raspberry Pi Imager. Le reste de ce chapitre est identique.

> **NOTE · Sur le réseau d'essai**
>
> La machine d'essai de ce chapitre n'a que 2 Go de mémoire. Immich y tourne sans son module d'analyse des images, qui en demande à lui seul un à deux de plus. Tout le reste est conforme à ce qui est décrit. Les noms en `maison.lan` et les certificats viennent du réseau d'essai du livre, comme au chapitre 23.

32.3 Installer sans intervenir

Le chapitre 2 a présenté l'installation automatique. Voici le fichier qui installe `perso`. On le place, sous le nom `user-data`, sur une clé USB dont le volume s'appelle `CIDATA`, avec un fichier `meta-data` vide :

```
#cloud-config
autoinstall:
  version: 1
  locale: fr_FR.UTF-8
  keyboard:
    layout: fr
  timezone: Africa/Douala
  identity:
    hostname: perso
    username: gary
    # mkpasswd -m sha-512
    password: "$6$FugzxAXDKRleb/
t2$4iN9xBFHvctSO2PbGoMq.XtYAMqz0LeBu2WxiL23tn440U.qFiJZypoVNDKRRRsPsZvxWxe7h
Mw2OHUTCVlHO0"
  ssh:
    install-server: true
    allow-pw: false
    authorized-keys:
      - ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIPgNVDIjKQsmt8dDpBT49TVl5I8/33ZXLd3lbTojso7Y
gary@vostro
  network:
    version: 2
    ethernets:
      lan:
        match:
          name: "en*"
        dhcp4: true
  storage:
    layout:
      name: lvm
```

```
      sizing-policy: all
  updates: security
  shutdown: poweroff
```

Clé	Effet
`identity`	Le nom de la machine, le compte, son mot de passe chiffré
`ssh`	Le serveur SSH, la clé publique du poste, et pas de connexion par mot de passe
`network`	Une carte réseau quelconque, configurée par DHCP
`storage`	LVM, sur tout le disque (chapitre 2)
`updates: security`	Les correctifs de sécurité installés pendant l'installation
`shutdown: poweroff`	Éteindre la machine à la fin, plutôt que redémarrer sur la clé

On démarre sur la clé d'Ubuntu Server, l'autre clé branchée. L'installateur trouve le fichier, et demande une seule confirmation avant d'effacer le disque :

```
Confirmation is required to continue.
Add 'autoinstall' to your kernel command line to avoid this


Continue with autoinstall? (yes|no)
yes
```

Pour s'en passer, on ajoute `autoinstall` à la ligne du noyau dans le menu de démarrage de la clé, comme au chapitre 31. Six minutes plus tard, la machine s'éteint. On retire les clés, on la rallume, et l'on s'y connecte depuis le poste :

```
gary@vostro:~$ ssh perso.maison.lan
The authenticity of host 'perso.maison.lan (192.168.1.52)' can't be established.
ED25519 key fingerprint is SHA256:j9D2HMPdwALXrvho1ByVUNhgkk+jWRpV5lDS83Jmtmw.
[…]
gary@perso:~$ hostnamectl | grep -E 'Static hostname|Operating System'
 Static hostname: perso
Operating System: Ubuntu 24.04.5 LTS
gary@perso:~$ localectl | head -n 3
System Locale: LANG=fr_FR.UTF-8
    VC Keymap: (unset)
   X11 Layout: fr
gary@perso:~$ timedatectl | grep 'Time zone'
               Time zone: Africa/Douala (WAT, +0100)
gary@perso:~$ lsblk -o NAME,SIZE,TYPE,MOUNTPOINTS | grep -v loop
NAME                      SIZE TYPE MOUNTPOINTS
```

```
sda                         8G disk
sr0                      1024M rom
vda                        20G disk
├─vda1                    953M part /boot/efi
├─vda2                    1,8G part /boot
└─vda3                   17,3G part
  └─ubuntu--vg-ubuntu--lv 17,3G lvm  /
gary@perso:~$ sudo grep -rh PasswordAuthentication /etc/ssh/sshd_config.d/
PasswordAuthentication no
```

Tout y est : le nom, la langue, le clavier, le fuseau, LVM sur la totalité du disque, SSH par clé seulement. Le disque `sda` de 8 Go est le disque externe des sauvegardes, déjà branché. L'installateur garde le fichier utilisé dans `/var/log/installer/autoinstall-user-data`.

SUR LE TERRAIN · La carte réseau qui avait changé de nom

La première installation de `perso` s'est faite sans la section `network`. Au premier démarrage, la machine était injoignable. Sur sa console :

```
gary@perso:~$ ip -br link
lo               UNKNOWN        00:00:00:00:00:00
<LOOPBACK,UP,LOWER_UP>
enp0s4           DOWN           52:54:00:12:34:59 <BROADCAST,MULTICAST>
gary@perso:~$ sudo cat /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
```

Pendant l'installation, la carte s'appelait `enp0s3`. Une fois la clé d'installation retirée, les emplacements des périphériques avaient bougé, et la carte s'appelait `enp0s4`. L'installateur avait écrit une configuration pour un nom qui n'existait plus. La section `network` du fichier règle la question pour de bon : `match: name: "en*"` désigne n'importe quelle carte Ethernet, quel que soit son nom.

La même console a révélé un second détail : le clavier y était bien en AZERTY, comme demandé. Un détail qui compte le jour où l'on doit taper un mot de passe devant la machine.

32.4 Durcir et préparer avec Ansible

Le rôle `durcissement` du chapitre 29 s'applique tel quel. Un playbook l'appelle, puis ajoute ce que ce serveur demande de plus : les ports du Web et de WireGuard, Docker depuis son dépôt, WireGuard, restic, et le dossier des services.

```yaml
- name: Préparer le serveur personnel
  hosts: perso
  become: true
  roles:
    - durcissement

  tasks:
    - name: Ouvrir le Web et WireGuard dans le pare-feu
      community.general.ufw:
        rule: allow
        port: "{{ item.port }}"
        proto: "{{ item.proto }}"
      loop:
        - { port: "80", proto: tcp }
        - { port: "443", proto: tcp }
        - { port: "443", proto: udp }
        - { port: "51820", proto: udp }

    - name: Ajouter le dépôt de Docker
      ansible.builtin.deb822_repository:
        name: docker
        types: deb
        uris: https://download.docker.com/linux/ubuntu
        suites: "{{ ansible_distribution_release }}"
        components: stable
        signed_by: https://download.docker.com/linux/ubuntu/gpg

    - name: Installer Docker, WireGuard et les outils de sauvegarde
      ansible.builtin.apt:
        name: [docker-ce, docker-ce-cli, containerd.io, docker-compose-plugin,
               wireguard, restic, sqlite3]
        update_cache: true

    - name: Donner Docker aux administrateurs
      ansible.builtin.user:
        name: "{{ item }}"
        groups: docker
        append: true
      loop: "{{ admins }}"

    - name: Créer le dossier des services
      ansible.builtin.file:
        path: /srv/perso
        state: directory
```

```
        owner: gary
        group: gary
        mode: "0750"
```

`deb822_repository` ajoute le dépôt de Docker et sa clé en une tâche, ce que le chapitre 28 faisait en trois commandes. On ajoute `perso` à l'inventaire, sous son nom :

```
gary@vostro:~/ansible$ grep perso inventaire.ini
perso   ansible_host=perso.maison.lan
gary@vostro:~/ansible$ ansible-playbook perso.yml -K --ask-vault-pass
BECOME password:
Vault password:
[…]
TASK [Ouvrir le Web et WireGuard dans le pare-feu] *********************************
changed: [perso] => (item={'port': '80', 'proto': 'tcp'})
changed: [perso] => (item={'port': '443', 'proto': 'tcp'})
changed: [perso] => (item={'port': '443', 'proto': 'udp'})
changed: [perso] => (item={'port': '51820', 'proto': 'udp'})
[…]
PLAY RECAP *************************************************************************
perso                      : ok=27   changed=25   unreachable=0    failed=0    […]
```

Puis une seconde fois, pour l'idempotence :

```
gary@vostro:~/ansible$ ansible-playbook perso.yml -K --ask-vault-pass | tail -n 3
[…]
perso                      : ok=25   changed=0    unreachable=0    failed=0    […]
```

Deux remarques nées des essais. Le mot de passe du coffre est demandé parce que `perso` appartient au groupe `serveurs`, dont les variables contiennent le fichier chiffré du chapitre 29. Et l'inventaire désigne `perso` par son nom plutôt que par son adresse : SSH avait enregistré l'empreinte sous le nom, et Ansible, venu par l'adresse, se voyait reposer la question de l'empreinte.

32.5 Le disque des sauvegardes

Le disque externe reçoit un système de fichiers et une étiquette, puis une ligne dans `/etc/fstab`. L'option `nofail` évite qu'un disque débranché bloque le démarrage, comme au chapitre 14 :

```
gary@perso:~$ lsblk -o NAME,SIZE,TRAN,MODEL /dev/sda
NAME SIZE TRAN MODEL
sda    8G usb  QEMU HARDDISK
gary@perso:~$ sudo mkfs.ext4 -q -L EXTERNE /dev/sda
gary@perso:~$ sudo mkdir -p /mnt/externe
gary@perso:~$ echo 'LABEL=EXTERNE /mnt/externe ext4 defaults,nofail,noatime 0 2' | \
    sudo tee -a /etc/fstab
```

```
LABEL=EXTERNE /mnt/externe ext4 defaults,nofail,noatime 0 2
gary@perso:~$ sudo systemctl daemon-reload
gary@perso:~$ sudo mount /mnt/externe
gary@perso:~$ df -h /mnt/externe | tail -n 1
/dev/sda        7,8G   24K  7,4G   1% /mnt/externe
```

Vérifiez deux fois le nom du disque avant `mkfs` : `lsblk` affiche sa taille et son modèle, et la colonne `TRAN` dit `usb` pour un disque branché en USB.

32.6 Les services

Le fichier Compose

Tout le serveur tient dans un dossier, `/srv/perso`, et un fichier :

```yaml
# Serveur personnel : un relais, cinq services. Les secrets sont dans .env.
name: perso

services:
  caddy:
    image: caddy:2.11
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
      - "443:443/udp"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      - ./donnees/caddy:/data

  # Nextcloud : fichiers, agenda, contacts
  nextcloud:
    image: nextcloud:35-apache
    restart: unless-stopped
    depends_on:
      nextcloud-base:
        condition: service_healthy
      nextcloud-redis:
        condition: service_started
    environment:
      MYSQL_HOST: nextcloud-base
      MYSQL_DATABASE: nextcloud
      MYSQL_USER: nextcloud
      MYSQL_PASSWORD: ${MDP_NEXTCLOUD_BASE}
      REDIS_HOST: nextcloud-redis
      NEXTCLOUD_ADMIN_USER: gary
      NEXTCLOUD_ADMIN_PASSWORD: ${MDP_NEXTCLOUD_ADMIN}
      NEXTCLOUD_TRUSTED_DOMAINS: cloud.maison.lan
      OVERWRITEPROTOCOL: https
      TRUSTED_PROXIES: 172.16.0.0/12
```

```
    volumes:
      - ./donnees/nextcloud:/var/www/html

  nextcloud-base:
    image: mariadb:11.4
    restart: unless-stopped
    command: --transaction-isolation=READ-COMMITTED
    environment:
      MARIADB_DATABASE: nextcloud
      MARIADB_USER: nextcloud
      MARIADB_PASSWORD: ${MDP_NEXTCLOUD_BASE}
      MARIADB_RANDOM_ROOT_PASSWORD: "1"
    volumes:
      - ./donnees/nextcloud-base:/var/lib/mysql
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--innodb_initialized"]
      interval: 10s
      retries: 10

  nextcloud-redis:
    image: redis:8-alpine
    restart: unless-stopped

  # Vaultwarden : mots de passe, compatible avec les applications Bitwarden
  vaultwarden:
    image: vaultwarden/server:1.37.3-alpine
    restart: unless-stopped
    environment:
      DOMAIN: https://mdp.maison.lan
      SIGNUPS_ALLOWED: ${VW_INSCRIPTIONS:-false}
    volumes:
      - ./donnees/vaultwarden:/data

  # Immich : photos, sans l'analyse des images (voir le texte)
  immich-server:
    image: ghcr.io/immich-app/immich-server:v3.2.4
    restart: unless-stopped
    depends_on: [immich-base, immich-redis]
    environment:
      DB_HOSTNAME: immich-base
      DB_USERNAME: postgres
      DB_PASSWORD: ${MDP_IMMICH_BASE}
      DB_DATABASE_NAME: immich
      REDIS_HOSTNAME: immich-redis
      IMMICH_MACHINE_LEARNING_ENABLED: "false"
    volumes:
      - ./donnees/immich/photos:/data
      - /etc/localtime:/etc/localtime:ro

  immich-base:
    image: ghcr.io/immich-app/postgres:14-vectorchord0.4.3-pgvectors0.2.0
    restart: unless-stopped
    shm_size: 128mb
```

```
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ${MDP_IMMICH_BASE}
      POSTGRES_DB: immich
      POSTGRES_INITDB_ARGS: --data-checksums
    volumes:
      - ./donnees/immich/base:/var/lib/postgresql/data

  immich-redis:
    image: valkey/valkey:9
    restart: unless-stopped

  # Forgejo : dépôts Git
  forgejo:
    image: codeberg.org/forgejo/forgejo:16
    restart: unless-stopped
    environment:
      USER_UID: "1000"
      USER_GID: "1000"
      FORGEJO__server__DOMAIN: git.maison.lan
      FORGEJO__server__ROOT_URL: https://git.maison.lan/
      FORGEJO__database__DB_TYPE: sqlite3
      FORGEJO__security__INSTALL_LOCK: "true"
      FORGEJO__service__DISABLE_REGISTRATION: "true"
    volumes:
      - ./donnees/forgejo:/data

  # Uptime Kuma : surveillance
  kuma:
    image: louislam/uptime-kuma:2
    restart: unless-stopped
    volumes:
      - ./donnees/kuma:/app/data
```

Les principes sont ceux du chapitre 28, appliqués à cinq applications :

- **Un seul relais.** Seul Caddy publie des ports. Les applications et leurs bases ne sont joignables que par le réseau interne de Compose, sous leur nom de service.
- **Des données rangées ensemble.** Chaque service écrit sous `./donnees`, par des montages liés plutôt que des volumes : la sauvegarde n'a qu'un dossier à emporter.
- **Des versions fixées.** `nextcloud:35-apache`, `immich-server:v3.2.4` : une mise à jour ne passe qu'au moment choisi.
- **Des secrets à part**, dans `.env`, hors du dépôt Git du projet.
- **Des dépendances déclarées.** Nextcloud attend que sa base soit prête, et démarre avec son cache Redis.

Ce dernier point vient d'un oubli. Une première version du fichier ne déclarait pas Redis parmi les dépendances de Nextcloud. Lancé seul, Nextcloud tournait sans son cache, et la première commande d'administration répondait `Redis server went away`.

Nextcloud reçoit quelques variables qui lui disent qu'il est derrière un relais : `OVERWRITEPROTOCOL` pour qu'il fabrique des adresses en `https`, `TRUSTED_PROXIES` pour qu'il croie l'adresse du visiteur que Caddy lui transmet. Immich tourne sans `IMMICH_MACHINE_LEARNING_ENABLED`, et donc sans son conteneur d'analyse d'images : avec la mémoire nécessaire, ajoutez le service `immich-machine-learning` de la documentation d'Immich et retirez cette ligne.

Caddy

```
{
        email gary@maison.lan
}

cloud.maison.lan {
        request_body {
                max_size 10GB
        }
        redir /.well-known/carddav /remote.php/dav 301
        redir /.well-known/caldav /remote.php/dav 301
        reverse_proxy nextcloud:80
}

mdp.maison.lan {
        reverse_proxy vaultwarden:80
}

photos.maison.lan {
        reverse_proxy immich-server:2283
}

git.maison.lan {
        reverse_proxy forgejo:3000
}

etat.maison.lan {
        reverse_proxy kuma:3001
}
```

Un bloc par nom, une ligne `reverse_proxy` vers le service. Caddy obtient lui-même un certificat pour chacun des noms, le renouvelle, et redirige le HTTP vers le HTTPS. Pour Nextcloud, deux redirections aident les applications d'agenda et de contacts à trouver leur adresse, et la taille des envois passe à 10 Go.

> **NOTE · Sur le réseau d'essai**
>
> Les certificats du banc viennent de Pebble, le serveur de test de Let's Encrypt utilisé au chapitre 23. Le `Caddyfile` du banc porte donc, dans son bloc global, deux lignes de plus : `acme_ca https://pebble:14000/dir` et `acme_ca_root`, le certificat de Pebble. Sur un serveur joignable depuis Internet, avec des noms de domaine publics, le fichier ci-dessus suffit : Caddy s'adresse d'office à Let's Encrypt.

Démarrer

```
gary@perso:/srv/perso$ printf '%s\n' "MDP_NEXTCLOUD_BASE=$(openssl rand -hex 24)" \
    "MDP_NEXTCLOUD_ADMIN=$(openssl rand -hex 16)" "MDP_IMMICH_BASE=$(openssl rand -
hex 24)" \
    VW_INSCRIPTIONS=true > .env
gary@perso:/srv/perso$ chmod 600 .env
gary@perso:/srv/perso$ docker compose config --services
caddy
forgejo
immich-base
immich-redis
immich-server
nextcloud-base
nextcloud
nextcloud-redis
kuma
vaultwarden
gary@perso:/srv/perso$ docker compose up -d
[…]
gary@perso:/srv/perso$ docker compose ps --format 'table {{.Service}}\t{{.Status}}'
SERVICE           STATUS
caddy             Up 36 minutes
forgejo           Up 30 minutes
immich-base       Up 12 minutes (healthy)
immich-redis      Up 12 minutes
immich-server     Up About a minute (healthy)
kuma              Up 38 minutes (healthy)
nextcloud         Up 38 minutes
nextcloud-base    Up 38 minutes (healthy)
nextcloud-redis   Up 33 minutes
vaultwarden       Up 38 minutes (healthy)
```

Le premier `docker compose up -d` télécharge les images : plus de quatre gigaoctets, dont deux et demi pour le seul serveur d'Immich. Sur une connexion lente, comptez plusieurs heures ; les services déjà téléchargés peuvent démarrer sans attendre les autres, en les nommant : `docker compose up -d caddy nextcloud`. C'est ce que montrent les durées ci-dessus, Immich ayant démarré une demi-heure après le reste.

La mémoire décide du reste :

```
gary@perso:/srv/perso$ docker stats --no-stream --format 'table {{.Name}}\t{{.MemUsage}}'
NAME                     MEM USAGE / LIMIT
perso-immich-server-1    612.3MiB / 1.916GiB
perso-immich-base-1      365.5MiB / 1.916GiB
perso-immich-redis-1     5.098MiB / 1.916GiB
perso-nextcloud-redis-1  2.926MiB / 1.916GiB
perso-caddy-1            18.42MiB / 1.916GiB
perso-nextcloud-1        29.35MiB / 1.916GiB
perso-kuma-1             85.37MiB / 1.916GiB
perso-nextcloud-base-1   10.69MiB / 1.916GiB
perso-forgejo-1          74.82MiB / 1.916GiB
perso-vaultwarden-1      15.07MiB / 1.916GiB
gary@perso:/srv/perso$ free -h | sed -n 2,3p
Mem:           1,9Gi       1,6Gi        96Mi       161Mi       618Mi       367Mi
Swap:          2,0Gi       1,0Gi       991Mi
```

Immich et sa base occupent à eux seuls un gigaoctet. Avec 2 Go, la machine d'essai puise déjà dans l'espace d'échange : c'est la limite basse, et la raison des 4 Go conseillés.

Vérifier depuis le poste

```
gary@vostro:~$ for n in cloud mdp git etat; do
>     curl -s -o /dev/null -w "$n.maison.lan : %{http_code}\n" https://$n.maison.lan/
> done
cloud.maison.lan : 302
mdp.maison.lan : 200
git.maison.lan : 200
etat.maison.lan : 302
gary@vostro:~$ curl -sI http://cloud.maison.lan | grep -iE '^HTTP|^location'
HTTP/1.1 308 Permanent Redirect
Location: https://cloud.maison.lan/
gary@vostro:~$ curl -s https://cloud.maison.lan/status.php; echo
{"installed":true,"maintenance":false,"needsDbUpgrade":false,"version":"35.0.1.1",[…]}
gary@vostro:~$ curl -s https://photos.maison.lan/api/server/version; echo
{"major":3,"minor":2,"patch":4,"prerelease":null}
```

Les codes 302 sont des redirections vers la page de connexion. Nextcloud est installé, avec le compte `gary` créé par `NEXTCLOUD_ADMIN_USER`. Envoyons-lui un fichier par WebDAV, le protocole que ses applications utilisent :

```
gary@vostro:~$ echo "Note du jour : le serveur personnel fonctionne." > note.txt
gary@vostro:~$ curl -s -u "gary:$MDP_NC" -T note.txt \
    https://cloud.maison.lan/remote.php/dav/files/gary/note.txt -w '%{http_code}\n'
201
gary@vostro:~$ curl -s -u "gary:$MDP_NC" \
    https://cloud.maison.lan/remote.php/dav/files/gary/note.txt
Note du jour : le serveur personnel fonctionne.
```

Pour les applications du téléphone et de l'ordinateur, créez plutôt un **mot de passe d'application** dans les réglages de sécurité de Nextcloud : il se révoque seul, sans changer le mot de passe du compte.

Forgejo

Forgejo démarre sans page d'installation (`INSTALL_LOCK`) et sans inscriptions libres. Le compte administrateur se crée en ligne de commande, puis un jeton d'accès pour le poste :

```
gary@vostro:~$ ssh perso.maison.lan 'cd /srv/perso && docker compose exec -u git forgejo \
    forgejo admin user create --admin --username gary --email gary@maison.lan \
    --random-password --must-change-password=false' | tail -n 1
New user 'gary' has been successfully created!
gary@vostro:~$ JETON_GIT=$(ssh perso.maison.lan 'cd /srv/perso && docker compose exec -T \
    -u git forgejo \
    forgejo admin user generate-access-token --username gary --token-name poste \
    --scopes write:repository,write:user --raw')
gary@vostro:~$ curl -s -H "Authorization: token $JETON_GIT" \
    -H 'Content-Type: application/json' \
    -d '{"name":"sauvegarde","private":true}' https://git.maison.lan/api/v1/user/repos | \
    python3 -c 'import json,sys; d=json.load(sys.stdin); print(d["full_name"])'
gary/sauvegarde https://git.maison.lan/gary/sauvegarde.git
gary@vostro:~$ cd sauvegarde
gary@vostro:~/sauvegarde$ git push https://gary:$JETON_GIT@git.maison.lan/gary/
sauvegarde.git \
    main --tags 2>&1 | tail -n 3
To https://git.maison.lan/gary/sauvegarde.git
 * [new branch]      main -> main
 * [new tag]         v1.0 -> v1.0
```

Le projet du chapitre 27 a désormais une copie chez vous, étiquette `v1.0` comprise. Le mot de passe tiré au hasard s'affiche une fois : on s'en sert pour la première connexion à l'interface, puis on le change.

Vaultwarden

Le coffre de mots de passe ne doit accepter qu'un compte, le vôtre. La variable `VW_INSCRIPTIONS` vaut `true` le temps de le créer depuis la page `https://mdp.maison.lan`, puis on ferme :

```
gary@perso:/srv/perso$ sed -i 's/^VW_INSCRIPTIONS=true$/VW_INSCRIPTIONS=false/' .env
gary@perso:/srv/perso$ docker compose up -d vaultwarden
[…]
gary@vostro:~$ curl -s -H "Content-Type: application/json" \
    -d '{"email":"essai@maison.lan","name":"Essai","receiveMarketingEmails":false}' \
    https://mdp.maison.lan/identity/accounts/register/send-verification-email | \
    head -c 70; echo
{"message":"Registration not allowed or user already exists","valid
```

Les extensions de navigateur et les applications Bitwarden se connectent ensuite à ce serveur : on indique `https://mdp.maison.lan` comme serveur auto-hébergé à la première connexion. Le mot de passe principal ne quitte jamais l'appareil : Vaultwarden ne stocke que des données chiffrées.

32.7 Surveiller

Uptime Kuma s'ouvre sur `https://etat.maison.lan`. Le premier écran choisit sa base (SQLite suffit), le second crée le compte administrateur. On ajoute ensuite une sonde HTTP par service, sur une adresse qui prouve qu'il fonctionne vraiment : `status.php` pour Nextcloud, `/alive` pour Vaultwarden, `/api/server/ping` pour Immich, `/api/healthz` pour Forgejo.

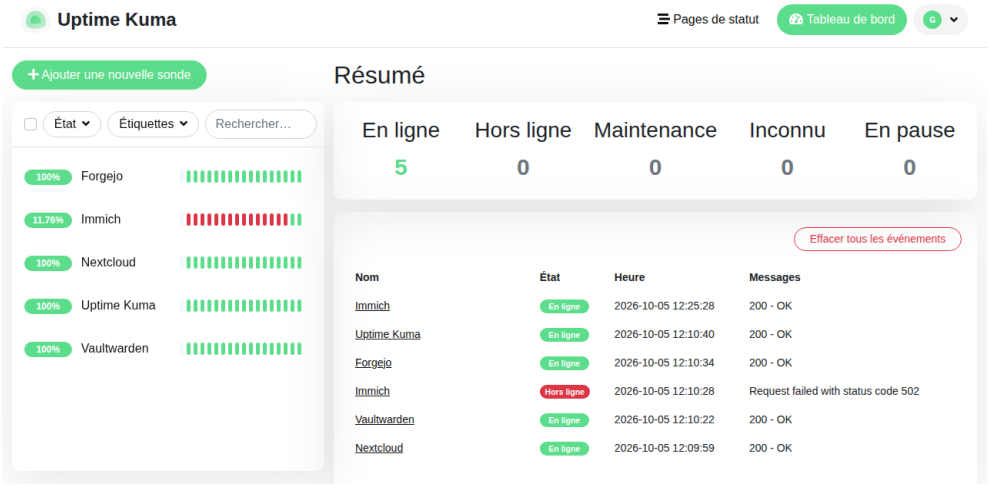


Figure 32.2 — Uptime Kuma surveille les cinq services. L'historique d'Immich garde la trace de sa demi-heure d'absence, le temps du téléchargement de ses images.

Les sondes de la machine d'essai ont l'option « Ignorer les erreurs TLS » cochée, les certificats de Pebble n'étant pas reconnus à l'intérieur du conteneur. Avec Let's Encrypt, laissez-la décochée : Kuma préviendra aussi avant l'expiration d'un certificat. Les notifications se règlent dans ses paramètres : courriel, Telegram, Matrix et des dizaines d'autres services.

Kuma tourne sur la machine qu'il surveille : si `perso` s'arrête, il s'arrête avec elle. Le chapitre 26 l'a dit, la disponibilité se vérifie aussi depuis ailleurs. Une seconde instance sur une autre machine, ou un service de surveillance en ligne, surveille `etat.maison.lan` lui-même. Et pour l'état de la machine, processeur, mémoire,

disques, `prometheus-node-exporter` installé sur `perso` s'ajoute aux cibles du Prometheus du chapitre 26.

32.8 Sauvegarder

Le chapitre 16 a posé les règles : automatique, vérifiée, hors de la machine. Le script exporte d'abord chaque base dans un fichier cohérent, puis confie le tout à restic :

```
#!/bin/bash
#
# sauvegarde-perso — exporte les bases, puis sauvegarde /srv/perso avec restic
# Lancé chaque nuit par sauvegarde-perso.timer.

set -euo pipefail

export RESTIC_REPOSITORY=/mnt/externe/restic
export RESTIC_PASSWORD_FILE=/root/.restic-perso

cd /srv/perso
mountpoint -q /mnt/externe || { echo "Le disque externe n'est pas branché." >&2; exit 1; }
mkdir -p exports

# Nextcloud en maintenance, le temps d'exporter sa base
docker compose exec -T -u www-data nextcloud php occ maintenance:mode --on
docker compose exec -T nextcloud-base sh -c \
    'mariadb-dump --single-transaction -u nextcloud -p"$MARIADB_PASSWORD" nextcloud' > expo
rts/nextcloud.sql
docker compose exec -T -u www-data nextcloud php occ maintenance:mode --off

docker compose exec -T immich-base pg_dumpall --clean --if-exists -U postgres > exports/
immich.sql
sqlite3 donnees/vaultwarden/db.sqlite3 ".backup exports/vaultwarden.sqlite3"

# Les fichiers bruts des bases sont remplacés par leurs exports
restic backup /srv/perso --exclude /srv/perso/donnees/nextcloud-base \
    --exclude /srv/perso/donnees/immich/base
restic forget --keep-daily 7 --keep-weekly 4 --keep-monthly 6 --prune
```

Service	Méthode
Nextcloud	`mariadb-dump --single-transaction`, en mode maintenance le temps de l'export
Immich	`pg_dumpall`, comme au chapitre 24
Vaultwarden	`.backup` de SQLite, sûr même pendant que le service écrit
Forgejo, Kuma, fichiers	Copiés tels quels par restic

Les dossiers bruts des deux bases sont exclus : copiés pendant que la base écrit, ils pourraient être incohérents, et les exports les remplacent. La minuterie lance le tout chaque nuit, à 3 h 15 avec un écart aléatoire d'au plus un quart d'heure :

```
[Unit]
Description=Sauvegarde nocturne du serveur personnel

[Timer]
OnCalendar=*-*-* 03:15
RandomizedDelaySec=15m
Persistent=true

[Install]
WantedBy=timers.target
```

```
gary@perso:/srv/perso$ sudo install -m 755 /tmp/sauvegarde-perso /usr/local/sbin/
gary@perso:/srv/perso$ sudo install -m 644 /tmp/sauvegarde-perso.service \
    /tmp/sauvegarde-perso.timer /etc/systemd/system/
gary@perso:/srv/perso$ sudo sh -c 'umask 077; openssl rand -base64 32 > /root/.restic-perso'
gary@perso:/srv/perso$ sudo restic init --repo /mnt/externe/restic \
    --password-file /root/.restic-perso
created restic repository 9b42ea41d3 at /mnt/externe/restic
[…]
gary@perso:/srv/perso$ time sudo systemctl start sauvegarde-perso.service

real    0m24,554s
[…]
gary@perso:/srv/perso$ journalctl -u sauvegarde-perso --since -10min --no-pager -o cat | \
    grep -E 'Files:|Added to the repository|processed|snapshot .* saved'
Files:       28275 new,     0 changed,     0 unmodified
Added to the repository: 756.525 MiB (282.764 MiB stored)
processed 28275 files, 826.215 MiB in 0:21
snapshot 47f33716 saved
gary@perso:/srv/perso$ sudo systemctl enable --now sauvegarde-perso.timer
gary@perso:/srv/perso$ systemctl list-timers sauvegarde-perso.timer --no-pager | head -n 2
NEXT                        LEFT LAST PASSED UNIT                    ACTIVATES
Tue 2026-10-06 03:25:08 WAT  15h -         - sauvegarde-perso.timer sauvegarde-perso.service
```

Le premier passage emporte tout : 826 Mo, réduits à 283 par la compression. Le suivant, une fois Immich en service, n'a pris que deux secondes :

```
Files:           6 new,     9 changed, 28266 unmodified
Added to the repository: 52.876 MiB (18.739 MiB stored)
processed 28281 files, 876.969 MiB in 0:02
```

Une sauvegarde ne vaut que si l'on sait restaurer. Retrouvons la note envoyée dans Nextcloud :

```
gary@perso:/srv/perso$ sudo restic -r /mnt/externe/restic \
    --password-file /root/.restic-perso restore latest --target /tmp/restauration \
    --include /srv/perso/donnees/nextcloud/data/gary/files/note.txt
[…]
```

```
gary@perso:/srv/perso$ sudo cat \
    /tmp/restauration/srv/perso/donnees/nextcloud/data/gary/files/note.txt
Note du jour : le serveur personnel fonctionne.
gary@perso:/srv/perso$ grep -c 'CREATE TABLE' exports/nextcloud.sql
138
```

ATTENTION · Un disque à la maison ne suffit pas

Le disque externe protège d'une erreur ou d'une panne du serveur. Pas d'un incendie, d'un vol, ni d'une surtension qui grillerait les deux. La règle du chapitre 16 demande une copie hors du lieu : un second dépôt restic chez un hébergeur, `restic copy` d'un dépôt à l'autre, ou un disque qu'on emporte ailleurs chaque mois. Et gardez le fichier `/root/.restic-perso` ailleurs que sur le serveur : sans lui, le dépôt est illisible.

32.9 Accéder de l'extérieur avec WireGuard

Ouvrir Nextcloud et Immich sur Internet est possible : c'est ce que font les ports 80 et 443, et c'est nécessaire pour que Let's Encrypt délivre ses certificats. Mais Kuma, l'administration de Forgejo ou SSH n'ont rien à faire face à Internet. **WireGuard** crée un réseau privé chiffré entre vos appareils et le serveur, d'où qu'ils se connectent.

WireGuard est dans le noyau Linux depuis 2020. Chaque appareil a une paire de clés, comme pour SSH. Le serveur connaît la clé publique de chaque appareil autorisé, et chaque appareil celle du serveur.

```
gary@perso:~$ sudo sh -c 'umask 077; wg genkey | tee /etc/wireguard/perso.cle | \
    wg pubkey > /etc/wireguard/perso.pub'
gary@perso:~$ sudo cat /etc/wireguard/perso.pub
+dw9QPgnSWZ1y2ROhG6V9qSpSPqO7Vc86AgPxFGDpkk=
gary@vostro:~$ sudo sh -c 'umask 077; wg genkey | tee /etc/wireguard/vostro.cle | \
    wg pubkey > /etc/wireguard/vostro.pub'
gary@vostro:~$ sudo cat /etc/wireguard/vostro.pub
uudq8PN5sNxNM+jyISlzuUvFF1vjio0xSi/K9eWkyx0=
```

Sur le serveur, `/etc/wireguard/wg0.conf` donne au tunnel son adresse, son port, sa clé privée, et la liste des appareils admis :

```
# /etc/wireguard/wg0.conf sur perso
[Interface]
Address = 10.8.0.1/24
ListenPort = 51820
PrivateKey = <contenu de /etc/wireguard/perso.cle>
```

```
[Peer]
# Le portable de Gary
PublicKey = <clé publique de vostro>
AllowedIPs = 10.8.0.2/32
```

Sur le poste, le même fichier pointe vers le serveur. `Endpoint` est l'adresse publique du serveur, ou le nom qui la désigne ; `PersistentKeepalive` garde le tunnel ouvert à travers les box et les pare-feu qui oublient les connexions inactives :

```
# /etc/wireguard/wg0.conf sur le poste
[Interface]
Address = 10.8.0.2/24
PrivateKey = <contenu de /etc/wireguard/vostro.cle>

[Peer]
# Le serveur personnel
PublicKey = <clé publique de perso>
Endpoint = perso.example.org:51820
AllowedIPs = 10.8.0.0/24
PersistentKeepalive = 25
```

On remplace les marques entre chevrons par les clés, on protège les fichiers (`chmod 600`), et l'on démarre :

```
gary@perso:~$ sudo systemctl enable --now wg-quick@wg0
gary@perso:~$ sudo wg show
interface: wg0
  public key: +dw9QPgnSWZ1y2ROhG6V9qSpSPqO7Vc86AgPxFGDpkk=
  private key: (hidden)
  listening port: 51820

peer: uudq8PN5sNxNM+jyISlzuUvFF1vjio0xSi/K9eWkyx0=
  allowed ips: 10.8.0.2/32
gary@vostro:~$ sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.8.0.2/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
gary@vostro:~$ ping -c 3 10.8.0.1
[…]
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
gary@vostro:~$ sudo wg show wg0 | grep -E 'endpoint|handshake|transfer'
  endpoint: 192.168.1.52:51820
  latest handshake: 3 seconds ago
  transfer: 476 B received, 564 B sent
gary@vostro:~$ curl -s --resolve cloud.maison.lan:443:10.8.0.1 \
    https://cloud.maison.lan/status.php | head -c 40; echo
{"installed":true,"maintenance":false,"n
```

La poignée de main a eu lieu, le serveur répond à son adresse du tunnel, et Nextcloud à travers lui. Sur le téléphone, l'application WireGuard importe la même configuration, souvent par un code QR : `qrencode -t ansiutf8 < fichier.conf`, du paquet `qrencode`, l'affiche dans le terminal. `sudo wg-quick down wg0` referme le tunnel.

Tailscale fait le même travail sans configuration : il s'appuie sur WireGuard, distribue les clés lui-même et traverse les box sans ouvrir de port. En échange, la coordination passe par ses serveurs. Headscale, son équivalent libre, se pose sur votre propre serveur.

> **ASTUCE · Le serveur derrière une box**
>
> À la maison, la box reçoit l'adresse publique. Pour joindre le serveur de l'extérieur, on redirige dans la box les ports 80, 443 et 51820 vers l'adresse du serveur, que le DHCP de la box doit lui réserver. Si votre fournisseur change votre adresse publique, un service de DNS dynamique tient un nom à jour. Et si votre fournisseur ne vous donne pas d'adresse publique du tout, un petit serveur loué peut servir de point d'entrée, relié à la maison par WireGuard.

32.10 Entretenir

Quand	Quoi	Comment
Chaque nuit	Sauvegarde	La minuterie, et son signe de vie (chapitre 26)
Chaque semaine	Lire Kuma et les journaux	`etat.maison.lan`, `docker compose logs --since 7d`
Chaque mois	Mettre à jour les images	`docker compose pull && docker compose up -d`, en lisant d'abord les notes de version
Chaque mois	Essayer une restauration	Un fichier, une base, sur une autre machine
Chaque trimestre	Relancer Ansible	`ansible-playbook perso.yml`, qui doit annoncer `changed=0`

Nextcloud et Immich demandent une attention particulière aux mises à jour : Nextcloud ne passe qu'une version majeure à la fois (de 35 à 36, jamais de 34 à 36), et Immich publie parfois des changements qui demandent une action. Leurs notes de version le disent.

En résumé

- Le matériel se choisit selon les données : un VPS pour ce qui doit toujours répondre, une machine à la maison pour le volume.
- L'installation automatique installe une machine en six minutes. Déclarez le réseau par un motif de nom (`en*`).
- Le rôle Ansible du chapitre 29, complété de Docker et des ports, prépare la machine, et se relance sans rien changer.
- Un seul fichier Compose, un seul relais Caddy qui obtient les certificats, des données sous un seul dossier, des secrets dans `.env`.
- Uptime Kuma surveille chaque service ; une surveillance extérieure surveille Kuma.
- Chaque nuit, les bases sont exportées puis restic emporte tout vers un disque externe. Une restauration s'essaie.
- WireGuard donne accès au serveur de partout, sans exposer ce qui n'a pas à l'être.

Atelier — Votre serveur

Objectif : monter votre propre serveur personnel, sur une machine virtuelle ou une vraie machine, avec au moins trois des cinq services. Comptez une journée, en plusieurs séances.

La machine

1. Adaptez le fichier `user-data` : votre nom de machine, votre compte, votre mot de passe chiffré, votre clé SSH. Installez.
2. Ajoutez la machine à votre inventaire Ansible, et appliquez `perso.yml`. Relancez-le : `changed=0`.

Les services

3. Choisissez vos services, et retirez les autres du fichier Compose et du `Caddyfile`. Créez le `.env`.
4. Si votre serveur n'est pas joignable depuis Internet, utilisez `tls internal` dans chaque bloc du `Caddyfile`, comme au chapitre 23, et importez la racine de Caddy sur vos postes.

5. Démarrez. Vérifiez chaque service depuis le poste.

La durée

6. Branchez un disque externe, installez la sauvegarde, lancez-la une fois.
7. Supprimez un fichier dans Nextcloud, et restaurez-le depuis restic.
8. Arrêtez un service et regardez Kuma le signaler.

L'accès

9. Montez WireGuard entre le serveur et votre téléphone ou votre portable. Joignez un service par le tunnel, depuis un autre réseau, par exemple le partage de connexion du téléphone.

Vérifier

- `docker compose ps` montre tous vos services `Up`, et ceux qui ont un contrôle de santé `healthy`.
- `restic snapshots` liste au moins deux instantanés, et la restauration a rendu le fichier intact.
- Depuis un autre réseau, le tunnel donne accès au service ; sans le tunnel, Kuma reste injoignable.

Exercices

Exercice 1 — Le choix

Un photographe amateur a 600 Go de photos, une connexion ADSL et un budget de 200 €. Un développeur veut un Forgejo et un Vaultwarden toujours joignables, avec peu de données. Que conseillez-vous à chacun ?

Exercice 2 — Un sixième service

Ajoutez au projet un service de votre choix, par exemple Jellyfin pour les films ou Paperless-ngx pour les documents. Écrivez son bloc dans `compose.yaml`, son nom dans le `Caddyfile`, et sa sonde dans Kuma. Que faut-il ajouter au script de sauvegarde ?

Exercice 3 — La sauvegarde hors du lieu

Créez un second dépôt restic sur une autre machine, accessible par SFTP. Modifiez le script pour qu'il copie chaque nuit les instantanés du disque externe vers ce dépôt avec `restic copy`. Comment vérifiez-vous que la copie distante est utilisable ?

Exercice 4 — Restaurer une base

Arrêtez Nextcloud, supprimez le dossier `donnees/nextcloud-base`, et remettez le service en état à partir de `exports/nextcloud.sql` seulement. Notez chaque commande : c'est votre plan de reprise.

Exercice 5 — Mettre à jour

Nextcloud publie une nouvelle version mineure. Décrivez, dans l'ordre, ce que vous faites avant, pendant et après la mise à jour, et comment vous revenez en arrière si elle échoue.

Pour aller plus loin

Les projets. Chaque service a une documentation soignée : `docs.nextcloud.com` (en particulier le guide d'administration et sa partie sur les relais), `docs.immich.app`, `github.com/dani-garcia/vaultwarden/wiki`, `forgejo.org/docs`, et le wiki d'Uptime Kuma sur GitHub. Lisez leurs pages de sauvegarde : elles disent ce qu'il faut emporter.

Pour aller plus loin dans l'auto-hébergement. La liste *awesome-selfhosted*, sur GitHub, recense plus d'un millier de logiciels libres à héberger soi-même, classés par usage. Le site de WireGuard, `wireguard.com`, explique en quelques pages son fonctionnement et ses choix cryptographiques.



CHAPITRE 33

Projet : un poste de travail

Le serveur est en place. Reste la machine sur laquelle vous passez vos journées. Ce dernier projet en fait un outil de travail : un bureau qui ne gêne pas, un terminal rapide, les langages de chaque projet sans conflit, des mots de passe et des fichiers synchronisés, une sauvegarde qui tourne seule, et des réglages qu'on réinstalle en une commande.

DANS CE CHAPITRE

- Choisir un environnement de bureau et l'alléger.
- Monter un terminal moderne : zsh, starship, tmux.
- Installer les langages par projet avec mise.
- Gérer ses mots de passe avec KeePassXC et ses fichiers avec Syncthing.
- Sauvegarder le système avec Timeshift et ses fichiers avec restic.
- Versionner ses réglages avec Git et GNU Stow.
- Choisir, pour chaque logiciel, la bonne source : APT, Snap, Flatpak, dépôt d'éditeur ou conteneur.

AVANT DE COMMENCER

Une installation d'Ubuntu Desktop 24.04, chapitres 2 et 3. Les chapitres 7, 8, 16, 27 et 28.

NOTE · Ce qui a été vérifié, et comment

Tout ce qui se passe dans un terminal a été rejoué sur une installation d'Ubuntu 24.04 neuve, nommée `portable`. Les sorties de ce chapitre en viennent. Les réglages purement graphiques, ceux du bureau GNOME et de Timeshift, sont décrits d'après la documentation de leurs projets et les paquets des dépôts d'Ubuntu 24.04, sans capture.

33.1 Le bureau

GNOME, KDE ou Xfce

Ubuntu 24.04 livre GNOME 46, modifié par Canonical : un dock à gauche, des icônes dans la barre du haut. Les variantes officielles proposent d'autres bureaux, sur la même base et avec la même durée de support.

	Ubuntu (GNOME 46)	Kubuntu (KDE Plasma 5.27)	Xubuntu (Xfce 4.18)
Philosophie	Épuré, peu de réglages, tout au clavier	Tout se règle	Léger et classique
Ressources	Moyennes	Moyennes	Faibles
Pour qui	Qui veut un bureau qui s'efface	Qui veut tout adapter	Une machine modeste, un ancien portable

Rien n'oblige à réinstaller pour essayer : `sudo apt install kubuntu-desktop` ou `xubuntu-desktop` ajoute l'autre bureau, qu'on choisit sur l'écran de connexion, par l'icône en bas à droite. Ces paquets amènent aussi les applications de la variante : plusieurs centaines de mégaoctets, et deux logiciels pour chaque usage. Pour un choix définitif, une installation propre de la variante vaut mieux.

Alléger GNOME

GNOME s'enrichit par des **extensions**. Ubuntu en installe quatre d'office : le dock (*Ubuntu Dock*), les icônes de la barre (*AppIndicators*), les icônes du bureau (*Desktop*

Icons NG) et le placement des fenêtres en mosaïque (*Tiling Assistant*). Le logiciel **Gestionnaire d'extensions** (`sudo apt install gnome-shell-extension-manager`) les liste, les active et en cherche d'autres sur `extensions.gnome.org`. **Ajustements** (`gnome-tweaks`) donne accès aux réglages que les Paramètres cachent : polices, comportement des fenêtres, applications au démarrage.

Une extension est du code qui tourne dans le processus même de l'interface. Une extension mal écrite ralentit tout le bureau, et une extension qui n'a pas été mise à jour pour la version de GNOME empêche parfois la session de s'ouvrir. D'où quelques règles :

- n'en installer que pour un besoin précis, et pas plus de quelques-unes ;
- préférer celles qui sont maintenues et compatibles avec GNOME 46 ;
- en cas de lenteur ou de plantage, les désactiver toutes, puis les réactiver une à une.

Quelques réglages valent plus que des extensions : apprendre les raccourcis (Super pour chercher et lancer, Super + flèches pour placer une fenêtre), désactiver les animations dans Accessibilité sur une machine lente, et retirer des applications au démarrage ce qui n'a pas besoin d'y être.

33.2 Le terminal

zsh et starship

Bash est très bien. **zsh** le complète de façon agréable au quotidien : une complétion par menu, un historique partagé entre tous les terminaux ouverts, des corrections de frappe. Il lit les scripts Bash les plus courants sans difficulté.

```
gary@portable:~$ sudo apt install zsh tmux
[…]
gary@portable:~$ zsh --version
zsh 5.9 (x86_64-ubuntu-linux-gnu)
gary@portable:~$ tmux -V
tmux 3.4
```

starship remplace l'invite par une ligne qui montre ce qui compte là où l'on se trouve : la branche Git et son état, la version de Python ou de Node du projet, la durée de la dernière commande si elle a été longue. Il est écrit en Rust, rapide, et fonctionne avec Bash comme avec zsh :

```
gary@portable:~$ curl -sS https://starship.rs/install.sh | sh -s -- -y
[…]
gary@portable:~$ starship --version | head -n 1
starship 1.26.0
gary@portable:~$ sudo chsh -s /usr/bin/zsh gary
gary@portable:~$ getent passwd gary | cut -d: -f7
/usr/bin/zsh
```

Le changement de shell vaut pour les prochaines sessions. Voici le fichier `~/.zshrc` du poste :

```
# ~/.zshrc — le shell du poste de travail

# Historique partagé entre les terminaux, sans doublons
HISTFILE=~/.zsh_history
HISTSIZE=50000
SAVEHIST=50000
setopt share_history hist_ignore_all_dups

# Complétion : Tab propose, les flèches choisissent
autoload -Uz compinit && compinit
zstyle ':completion:*' menu select

# Raccourcis
alias ll='ls -lh --group-directories-first'
alias gs='git status --short'

# Les langages par projet, puis l'invite
eval "$(~/.local/bin/mise activate zsh)"
eval "$(starship init zsh)"
```

Et la configuration de starship, volontairement sobre :

```
# ~/.config/starship.toml — une invite sobre, sur une ligne
add_newline = false
format = "$directory$git_branch$git_status$python$nodejs$cmd_duration$character"

[directory]
truncation_length = 3

[cmd_duration]
min_time = 2_000
format = "[$duration]($style) "
```

Les modules de starship s'appuient sur des symboles que les polices ordinaires n'ont pas toujours. Une police de la famille *Nerd Fonts*, installée dans `~/.local/share/fonts`, les affiche tous.

tmux

tmux découpe un terminal en plusieurs panneaux et, surtout, garde les sessions vivantes quand on ferme la fenêtre ou que la connexion SSH tombe :

```
gary@portable:~$ tmux new-session -d -s travail
gary@portable:~$ tmux list-sessions
travail: 1 windows (created Mon Oct  5 12:28:07 2026)
gary@portable:~$ tmux kill-session -t travail
```

Raccourci	Effet
`Ctrl-b d`	Se détacher de la session, qui continue
`tmux attach -t travail`	La retrouver, même depuis une autre connexion
`Ctrl-b c`, `Ctrl-b n`	Nouvelle fenêtre, fenêtre suivante
`Ctrl-b |`, `Ctrl-b -`	Partager l'écran, avec la configuration ci-dessous

```
# ~/.tmux.conf
set -g mouse on                 # la souris choisit le panneau et fait défiler
set -g history-limit 50000
set -g base-index 1             # fenêtres numérotées à partir de 1
bind | split-window -h -c "#{pane_current_path}"
bind - split-window -v -c "#{pane_current_path}"
```

Sur un serveur, lancer les longues opérations dans tmux évite qu'une coupure de réseau les interrompe.

L'émulateur de terminal

Le Terminal de GNOME suffit à la plupart. D'autres misent sur la vitesse d'affichage par la carte graphique et sur leurs propres onglets et panneaux : **kitty** (0.32 dans les dépôts : `sudo apt install kitty`), **Alacritty** (dans les dépôts aussi), **WezTerm** et **Ghostty**, à installer depuis leurs sites. Le choix est une affaire de goût ; aucun ne change ce que vous tapez.

33.3 L'environnement de développement

Git

Git est installé et configuré comme au chapitre 27. Ses réglages rejoindront les autres dans les dotfiles, plus loin.

Les langages, par projet, avec mise

Ubuntu fournit Python 3.12, et des versions plus anciennes de Node.js et de Go. Un projet en demande une autre, le suivant une autre encore. Installer ces versions au-dessus de celles du système finit toujours par casser quelque chose. **mise** installe chaque version dans le dossier de l'utilisateur, et active la bonne selon le dossier où l'on se trouve. Il succède à asdf, dont il reprend l'idée, en plus rapide.

```
gary@portable:~$ curl -fsSL https://mise.run | sh
[…]
mise: installed successfully to /home/gary/.local/bin/mise
gary@portable:~$ mkdir -p ~/projets/demo && cd ~/projets/demo
gary@portable:~/projets/demo$ ~/.local/bin/mise use python@3.12 node@22 go@1.24 2>&1 | \
    tail -n 2
mise ████████████████ 3/3 · installed 3 tools in 512.3s
mise ~/projets/demo/mise.toml tools: python@3.12.15, node@22.23.3, go@1.24.13
gary@portable:~/projets/demo$ cat mise.toml
[tools]
go = "1.24"
node = "22"
python = "3.12"
gary@portable:~/projets/demo$ ~/.local/bin/mise ls
Tool    Version  Source                  Requested
go      1.24.13  ~/projets/demo/mise.toml  1.24
node    22.23.3  ~/projets/demo/mise.toml  22
python  3.12.15  ~/projets/demo/mise.toml  3.12
```

Le fichier `mise.toml` se range dans le dépôt du projet : quiconque le clone obtient les mêmes versions par `mise install`. Avec la ligne `mise activate` du `.zshrc`, entrer dans le dossier suffit :

```
gary@portable:~$ zsh -ic 'cd ~/projets/demo && python --version && node --version'
Python 3.12.15
v22.23.3
```

Le téléchargement a pris huit minutes et demie, presque tout pour Go, sur une connexion lente. Les langages ne se téléchargent qu'une fois par version. Pour Rust, l'outil officiel `rustup` (`curl --proto '=https' -sSf https://sh.rustup.rs | sh`) fait le même travail que mise, et mise sait aussi s'en servir (`mise use rust`).

> **ATTENTION · Python : ne pas toucher à celui du système**
>
> Ubuntu se sert de son propre Python, dans `/usr/bin/python3`, pour des outils du système. Depuis la 23.04, `pip install` y est refusé (« externally-managed-environment »). Les projets utilisent un environnement virtuel (`python -m venv .venv`) ou le Python de mise ; les outils en ligne de commande écrits en Python s'installent avec `pipx`.

L'éditeur

VS Code se publie sur le dépôt de Microsoft, ce qui garde l'éditeur à jour avec le reste du système. On ajoute le dépôt comme au chapitre 7 :

```
gary@portable:~$ sudo install -m 0755 -d /etc/apt/keyrings
gary@portable:~$ curl -fsSL https://packages.microsoft.com/keys/microsoft.asc | \
    sudo tee /etc/apt/keyrings/microsoft.asc > /dev/null
gary@portable:~$ printf '%s\n' 'Types: deb' 'URIs: https://packages.microsoft.com/repos/
code' \
    'Suites: stable' 'Components: main' 'Signed-By: /etc/apt/keyrings/microsoft.asc' | \
    sudo tee /etc/apt/sources.list.d/vscode.sources
Types: deb
URIs: https://packages.microsoft.com/repos/code
Suites: stable
Components: main
Signed-By: /etc/apt/keyrings/microsoft.asc
gary@portable:~$ sudo apt update
[…]
Get:5 https://packages.microsoft.com/repos/code stable/main amd64 Packages [31.0 kB]
gary@portable:~$ apt-cache policy code | head -n 3
code:
  Installed: (none)
  Candidate: 1.140.0-1790759618
gary@portable:~$ sudo apt install code
```

Le fichier `.sources` est le format moderne des dépôts d'APT, celui qu'Ubuntu 24.04 utilise pour ses propres dépôts. Cursor, éditeur dérivé de VS Code et centré sur l'assistance par intelligence artificielle, se télécharge sur son site en `.deb` ou en AppImage. Pour qui préfère le terminal, Vim et Neovim (chapitre 6) restent des éditeurs complets.

Les conteneurs de développement

Un projet qui demande une base de données, un service de messages ou une distribution précise ne s'installe pas sur le poste : il tourne dans des conteneurs, décrits par un fichier Compose, comme au chapitre 28. VS Code sait ouvrir un projet directement dans un conteneur (*Dev Containers*). **Distrobox** (`sudo apt install distrobox`, version 1.7) fait tourner une autre distribution dans un conteneur parfaitement intégré au bureau : un Fedora ou un Arch dont les programmes s'ouvrent comme des applications d'Ubuntu.

33.4 Mots de passe et fichiers

KeePassXC

KeePassXC garde les mots de passe dans un fichier chiffré, que l'on copie et synchronise où l'on veut. Il remplit les formulaires des navigateurs par son extension, et s'utilise aussi en ligne de commande :

```
gary@portable:~$ sudo apt install keepassxc
[…]
gary@portable:~$ keepassxc-cli db-create --set-password ~/coffre.kdbx
Enter password to encrypt database (optional):
Repeat password:
Successfully created new database.
gary@portable:~$ keepassxc-cli add --username gary --url https://cloud.maison.lan \
    --generate --length 24 ~/coffre.kdbx Nextcloud
Enter password to unlock /home/gary/coffre.kdbx:
Successfully added entry Nextcloud.
gary@portable:~$ keepassxc-cli show ~/coffre.kdbx Nextcloud
Enter password to unlock /home/gary/coffre.kdbx:
Title: Nextcloud
UserName: gary
Password: PROTECTED
URL: https://cloud.maison.lan
[…]
```

Le mot de passe de 24 caractères a été tiré au hasard, et l'outil refuse de l'afficher sans qu'on le demande (`show -s`). Si vous avez installé le Vaultwarden du chapitre 32, ses applications font le même travail, avec la synchronisation en plus ; KeePassXC convient à qui préfère un simple fichier.

Syncthing

Syncthing synchronise des dossiers entre vos machines, directement de l'une à l'autre, sans serveur central ni compte. Chaque appareil a un identifiant, qu'on échange pour les relier :

```
gary@portable:~$ sudo apt install syncthing
[…]
gary@portable:~$ syncthing --version
syncthing v1.27.2-ds4 "Gold Grasshopper" (go1.22.2 linux-amd64) debian@debian
  2025-07-21 04:55:34 UTC
gary@portable:~$ syncthing generate --home ~/.local/state/syncthing 2>&1 | \
    grep 'Device ID'
2026/10/05 13:12:03 INFO: Device ID:
  NRQZ2WA-2VSTLDL-CCKDHUR-CRVPYUV-ZQT4EKC-2ISQ6U5-DHB4WWZ-BENYXAS
gary@portable:~$ systemctl --user enable --now syncthing.service
```

Le paquet fournit un service utilisateur systemd : Syncthing démarre avec la session. Son interface web, sur `http://127.0.0.1:8384`, ajoute les autres appareils et les dossiers à partager. Syncthing n'est pas une sauvegarde : un fichier effacé sur une machine l'est aussi sur les autres.

Tailscale

Pour joindre le serveur du chapitre 32 ou un autre poste depuis n'importe où, **Tailscale** monte un réseau WireGuard entre vos appareils sans configuration. Il s'installe par le dépôt de son éditeur, puis `sudo tailscale up` ouvre une page de connexion. Le chapitre 32 montre la version sans intermédiaire, avec WireGuard seul.

33.5 Sauvegarder le poste

Deux sauvegardes, pour deux risques différents.

Timeshift photographie le système : `/etc`, les programmes, la configuration. Après une mise à jour qui a mal tourné, on revient à la photo de la veille. Il est dans les dépôts (`sudo apt install timeshift`, version 24.01) et se règle par son interface : le mode `rsync`, un disque de destination, la fréquence (quotidienne, cinq photos gardées est un bon début). Par défaut, il laisse de côté les dossiers personnels : ce n'est pas son rôle.

restic, comme au chapitre 16, sauvegarde les fichiers personnels vers un disque externe ou le serveur du chapitre 32 :

```
gary@portable:~$ sudo apt install restic
[…]
gary@portable:~$ restic init --repo /media/gary/SAUVEGARDE/portable
enter password for new repository:
enter password again:
created restic repository 02bfd335dc at /media/gary/SAUVEGARDE/portable
[…]
gary@portable:~$ restic -r /media/gary/SAUVEGARDE/portable backup ~ \
    --exclude ~/.cache --exclude-caches -q
enter password for repository:
gary@portable:~$ restic -r /media/gary/SAUVEGARDE/portable snapshots
enter password for repository:
repository 02bfd335 opened (version 2, compression level auto)
ID        Time                 Host        Tags        Paths
----------------------------------------------------------------
ca1a03be  2026-10-05 13:12:14  portable                /home/gary
----------------------------------------------------------------
1 snapshots
```

`--exclude-caches` laisse de côté les dossiers qui se déclarent caches, comme ceux des navigateurs. Une minuterie systemd utilisateur, sur le modèle du chapitre 16, lance la sauvegarde chaque jour ; un disque USB qu'on branche le soir, ou le serveur personnel joint par SFTP, en est la destination. Pour un portable souvent éteint, `Persistent=true` rattrape les passages manqués.

33.6 Les réglages, versionnés

Les fichiers de configuration du dossier personnel, les **dotfiles**, représentent des années de petits réglages. Les ranger dans un dépôt Git les met à l'abri, et les réinstalle en une minute sur une nouvelle machine. **GNU Stow** fait le lien : chaque logiciel a son dossier dans le dépôt, et Stow pose dans le dossier personnel des liens vers ces fichiers.

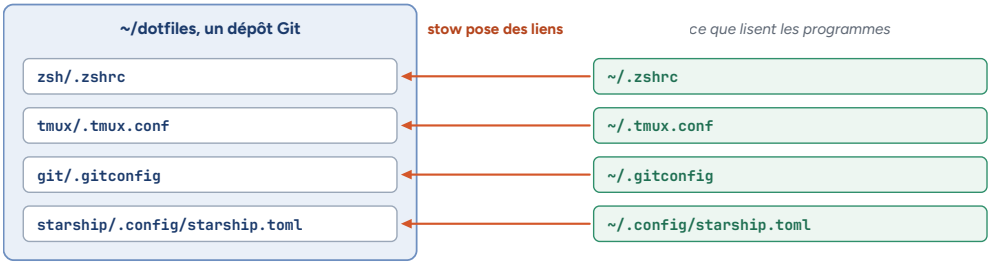


Figure 33.1 — Stow relie chaque fichier du dossier personnel à sa version dans le dépôt.

```
gary@portable:~$ sudo apt install stow
[…]
gary@portable:~$ mkdir -p ~/dotfiles/{zsh,tmux,git,starship/.config}
gary@portable:~$ mv ~/.zshrc ~/dotfiles/zsh/.zshrc
gary@portable:~$ mv ~/.tmux.conf ~/dotfiles/tmux/.tmux.conf
gary@portable:~$ mv ~/.gitconfig ~/dotfiles/git/.gitconfig
gary@portable:~$ mv ~/.config/starship.toml ~/dotfiles/starship/.config/starship.toml
gary@portable:~$ cd ~/dotfiles
gary@portable:~/dotfiles$ find . -type f | sort
./git/.gitconfig
./starship/.config/starship.toml
./tmux/.tmux.conf
./zsh/.zshrc
gary@portable:~/dotfiles$ stow zsh tmux git starship
gary@portable:~/dotfiles$
ls -l ~/.zshrc ~/.tmux.conf ~/.gitconfig ~/.config/starship.toml | \
    cut -c 1-10,40-
lrwxrwxrwx /home/gary/.config/starship.toml -> ../dotfiles/starship/.config/starship.toml
lrwxrwxrwx /home/gary/.gitconfig -> dotfiles/git/.gitconfig
lrwxrwxrwx /home/gary/.tmux.conf -> dotfiles/tmux/.tmux.conf
lrwxrwxrwx /home/gary/.zshrc -> dotfiles/zsh/.zshrc
```

```
gary@portable:~/dotfiles$ git init -q
gary@portable:~/dotfiles$ git add .
gary@portable:~/dotfiles$ git commit -qm "Mes réglages : zsh, tmux, git, starship"
```

Dans chaque dossier, l'arborescence reproduit celle du dossier personnel : `starship/.config/starship.toml` devient `~/.config/starship.toml`. Les programmes lisent les liens comme des fichiers ordinaires, et toute modification atterrit dans le dépôt, prête à être enregistrée.

Sur une nouvelle machine, on clone et l'on relance Stow. Essayons, après avoir retiré les liens (`stow -D`) :

```
gary@portable:~$ stow -d ~/dotfiles -D zsh tmux git starship
gary@portable:~$ ls ~/.zshrc 2>&1
ls: cannot access '/home/gary/.zshrc': No such file or directory
gary@portable:~$ git clone -q ~/dotfiles ~/dotfiles-clone
gary@portable:~$ stow -d ~/dotfiles-clone zsh tmux git starship
gary@portable:~$ readlink ~/.zshrc
dotfiles-clone/zsh/.zshrc
gary@portable:~$ git config --global user.name
Gary Nkweka
```

Le clone viendrait, en vrai, du Forgejo du chapitre 32 ou d'une plateforme. **chezmoi** va plus loin que Stow : il gère des modèles pour les réglages qui diffèrent d'une machine à l'autre, et le chiffrement des fichiers sensibles. Dans les deux cas, la règle du chapitre 27 vaut : pas de clé privée ni de mot de passe dans le dépôt.

33.7 D'où installer quoi

Un logiciel peut souvent venir de plusieurs sources. Le choix n'est pas anodin : il décide des mises à jour, du confinement et de la place occupée.

Source	Pour quoi	Mises à jour	Remarque
APT, dépôts d'Ubuntu	Les outils du système, la ligne de commande	Avec le système, correctifs de sécurité	Versions figées pendant la vie de la 24.04
Dépôt d'éditeur (APT)	VS Code, Docker, Tailscale	Avec le système	Une clé et un fichier `.sources` par éditeur
Snap	Firefox, Thunderbird, applications de l'App Center	Automatiques, en arrière-plan	Confinées ; installées d'office sur Ubuntu

Source	Pour quoi	Mises à jour	Remarque
Flatpak	Applications de bureau récentes, depuis Flathub	`flatpak update`	Confinées ; non intégré à l'App Center
`.deb` téléchargé	Logiciel absent des dépôts	Aucune, sauf s'il ajoute son dépôt	Le dernier recours
mise, rustup, pipx	Langages et outils de développement	Par l'outil	Dans le dossier de l'utilisateur
Conteneurs	Bases de données, services, autres distributions	En tirant l'image	Rien n'est installé sur le poste

Snap est intégré d'office à Ubuntu : Firefox et l'App Center en viennent. Flatpak s'ajoute en deux commandes, et ouvre Flathub, le plus grand catalogue d'applications de bureau pour Linux :

```
gary@portable:~$ sudo apt install flatpak
[…]
gary@portable:~$ flatpak --version
Flatpak 1.14.6
gary@portable:~$ flatpak remote-add --user --if-not-exists flathub \
    https://dl.flathub.org/repo/flathub.flatpakrepo
gary@portable:~$ flatpak remote-info --user flathub org.keepassxc.KeePassXC 2>&1 | \
    grep -E 'ID:|Download:|Installed:|Runtime:'
        ID: org.keepassxc.KeePassXC
  Download: 22.9 MB
 Installed: 51.2 MB
   Runtime: org.kde.Platform/x86_64/5.15-25.08
```

L'application KeePassXC de Flathub est plus récente que celle des dépôts, mais elle demande l'environnement d'exécution de KDE : la première application d'une famille coûte cher, les suivantes partagent l'environnement. `flatpak install --user flathub org.keepassxc.KeePassXC` l'installerait. Une règle simple : APT d'abord pour ce qui est en ligne de commande, Snap ou Flatpak pour les applications de bureau qu'on veut récentes, et jamais le même logiciel par deux sources à la fois.

En résumé

- GNOME, KDE ou Xfce : même Ubuntu, même support. Peu d'extensions, bien choisies.

- zsh pour le confort, starship pour l'invite, tmux pour des sessions qui survivent.
- mise installe les langages par projet, décrits dans `mise.toml`. Le Python du système reste au système.
- VS Code s'installe par le dépôt de Microsoft, au format `.sources`.
- KeePassXC pour les mots de passe, Syncthing pour synchroniser, Tailscale ou WireGuard pour joindre ses machines.
- Timeshift photographie le système, restic sauvegarde les fichiers personnels.
- Les dotfiles vivent dans un dépôt Git, posés par Stow ; une nouvelle machine les retrouve en une commande.
- Chaque logiciel a sa source : APT, dépôt d'éditeur, Snap, Flatpak, mise ou conteneur.

Atelier — Votre poste

Objectif : faire de votre Ubuntu un poste de travail reproductible. Comptez une demi-journée.

Le bureau

1. Faites l'inventaire des extensions GNOME actives. Désactivez celles dont vous ne vous servez pas, et notez le temps d'ouverture de session avant et après.
2. Installez Timeshift, configurez une photo quotidienne sur un second disque ou une partition, et prenez-en une à la main.

Le terminal et le code

3. Installez zsh, starship et tmux. Faites de zsh votre shell.
4. Installez mise. Dans un projet, fixez une version de Python et une de Node.js, et vérifiez qu'elles s'activent en entrant dans le dossier.
5. Installez VS Code par le dépôt de Microsoft.

Les données

6. Créez un coffre KeePassXC et rangez-y trois mots de passe. Sauvegardez le fichier.
7. Installez restic et sauvegardez votre dossier personnel vers un disque externe, en excluant les caches. Restaurez un fichier.

La reproductibilité

8. Rangez vos dotfiles dans `~/dotfiles`, un dossier par logiciel, posez-les avec Stow, et faites-en un dépôt Git poussé sur votre Forgejo ou une plateforme.

Vérifier

- Dans un nouvel utilisateur ou une machine virtuelle, `git clone` puis `stow` redonnent votre terminal tel que vous l'aimez.
- `mise ls` dans le projet montre les versions fixées.
- Timeshift et restic ont chacun au moins une sauvegarde, et vous avez restauré un fichier.

Exercices

Exercice 1 — Quelle source ?

D'où installeriez-vous chacun de ces logiciels, et pourquoi ? GIMP, `htop`, PostgreSQL pour un projet, Node.js 20 pour un projet ancien, Docker, Slack, `ripgrep`.

Exercice 2 — Un conflit de versions

Un projet exige Python 3.11, un autre Python 3.13. Écrivez les commandes mise pour chacun, et montrez que les deux coexistent.

Exercice 3 — Un alias de plus

Ajoutez à votre `.zshrc` un alias qui crée et active un environnement virtuel Python dans le dossier courant. Enregistrez la modification dans le dépôt des dotfiles. Pourquoi n'avez-vous pas eu à relancer Stow ?

Exercice 4 — La sauvegarde automatique

Écrivez un service et une minuterie systemd **utilisateur** (dans `~/.config/systemd/user/`) qui lancent la sauvegarde restic chaque jour à 19 h, avec `Persistent=true`. Où ranger le mot de passe du dépôt ? Comment vérifier que la minuterie est active ?

Exercice 5 — Le poste perdu

Votre portable est volé. Listez, dans l'ordre, ce que vous faites dans l'heure qui suit, puis ce qu'il vous faut pour retrouver un poste identique sur une nouvelle machine. Qu'est-ce qui, dans ce chapitre et le précédent, rend cette reconstruction possible ?

Pour aller plus loin

Sur la machine. `man zshoptions` décrit les options de zsh, `man tmux` toutes les commandes de tmux. `starship explain` dit, dans un dossier, d'où vient chaque morceau de l'invite. `mise doctor` vérifie l'installation de mise.

En ligne. La documentation de mise, `mise.jdx.dev`, liste les langages et outils qu'il sait installer. Celle de starship, `starship.rs`, présente tous les modules et des configurations toutes faites. Le guide de Syncthing, `docs.syncthing.net`, explique ses choix de sécurité. Flathub, `flathub.org`, indique pour chaque application si elle est publiée par ses auteurs eux-mêmes (« vérifiée »).

RÉFÉRENCES

Annexes

Des pages à garder sous la main : les commandes du livre, la référence de systemd, le vocabulaire, les ressources pour aller plus loin et l'installation express.

Annexe A — Aide-mémoire des commandes

Les commandes du livre, classées par usage, avec le chapitre qui les explique. Chacune a été employée sur Ubuntu 24.04 dans les chapitres cités. Le reste se trouve dans `man commande` et `commande --help`.

Se repérer et obtenir de l'aide

Commande	Effet	Ch.
`man ls`, `man 5 fstab`	Le manuel d'une commande, d'un fichier	3
`commande --help`	L'aide courte	3
`apropos mot`	Chercher une commande par mot-clé	3
`type cmd`, `which cmd`	D'où vient une commande	8
`lsb_release -a`, `cat /etc/os-release`	La version d'Ubuntu	2, 3
`uname -r`	La version du noyau	11
`hostnamectl`	Nom de la machine, système, matériel	18
`history`, Ctrl+R	L'historique, la recherche dans l'historique	5, 8

Fichiers et dossiers

Commande	Effet	Ch.
`pwd`, `cd dossier`, `cd -`	Où suis-je, aller, revenir	3, 4
`ls -lah`	Lister, détails, fichiers cachés, tailles lisibles	5
`tree -L 2`	L'arborescence sur deux niveaux	4
`cp -a source dest`	Copier en gardant droits et dates	5

Commande	Effet	Ch.
`mv ancien nouveau`	Déplacer ou renommer	5
`rm fichier`, `rm -r dossier`	Supprimer, sans corbeille	5
`mkdir -p a/b/c`	Créer, parents compris	5
`ln -s cible lien`	Lien symbolique	4
`find /etc -name '*.conf' -mmin -60`	Chercher par nom, par date	5
`du -sh dossier`, `ncdu`	La place occupée	5
`df -h`, `df -i`	La place libre, les inodes	14
`stat fichier`	Toutes les métadonnées	4
`file fichier`	La nature d'un fichier	4
`tar -czf a.tar.gz dossier`, `tar -xf a.tar.gz`	Archiver, extraire	5, 16

Lire et transformer du texte

Commande	Effet	Ch.
`less fichier`	Lire page par page (/ cherche, q quitte)	4
`head -n 20`, `tail -n 20`, `tail -f`	Le début, la fin, suivre en direct	3, 5
`grep -r motif dossier`	Chercher dans tout un dossier	9
`grep -E`, `grep -v`	Expression étendue, lignes qui ne correspondent pas	9
`cut -d: -f1`, `awk '{print $1}'`	Extraire des colonnes	5, 9
`sort`, `sort -hr`, `uniq -c`	Trier, trier des tailles du plus grand au plus petit, compter les doublons	5, 9
`sed 's/ancien/nouveau/'`, `sed -i`	Remplacer, dans le fichier	9
`wc -l`	Compter les lignes	9
`diff -u a b`	Les différences entre deux fichiers	9
`nano`, `vim`	Éditer	3

Droits et utilisateurs

Commande	Effet	Ch.
`id`, `id -nG`	Mon compte, mes groupes	6
`chmod 640 f`, `chmod u+x f`	Changer les droits	6
`chown gary:gary f`	Changer le propriétaire	6
`sudo commande`, `sudo -l`	Agir en root, ce que j'ai le droit de faire	5, 20
`sudo visudo -f /etc/sudoers.d/nom`	Donner un droit précis, avec vérification	6, 20
`sudo adduser alice`, `sudo usermod -aG groupe alice`	Créer un compte, l'ajouter à un groupe	6
`sudo passwd -l alice`, `sudo chage -l alice`	Verrouiller, voir l'expiration	20
`getfacl f`, `setfacl -R -m u:max:rX dossier`	Droits étendus (ACL)	6

Paquets

Commande	Effet	Ch.
`sudo apt update`	Relire les dépôts	7
`sudo apt upgrade`, `sudo apt full-upgrade`	Mettre à jour	7
`sudo apt install p`, `sudo apt remove p`, `sudo apt purge p`	Installer, retirer, retirer avec la configuration	7
`apt search mot`, `apt show p`, `apt-cache policy p`	Chercher, décrire, versions disponibles	7
`apt list --installed`, `dpkg -S /chemin`, `sudo dpkg -i f.deb`	Installés, paquet d'un fichier, installer un `.deb`	7
`sudo apt autoremove`	Retirer les dépendances devenues inutiles	7
`snap list`, `sudo snap refresh`	Les snaps, leur mise à jour	3, 7
`flatpak install --user flathub app`	Une application de Flathub	33

Commande	Effet	Ch.
`cat /var/run/reboot-required`	Un redémarrage est-il attendu ?	20

Processus

Commande	Effet	Ch.
`ps aux`, `ps -ef --forest`	Tous les processus, en arbre	12
`top`, `htop`, `btop`	En direct	12, 26
`pgrep -a nom`	Trouver un processus	12
`kill PID`, `kill -9 PID`, `pkill nom`	Signaler, tuer	12
`nice -n 10 cmd`, `renice 10 -p PID`	Priorité	12
`jobs`, `fg`, `bg`, `commande &`, `nohup`	Tâches du shell	12
`uptime`, `nproc`	Charge, nombre de processeurs	12, 26
`free -h`, `vmstat 2`	Mémoire	26
`iostat -dxs 2`, `iotop`	Disques	26
`sar`, `sar -r`, `sar -d`	Historique	26
`strace -f -e trace=openat cmd`	Appels système	30
`lsof -i :443`, `lsof +L1`	Fichiers et ports ouverts	30

systemd et journaux

Commande	Effet	Ch.
`systemctl status s`	État d'un service	12
`sudo systemctl start s`, `stop`, `restart`, `reload`	Le piloter	12
`sudo systemctl enable --now s`, `disable --now`	Au démarrage, et tout de suite	12

Commande	Effet	Ch.
`systemctl --failed`, `systemctl list-timers`	Ce qui a échoué, les minuteries	11, 13
`systemctl cat s`, `sudo systemctl edit s`	Voir l'unité, la surcharger	12
`sudo systemctl daemon-reload`	Relire les unités modifiées	12
`systemd-analyze blame`, `systemd-analyze security s`	Démarrage, exposition d'un service	11, 12
`journalctl -u s -f`	Journal d'un service, en direct	13
`journalctl -p err -b`, `journalctl -b -1`	Erreurs depuis le démarrage, démarrage précédent	13, 30
`journalctl -k`, `dmesg -T`	Messages du noyau	13, 30
`systemd-run --scope -p MemoryMax=200M cmd`	Lancer sous limite	12, 30
`timedatectl`, `localectl`, `hostnamectl`	Heure, langue, nom	18, 32

Disques et stockage

Commande	Effet	Ch.
`lsblk -f`, `blkid`	Disques, partitions, systèmes de fichiers	14
`sudo fdisk -l`, `parted`	Partitionner	14
`sudo mkfs.ext4 -L NOM /dev/sdX1`	Formater	14
`sudo mount /dev/sdX1 /mnt`, `findmnt`	Monter, voir les montages	14
`sudo mount -a` après modification de `/etc/fstab`	Vérifier `fstab` sans redémarrer	14
`pvs`, `vgs`, `lvs`	LVM : état	15
`sudo lvextend -r -L +5G vg/lv`	Agrandir un volume et son système de fichiers	15
`cat /proc/mdstat`, `sudo mdadm --detail /dev/md0`	RAID logiciel	15, 31

Commande	Effet	Ch.
`sudo cryptsetup open /dev/sdX nom`	Ouvrir un volume chiffré	15
`sudo e2fsck -f /dev/sdX1` (démonté)	Vérifier un ext4	14, 31
`sudo smartctl -a /dev/sda`	Santé d'un disque	31
`restic backup`, `restic snapshots`, `restic restore`	Sauvegarder, lister, restaurer	16

Réseau

Commande	Effet	Ch.
`ip -br address`, `ip route`, `ip neigh`	Adresses, routes, voisins	17
`ping -c 3 hôte`, `mtr -r hôte`	Joindre, suivre le trajet	17, 30
`ss -tlnp`, `ss -tunp`	Ports en écoute, connexions	17, 26
`dig nom`, `dig @serveur nom`, `resolvectl status`	DNS	17, 30
`curl -I https://site`	Interroger un serveur web	17, 23
`sudo netplan try`, `sudo netplan apply`	Appliquer la configuration réseau	18
`nmcli device`, `nmcli connection`	NetworkManager, sur un poste	18
`iperf3 -s`, `iperf3 -c hôte`	Mesurer le débit	30
`sudo tcpdump -ni enp0s3 port 53`	Voir passer les paquets	17

SSH

Commande	Effet	Ch.
`ssh-keygen -t ed25519`	Créer une paire de clés	19
`ssh-copy-id serveur`	Installer sa clé sur un serveur	19
`ssh serveur`, `ssh -J rebond serveur`	Se connecter, par un rebond	19
`scp f serveur:/tmp/`, `rsync -avh src/ serveur:dest/`	Copier	16, 19

Commande	Effet	Ch.
`ssh -f -N -L 8080:localhost:8000 serveur`	Tunnel vers un service interne	19, 26
`eval "$(ssh-agent)"`, `ssh-add`	Garder la clé déverrouillée	19
`sudo sshd -t`	Vérifier la configuration avant de recharger	19

Sécurité

Commande	Effet	Ch.
`sudo ufw status verbose`, `sudo ufw allow OpenSSH`, `sudo ufw enable`	Pare-feu	21
`sudo fail2ban-client status sshd`	Adresses bannies	21
`sudo aa-status`	Profils AppArmor	22
`sudo lynis audit system --quick`	Audit de durcissement	20
`sudo auditctl -l`, `sudo ausearch -k clé -i`	Règles et événements d'audit	20
`openssl x509 -in cert.pem -noout -dates`	Dates d'un certificat	23

Conteneurs, Git, Ansible

Commande	Effet	Ch.
`docker ps`, `docker logs -f c`, `docker exec -it c sh`	Conteneurs	28
`docker compose up -d`, `docker compose ps`, `docker compose pull`	Une application Compose	28, 32
`docker system df`, `docker image prune`	La place, le ménage	28
`git status`, `git add`, `git commit -m`, `git log --oneline`	Le cycle de Git	27
`git switch -c branche`, `git merge`, `git pull --rebase`, `git push`	Branches et dépôts distants	27

Commande	Effet	Ch.
`ansible all -m ping`, `ansible-playbook p.yml -K --check --diff`	Ansible	29
`ansible-vault create f.yml`, `--ask-vault-pass`	Secrets chiffrés	29

Annexe B — Référence systemd

Les directives des fichiers d'unité que l'on écrit le plus souvent, section par section, et un service durci commenté de bout en bout. Toutes les directives citées ont été acceptées par le systemd 255 d'Ubuntu 24.04, et une même unité d'essai les a toutes portées à la fois. La liste complète est dans `man systemd.directives`, qui renvoie pour chacune à sa page de manuel.

Où vivent les unités

Dossier	Contenu
`/usr/lib/systemd/system`	Les unités livrées par les paquets. On n'y touche pas : une mise à jour les remplace
`/etc/systemd/system`	Les unités de l'administrateur, et ses surcharges. Prioritaire sur les paquets
`/run/systemd/system`	Les unités temporaires, perdues au redémarrage
`~/.config/systemd/user`	Les unités d'un utilisateur, pilotées par `systemctl --user`

Sur Ubuntu 24.04, `/lib` est un lien vers `usr/lib` : `/lib/systemd/system` désigne le même dossier. L'ordre exact de recherche, du plus prioritaire au moins prioritaire :

```
gary@serveur:~$ systemd-analyze unit-paths
/etc/systemd/system.control
/run/systemd/system.control
/run/systemd/transient
/run/systemd/generator.early
/etc/systemd/system
/etc/systemd/system.attached
/run/systemd/system
/run/systemd/system.attached
/run/systemd/generator
```

```
/usr/local/lib/systemd/system
/usr/lib/systemd/system
/run/systemd/generator.late
```

Une **surcharge** (*drop-in*) modifie une unité sans la recopier : un fichier `.conf` dans un dossier `nom.service.d/`, qui ne contient que les lignes à changer, sous leur section. `systemctl edit` le crée (chapitre 12).

Commande	Effet
`systemctl cat s`	L'unité et ses surcharges, telles que systemd les lit
`sudo systemctl edit s`	Créer ou modifier la surcharge `override.conf`
`sudo systemctl edit --full s`	Copier l'unité entière dans `/etc/systemd/system` pour la modifier
`sudo systemctl revert s`	Supprimer surcharges et copies, revenir à l'unité du paquet
`systemctl show -p FragmentPath -p DropInPaths s`	Le fichier lu et ses surcharges
`systemd-delta`	Toutes les unités surchargées de la machine
`sudo systemctl daemon-reload`	Relire les fichiers après une modification à la main

Dans une surcharge, une liste se vide par une affectation vide : `ExecStart=` seul, puis la nouvelle ligne `ExecStart=`. Un commentaire occupe toujours une ligne entière : systemd ne reconnaît pas le `#` en fin de ligne.

La section [Unit]

Elle décrit l'unité et ses liens avec les autres, quel que soit son type.

Directive	Effet
`Description=`	Le nom affiché par `systemctl status` et le journal
`Documentation=`	Des références : `man:nginx(8)`, `https://…`
`Wants=`	Démarrer aussi ces unités ; leur échec n'empêche rien

Directive	Effet
`Requires=`	Démarrer aussi ces unités ; si l'une échoue ou s'arrête, celle-ci aussi
`BindsTo=`	Comme `Requires=`, et s'arrêter dès que l'autre disparaît
`PartOf=`	Être arrêtée et redémarrée avec l'autre unité
`Conflicts=`	Ne jamais tourner en même temps que ces unités
`After=`, `Before=`	L'ordre seulement : démarrer après ou avant. Ne crée aucune dépendance
`OnFailure=`, `OnSuccess=`	Lancer une unité quand celle-ci échoue ou réussit (chapitre 16)
`ConditionPathExists=`	Ne démarrer que si le chemin existe ; sinon, sauter sans erreur
`AssertPathExists=`	Même test, mais son échec est une erreur
`StartLimitIntervalSec=`, `StartLimitBurst=`	Abandonner après tant de démarrages dans tel intervalle

`Wants=` et `After=` vont presque toujours ensemble : l'un dit *quoi*, l'autre *dans quel ordre*.

La section [Service] : lancer et arrêter

Le type

`Type=`	systemd considère le service démarré…	Pour
`simple` (défaut)	dès que le processus est créé	Un programme qui reste au premier plan
`exec`	dès que le programme est effectivement lancé	Le même, avec un échec signalé si le programme est introuvable
`forking`	quand le processus lancé s'est dédoublé et a rendu la main	Les démons à l'ancienne ; avec `PIDFile=`
`oneshot`	quand la commande est terminée	Une tâche ponctuelle : sauvegarde, réglage au démarrage

Type=	systemd considère le service démarré…	Pour
`notify`	quand le programme le lui signale	Les programmes qui savent parler à systemd
`notify-reload`	comme `notify`, et le rechargement se fait par un signal	Idem
`dbus`	quand le nom D-Bus indiqué apparaît	Les services du bureau
`idle`	comme `simple`, une fois les autres démarrages terminés	Rarement utile

Les commandes

Directive	Effet
`ExecStart=`	La commande principale. Chemin absolu ; plusieurs lignes possibles avec `oneshot`
`ExecStartPre=`, `ExecStartPost=`	Avant, après le démarrage
`ExecReload=`	Ce que fait `systemctl reload`, souvent `/bin/kill -HUP $MAINPID`
`ExecStop=`	Arrêter autrement que par un signal
`ExecStopPost=`	Après l'arrêt, même en cas d'échec : nettoyage
`RemainAfterExit=yes`	Rester « actif » après la fin de la commande (`oneshot`)

Une commande n'est pas passée à un shell : ni redirection, ni tube, ni `*`. Il faut écrire `/bin/sh -c '…'` si l'on en a besoin. Les variables `$VAR` et `${VAR}` sont remplacées par systemd. Un caractère devant le chemin change le comportement :

Préfixe	Effet
`-`	Un échec de cette commande est ignoré
`+`	Lancer cette commande avec tous les privilèges, hors cloisonnement
`@`	Le deuxième mot devient le nom du processus (`argv[0]`)
`:`	Ne pas remplacer les variables

Redémarrer, attendre, tuer

Directive	Effet
`Restart=`	Quand redémarrer (tableau suivant)
`RestartSec=`	Attendre avant de redémarrer (100 ms par défaut)
`RestartPreventExitStatus=`	Codes de sortie qui interdisent le redémarrage
`SuccessExitStatus=`	Codes ou signaux à compter comme une réussite
`TimeoutStartSec=`, `TimeoutStopSec=`	Délai accordé au démarrage, à l'arrêt (90 s par défaut)
`RuntimeMaxSec=`	Durée de vie maximale ; au-delà, le service est arrêté
`WatchdogSec=`	Le programme doit donner signe de vie à ce rythme (`notify`)
`KillMode=`	Qui reçoit le signal d'arrêt : `control-group` (défaut, tous les processus), `mixed`, `process`
`KillSignal=`	Le signal d'arrêt (`SIGTERM` par défaut)

Selon la cause de l'arrêt, `Restart=` relance (×) ou non :

Cause	`no`	`always`	`on-success`	`on-failure`	`on-abnormal`	`on-abort`	`on-watchdog`
Sortie propre		×	×				
Code non nul		×		×			
Signal anormal		×		×	×	×	
Délai dépassé		×		×	×		
Chien de garde		×		×	×		×

Une sortie propre est un code 0, ou l'un des signaux `SIGHUP`, `SIGINT`, `SIGTERM` et `SIGPIPE`, ou ce que liste `SuccessExitStatus=`. `on-failure` convient à presque tous les services. Un service `oneshot` refuse `always` et `on-success`.

L'environnement d'exécution

Ces directives se placent dans `[Service]` ; elles valent aussi pour les autres unités qui lancent des processus.

Directive	Effet
`User=`, `Group=`	Le compte du service. systemd avertit si c'est `nobody`, partagé par trop de programmes
`DynamicUser=yes`	Un compte créé au démarrage et supprimé à l'arrêt
`SupplementaryGroups=`	Des groupes en plus
`WorkingDirectory=`	Le dossier courant
`Environment=`	Des variables : `Environment=LANG=C.UTF-8 MODE=prod`
`EnvironmentFile=`	Les lire dans un fichier ; `-` devant le chemin le rend facultatif
`LoadCredential=nom:chemin`	Transmettre un secret par un fichier privé (ci-dessous)
`StandardOutput=`, `StandardError=`	Où vont les sorties : `journal` (défaut), `null`, `append:/chemin`
`SyslogIdentifier=`	Le nom sous lequel le service écrit au journal
`UMask=`	Les droits retirés aux fichiers créés : `0077` les rend privés
`Nice=`	La priorité (chapitre 12)
`LimitNOFILE=`	Le nombre de fichiers ouverts à la fois

Plutôt que de créer ses dossiers à la main, le service les demande à systemd, qui les crée au démarrage et les donne à son compte :

Directive	Crée
`StateDirectory=nom`	`/var/lib/nom`, les données
`CacheDirectory=nom`	`/var/cache/nom`, ce qui peut être effacé
`LogsDirectory=nom`	`/var/log/nom`
`RuntimeDirectory=nom`	`/run/nom`, effacé à l'arrêt
`ConfigurationDirectory=nom`	`/etc/nom`

Avec `ProtectSystem=strict`, ces dossiers restent les seuls où le service peut écrire, avec ceux de `ReadWritePaths=`.

Un mot de passe placé dans `Environment=` se lit dans `systemctl show` et dans `/proc`. `LoadCredential=` le transmet par un fichier que seul le service peut lire, dont le dossier est dans `$CREDENTIALS_DIRECTORY`. La source peut rester lisible par root seul, même pour un compte dynamique :

```
gary@serveur:~$ sudo ls -l /etc/tableau/jeton
-rw------- 1 root root 23 Oct  5 13:50 /etc/tableau/jeton
gary@serveur:~$ sudo systemd-run --wait --pipe -q -p DynamicUser=yes \
    -p LoadCredential=jeton:/etc/tableau/jeton \
    sh -c 'id -un; ls -l "$CREDENTIALS_DIRECTORY"; cat "$CREDENTIALS_DIRECTORY/jeton"'
run-u67
total 4
-r--r-----+ 1 root root 23 Oct  5 13:51 jeton
jeton-de-demonstration
```

Le `+` après les droits signale une ACL, qui ouvre la lecture au seul compte `run-u67`.

Les ressources

Directive	Effet
`MemoryHigh=`	Au-delà, le service est ralenti et sa mémoire récupérée
`MemoryMax=`	Limite stricte : au-delà, le noyau tue un processus du service
`CPUQuota=`	Part d'un processeur : `50%` ; `200%` pour deux processeurs
`CPUWeight=`	Poids relatif en cas de concurrence ; le noyau compte 100 si rien n'est réglé
`IOWeight=`	Idem pour les disques
`TasksMax=`	Nombre maximal de processus et de fils d'exécution
`IPAddressAllow=`, `IPAddressDeny=`	Les adresses que le service peut joindre ou qui peuvent le joindre ; `IPAddressDeny=any` puis `IPAddressAllow=localhost` le limite à la machine

`systemctl status` affiche la consommation face aux limites, et `systemd-cgtop` la suit en direct (chapitre 12).

Le cloisonnement

Chaque directive retire au service une possibilité dont il n'a pas besoin. `systemd-analyze security s` en dresse la liste et note l'exposition de 0 à 10.

Directive	Effet
Fichiers	
`ProtectSystem=`	`yes` : `/usr` et `/boot` en lecture seule ; `full` : aussi `/etc` ; `strict` : tout, sauf `/dev`, `/proc` et `/sys`
`ReadWritePaths=`, `ReadOnlyPaths=`, `InaccessiblePaths=`	Exceptions ; `-` devant un chemin le rend facultatif
`ProtectHome=`	`yes` : `/home`, `/root` et `/run/user` invisibles ; `read-only` ; `tmpfs` : vides
`PrivateTmp=yes`	Un `/tmp` et un `/var/tmp` propres au service
`PrivateDevices=yes`	Seuls les pseudo-périphériques (`/dev/null`, `/dev/random`…) restent visibles
`DevicePolicy=closed`	Idem, par le contrôle d'accès des groupes de contrôle
Noyau et système	
`ProtectKernelTunables=yes`	`/proc/sys` et `/sys` en lecture seule
`ProtectKernelModules=yes`	Pas de chargement de module
`ProtectKernelLogs=yes`	Pas d'accès au journal du noyau
`ProtectControlGroups=yes`	Les groupes de contrôle en lecture seule
`ProtectClock=yes`	Pas de réglage de l'horloge
`ProtectHostname=yes`	Pas de changement du nom de la machine
Processus	
`ProtectProc=invisible`	Les processus des autres comptes disparaissent de `/proc`
`ProcSubset=pid`	`/proc` ne montre que des processus, pas `/proc/meminfo` ni `/proc/loadavg`
`PrivateUsers=yes`	Un espace de noms d'utilisateurs : hors root et le compte du service, tous les comptes apparaissent comme `nobody`

Directive	Effet
`RemoveIPC=yes`	Supprimer à l'arrêt les files et mémoires partagées du compte
Privilèges	
`NoNewPrivileges=yes`	Impossible d'acquérir des droits, par `sudo` ou un programme setuid
`RestrictSUIDSGID=yes`	Impossible de créer des fichiers setuid ou setgid
`CapabilityBoundingSet=`	Les capacités autorisées. Vide : aucune
`AmbientCapabilities=`	Les capacités données à un service non root, par exemple `CAP_NET_BIND_SERVICE` pour écouter sur le port 80
`LockPersonality=yes`	Impossible de changer de mode d'exécution
Réseau	
`PrivateNetwork=yes`	Aucun réseau, sauf une boucle locale à lui seul
`RestrictAddressFamilies=`	Les familles de sockets permises : `AF_UNIX AF_INET AF_INET6`
Appels système	
`SystemCallFilter=`	Les appels permis, par groupes : `@system-service` ; ~ devant interdit
`SystemCallArchitectures=native`	Refuser les appels d'une autre architecture
`SystemCallErrorNumber=EPERM`	Refuser un appel par une erreur plutôt qu'en tuant le processus
`RestrictNamespaces=yes`	Impossible de créer des espaces de noms
`RestrictRealtime=yes`	Impossible de passer en temps réel
`MemoryDenyWriteExecute=yes`	Pas de mémoire à la fois inscriptible et exécutable. Incompatible avec les programmes qui produisent du code à l'exécution, comme les moteurs JIT de Java ou de Node.js

`DynamicUser=yes` entraîne six de ces protections, qu'on ne peut plus désactiver :

- `ProtectSystem=strict` et `ProtectHome=read-only` ;
- `PrivateTmp=yes` et `RemoveIPC=yes` ;
- `NoNewPrivileges=yes` et `RestrictSUIDSGID=yes`.

`systemd-analyze syscall-filter @groupe` liste les appels d'un groupe.

La section [Install]

Elle ne sert qu'à `systemctl enable` et `disable`.

Directive	Effet
`WantedBy=`	`enable` crée un lien dans `cible.wants/` : la cible démarre l'unité. `multi-user.target` pour un service, `timers.target` pour une minuterie
`RequiredBy=`	Idem, avec un lien dans `cible.requires/`
`Alias=`	Un autre nom pour l'unité
`Also=`	Activer aussi ces unités, par exemple la minuterie avec son service

Les minuteries

Une minuterie `nom.timer` lance `nom.service` (chapitre 13).

Directive	Effet
`OnCalendar=`	À des dates précises (tableau suivant)
`OnBootSec=`	Tant de temps après le démarrage de la machine
`OnStartupSec=`	Tant de temps après le démarrage de systemd
`OnActiveSec=`	Tant de temps après l'activation de la minuterie
`OnUnitActiveSec=`	Tant de temps après le dernier lancement du service
`OnUnitInactiveSec=`	Tant de temps après la dernière fin du service
`Persistent=true`	Rattraper au démarrage un passage manqué machine éteinte (`OnCalendar=`)
`RandomizedDelaySec=`	Un retard au hasard, pour étaler la charge
`FixedRandomDelay=yes`	Le même retard à chaque fois
`AccuracySec=`	La précision tolérée (1 min par défaut)
`Unit=`	Lancer une autre unité que `nom.service`

Directive	Effet
`WakeSystem=yes`	Sortir la machine de veille

`systemd-analyze calendar` vérifie une expression et donne la prochaine échéance :

```
gary@serveur:~$ systemd-analyze calendar 'Mon..Fri 08:00'
  Original form: Mon..Fri 08:00
Normalized form: Mon..Fri *-*-* 08:00:00
    Next elapse: Tue 2026-10-06 08:00:00 UTC
       From now: 18h left
```

La forme normalisée donne, dans l'ordre, le jour de la semaine, la date et l'heure ; `*` y signifie « n'importe lequel » :

Expression	Forme normalisée	Signification
`daily`	`*-*-* 00:00:00`	Chaque jour à minuit
`weekly`	`Mon *-*-* 00:00:00`	Chaque lundi à minuit
`*-*-* 02:30:00`	`*-*-* 02:30:00`	Chaque nuit à 2 h 30
`Mon..Fri 08:00`	`Mon..Fri *-*-* 08:00:00`	Les jours ouvrés à 8 h
`Sat,Sun 10:00`	`Sat,Sun *-*-* 10:00:00`	Le week-end à 10 h
`*-*-01 03:00`	`*-*-01 03:00:00`	Le premier du mois à 3 h
`*:0/15`	`*-*-* *:00/15:00`	Toutes les quinze minutes

Les durées s'écrivent avec leurs unités, `s`, `min`, `h`, `d`, `w`, et se combinent : `systemd-analyze timespan '1h 30min'` confirme 5 400 000 000 microsecondes. Un nombre seul compte en secondes.

Modèles et spécificateurs

Une unité dont le nom finit par `@` est un **modèle** : `alerte@.service` (chapitre 16) se lance sous le nom `alerte@quelquechose.service`, et la partie après `@` est l'**instance**. Dans le fichier, des **spécificateurs** sont remplacés au chargement. Ce modèle les affiche :

```
# Un modèle qui affiche ses spécificateurs
[Unit]
Description=Essai des spécificateurs (%i)

[Service]
Type=oneshot
ExecStart=/bin/echo "n=%n p=%p i=%i I=%I"
ExecStart=/bin/echo "H=%H u=%u h=%h S=%S C=%C L=%L t=%t"
```

```
gary@serveur:~$ sudo systemctl start essai@sauvegarde-disque.service
gary@serveur:~$ journalctl -u essai@sauvegarde-disque.service -o cat --no-pager | grep '='
n=essai@sauvegarde-disque.service p=essai i=sauvegarde-disque I=sauvegarde/disque
H=serveur u=root h=/root S=/var/lib C=/var/cache L=/var/log t=/run
```

Spécificateur	Valeur
`%n`	Le nom complet de l'unité
`%p`	Le préfixe, avant `@`
`%i`	L'instance, telle quelle
`%I`	L'instance déchiffrée : `-` redevient `/`
`%H`	Le nom de la machine
`%u`, `%h`	Le compte du service et son dossier personnel (`root` sans `User=`)
`%S`, `%C`, `%L`, `%t`	Les dossiers de données, de cache, de journaux et d'exécution

`%I` sert aux instances qui portent un chemin : `systemd-escape -p /srv/donnees` donne `srv-donnees`, que `%I` retransforme en `srv/donnees`. Pour écrire le caractère `%` lui-même, on le double.

Exemple commenté : un service durci

L'application web du chapitre 23 tournait sous un compte dynamique, avec deux protections. Voici la même, cloisonnée autant qu'elle le supporte :

```
# /etc/systemd/system/tableau.service — l'application web du chapitre 23, durcie

[Unit]
Description=Tableau de la charge (application web)
Documentation=man:gunicorn(1)
# Démarrer après le réseau ; ne crée pas de dépendance
After=network.target
```

```
[Service]
# exec : l'échec du lancement de gunicorn est signalé tout de suite
Type=exec
WorkingDirectory=/opt/tableau
ExecStart=/usr/bin/gunicorn --bind 127.0.0.1:8000 --workers 2 tableau:application
# gunicorn relit son code et relance ses ouvriers sur SIGHUP
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure
RestartSec=2

# Un compte créé au démarrage, détruit à l'arrêt.
# Implique ProtectSystem=strict, ProtectHome=read-only, PrivateTmp,
# NoNewPrivileges, RestrictSUIDSGID et RemoveIPC.
DynamicUser=yes
UMask=0077

# Ce que le service voit
ProtectHome=yes
PrivateDevices=yes
PrivateUsers=yes
ProtectProc=invisible
ProcSubset=pid
ProtectKernelTunables=yes
ProtectKernelModules=yes
ProtectKernelLogs=yes
ProtectControlGroups=yes
ProtectClock=yes
ProtectHostname=yes

# Ce que le service peut faire
CapabilityBoundingSet=
RestrictAddressFamilies=AF_UNIX AF_INET AF_INET6
RestrictNamespaces=yes
RestrictRealtime=yes
LockPersonality=yes
MemoryDenyWriteExecute=yes
SystemCallArchitectures=native
SystemCallFilter=@system-service
# Pas ~@privileged : gunicorn appelle setgroups() pour chaque ouvrier
SystemCallFilter=~@resources

# Le réseau : la boucle locale seulement (Nginx est sur la même machine)
IPAddressAllow=localhost
IPAddressDeny=any

# Les ressources
MemoryMax=200M
TasksMax=20

[Install]
WantedBy=multi-user.target
```

Avant, puis après :

```
gary@serveur:~$ systemd-analyze security tableau.service --no-pager | tail -n 1
→ Overall exposure level for tableau.service: 7.5 EXPOSED
gary@serveur:~$ sudo cp ~/tableau.service /etc/systemd/system/tableau.service
gary@serveur:~$ systemd-analyze verify /etc/systemd/system/tableau.service
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl restart tableau.service
gary@serveur:~$ systemctl status tableau.service --no-pager -n 0
● tableau.service - Tableau de la charge (application web)
     Loaded: loaded (/etc/systemd/system/tableau.service; enabled; preset: enabled)
     Active: active (running) since Mon 2026-10-05 13:51:07 UTC; 362ms ago
       Docs: man:gunicorn(1)
   Main PID: 10444 (gunicorn)
      Tasks: 3 (limit: 20)
     Memory: 22.2M (max: 200.0M available: 177.7M peak: 22.3M)
[…]
gary@serveur:~$ curl -s http://127.0.0.1:8000/ | tail -n 1
Demandé par 127.0.0.1
gary@serveur:~$ systemd-analyze security tableau.service --no-pager | tail -n 1
→ Overall exposure level for tableau.service: 1.0 OK
```

`systemd-analyze verify` ne dit rien quand tout va bien. Les limites apparaissent dans `status`. Ce qui reste ouvert, `systemd-analyze security tableau.service | grep '×'` le liste avec une croix :

Point signalé	Pourquoi il reste ouvert
`SystemCallFilter=~@privileged`	Les appels du groupe `@privileged` sont permis (voir plus bas)
`RootDirectory=`, `RootImage=`	Le service voit l'arborescence de la machine, en lecture seule
`RestrictAddressFamilies=~AF_UNIX`	Il ouvre des sockets locales, ne serait-ce que pour écrire au journal
`RestrictAddressFamilies=~AF_(INET|INET6)`	C'est un serveur web : il ouvre des sockets réseau
`PrivateNetwork=`	Il a accès au réseau de la machine
`DeviceAllow=`	Il peut lire l'horloge matérielle (`char-rtc`), ce qu'ajoute `ProtectClock=yes`
`IPAddressDeny=`	La liste d'adresses permises ne contient que la boucle locale : c'est le but

Recharger le service envoie SIGHUP à gunicorn, qui remplace ses ouvriers sans couper le service :

```
gary@serveur:~$ sudo systemctl reload tableau.service
gary@serveur:~$ journalctl -u tableau.service -n 4 -o cat --no-pager
[2026-10-05 13:51:09 +0000] [10483] [INFO] Booting worker with pid: 10483
[2026-10-05 13:51:09 +0000] [10484] [INFO] Booting worker with pid: 10484
[2026-10-05 13:51:09 +0000] [10448] [INFO] Worker exiting (pid: 10448)
[2026-10-05 13:51:09 +0000] [10447] [INFO] Worker exiting (pid: 10447)
```

Le piège du filtre d'appels système

`systemd-analyze security` compte encore `@privileged` parmi les appels permis. Essayons de l'interdire, par une surcharge :

```
gary@serveur:~$ sudo mkdir -p /etc/systemd/system/tableau.service.d
gary@serveur:~$ printf '[Service]\nSystemCallFilter=~@privileged\n' | \
    sudo tee /etc/systemd/system/tableau.service.d/essai.conf
[Service]
SystemCallFilter=~@privileged
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ systemctl show -p FragmentPath -p DropInPaths tableau.service
FragmentPath=/etc/systemd/system/tableau.service
DropInPaths=/etc/systemd/system/tableau.service.d/essai.conf
gary@serveur:~$ sudo systemctl restart tableau.service
gary@serveur:~$ journalctl -u tableau.service -n 4 -o cat --no-pager
[2026-10-05 13:51:15 +0000] [10677] [INFO] Booting worker with pid: 10677
[2026-10-05 13:51:15 +0000] [10561] [WARNING] Worker with pid 10675 was terminated
  due to signal 31
[2026-10-05 13:51:15 +0000] [10561] [WARNING] Worker with pid 10677 was terminated
  due to signal 31
[2026-10-05 13:51:15 +0000] [10679] [INFO] Booting worker with pid: 10679
gary@serveur:~$ curl -s --max-time 5 http://127.0.0.1:8000/ ; echo "curl : code $?"
curl : code 28
```

Le service est « actif », mais ses ouvriers meurent dès leur naissance et la page ne répond plus : le code 28 de `curl` est un délai dépassé. Le signal 31 est SIGSYS, celui par lequel le filtre tue un processus qui fait un appel interdit. Lequel ? Le filtre ne laisse aucune trace au journal. `strace -f -p` sur le processus principal montre le dernier appel d'un ouvrier avant sa mort :

```
8996  getgid()                          = 63802
8996  setgroups(0, [])                  = 116
8996  +++ killed by SIGSYS (core dumped) +++
```

Chaque ouvrier de gunicorn abandonne ses groupes par `setgroups`, un appel du groupe `@privileged` :

```
gary@serveur:~$ systemd-analyze syscall-filter @privileged | grep -w setgroups
    setgroups
gary@serveur:~$ sudo rm -r /etc/systemd/system/tableau.service.d
```

```
gary@serveur:~$ sudo systemctl daemon-reload
gary@serveur:~$ sudo systemctl restart tableau.service
gary@serveur:~$ curl -s http://127.0.0.1:8000/ | tail -n 1
Demandé par 127.0.0.1
```

La leçon vaut pour toutes les directives de ce chapitre : un service peut rester « actif » et ne plus rien servir. Après chaque ajout, on vérifie qu'il *rend son service*, pas seulement qu'il tourne. `SystemCallErrorNumber=EPERM` remplace la mise à mort par une erreur, souvent plus facile à diagnostiquer dans le journal du programme.

Vérifier une unité

Commande	Effet
`systemd-analyze verify fichier`	Erreurs de syntaxe, directives inconnues, programmes introuvables
`systemd-analyze security s`	L'exposition, directive par directive
`systemd-analyze calendar 'expr'`	Une expression de date, sa prochaine échéance
`systemd-analyze timespan 'durée'`	Une durée
`systemd-analyze syscall-filter @groupe`	Les appels système d'un groupe
`systemctl list-dependencies s`	L'arbre des dépendances
`systemctl show s -p Directive`	La valeur effective d'une directive
`journalctl -u s -b`	Le journal du service depuis le démarrage

Annexe C — Glossaire

Les termes techniques du livre, avec leur équivalent anglais quand on le rencontre dans la documentation, et le chapitre qui les explique. Les définitions sont volontairement courtes : le chapitre cité en donne le détail.

A

ACL (*access control list*)
Droits ajoutés pour un utilisateur ou un groupe précis, en plus du propriétaire, du groupe et des autres. `getfacl` les lit, `setfacl` les pose. *Chapitre 6.*

Agent SSH (*ssh-agent*)
Programme qui garde une clé privée déverrouillée en mémoire pendant la session, pour ne pas retaper sa phrase secrète à chaque connexion. *Chapitre 19.*

Agrégat (*bond*)
Réunion de deux cartes réseau en une seule interface. *Chapitre 18.*

Alertmanager
Programme qui reçoit les alertes de Prometheus, les trie et les envoie. *Chapitre 26.*

Alias
Nom court que le shell remplace par une commande plus longue : `alias ll='ls -l'`. *Chapitres 5 et 8.*

Ansible
Outil d'automatisation qui applique par SSH un état décrit dans des fichiers YAML, sans rien installer sur les machines gérées. *Chapitre 29.*

Ansible Vault
Mécanisme d'Ansible qui chiffre les fichiers contenant des secrets. *Chapitre 29.*

AppArmor
Le contrôle d'accès obligatoire d'Ubuntu : chaque programme confiné a un profil qui dit ce qu'il peut toucher. *Chapitre 22.*

Appel système (*system call*)
Demande d'un programme au noyau : ouvrir un fichier, lire, écrire, se connecter. `strace` les affiche. *Chapitre 30.*

AppImage
Application livrée en un seul fichier exécutable qui contient ses bibliothèques. Ni installation ni mise à jour automatique. *Chapitre 7.*

APT (*Advanced Package Tool*)
Le gestionnaire de paquets d'Ubuntu : il cherche les paquets dans les dépôts, résout leurs dépendances et les fait installer par `dpkg`. *Chapitre 7.*

auditd
Service qui enregistre, au niveau du noyau, les accès à ce qu'on lui demande de surveiller. *Chapitre 20.*

Autorité de certification (*certificate authority*)
Organisme dont les navigateurs connaissent la clé, et qui signe les certificats. Let's Encrypt en est une. *Chapitre 23.*

B

Base de données (*database*)
Logiciel qui range des informations pour les retrouver vite, à plusieurs en même temps, sans les abîmer. *Chapitre 24.*

become
Dans Ansible, le passage aux droits d'administrateur, par `sudo` le plus souvent. *Chapitre 29.*

Bibliothèque système (*system library*)
Ensemble de fonctions que tous les programmes réutilisent. La principale est la `glibc`. *Chapitre 1.*

BIOS (*Basic Input/Output System*)
Micrologiciel des anciens PC : il exécute le programme écrit dans le premier secteur du disque. Remplacé par l'UEFI. *Chapitre 11.*

Branche (*branch*)
Dans Git, ligne de travail parallèle, que l'on fusionne ensuite. *Chapitre 27.*

C

Capacité (*capability*)
Morceau des pouvoirs de root que le noyau peut accorder séparément, par exemple `CAP_NET_BIND_SERVICE` pour écouter sur un port inférieur à 1024. *Annexe B.*

Certbot
Le client de Let's Encrypt : il obtient les certificats, adapte la configuration de Nginx et les renouvelle. *Chapitre 23.*

Certificat (*certificate*)
Fichier qui associe un nom de domaine à une clé publique, signé par une autorité de certification. *Chapitre 23.*

Chaîne (*chain*)
Dans nftables, liste de règles. Une table contient des chaînes, une chaîne des règles, chacune terminée par un verdict. *Chapitre 21.*

Charge moyenne (*load average*)
Nombre moyen de processus qui utilisent un processeur ou en attendent un, disque compris, sur une, cinq et quinze minutes. *Chapitre 12.*

Chargeur d'amorçage (*bootloader*)
Programme lancé par le micrologiciel, qui charge le noyau et l'initramfs. Sur Ubuntu, GRUB. *Chapitre 11.*

Chemin absolu (*absolute path*)
Chemin qui part de la racine, et commence donc par `/`. *Chapitre 4.*

Chemin relatif (*relative path*)
Chemin qui part du dossier courant. *Chapitre 4.*

chroot
Commande qui fait d'un dossier la racine d'un shell. Depuis un système de secours, on « entre » ainsi dans le système en panne. *Chapitre 11.*

Cible (*target*)
Unité systemd qui représente un état du système : `graphical.target`, `multi-user.target`, `rescue.target`. *Chapitre 11.*

Clé d'hôte (*host key*)
Clé par laquelle un serveur SSH prouve son identité. *Chapitre 19.*

Clé privée (*private key*)
La moitié d'une paire de clés qui reste sur le poste et n'en sort jamais. *Chapitre 19.*

Clé publique (*public key*)
La moitié d'une paire de clés que l'on dépose sur les serveurs. *Chapitre 19.*

cloud-init
Outil qui configure une machine Ubuntu à son premier démarrage : comptes, clés, réseau, paquets. *Chapitres 1 et 2.*

Code de retour (*exit status*)
Nombre renvoyé par une commande qui se termine : 0 pour un succès, autre chose sinon. `$?` le contient. *Chapitre 8.*

Collection (*collection*)
Paquet de modules prêts à l'emploi pour Ansible. *Chapitre 29.*

Commit
Dans Git, enregistrement d'un état complet du projet, avec son message. *Chapitre 27.*

complain
Mode d'AppArmor où tout est permis, mais où chaque écart au profil est noté. *Chapitre 22.*

Conteneur (*container*)
Groupe de processus de l'hôte, isolés par les espaces de noms et limités par les groupes de contrôle, qui exécutent une image. *Chapitre 28.*

Contrôle d'accès obligatoire (*mandatory access control*)
Seconde vérification, qui porte sur le programme et non sur l'utilisateur, root compris. *Chapitre 22.*

Copyleft
Clause des licences libres qui impose qu'une version modifiée et distribuée garde la même licence. *Chapitre 1.*

Couche (*layer*)
Dans une image Docker, ce que produit une instruction du Dockerfile. Docker garde les couches en cache. *Chapitre 28.*

Crochet (*hook*)
Script que Git exécute à un moment précis, par exemple avant d'enregistrer un commit. *Chapitre 27.*

cron
Le planificateur historique d'Unix, qui lance des commandes à heures fixes. *Chapitre 13.*

CrowdSec
Outil qui repère les comportements hostiles dans les journaux et met en commun les adresses fautives entre ses utilisateurs. *Chapitre 21.*

D

deb822
Format des fichiers de dépôts `.sources` d'APT, utilisé par Ubuntu depuis la 24.04. *Chapitres 1 et 7.*

Défense en profondeur (*defense in depth*)
Principe qui superpose plusieurs protections, aucune n'étant parfaite. *Chapitre 20.*

Demande de fusion (*pull request, merge request*)
Proposition d'intégrer une branche, discutée et relue sur une plateforme. *Chapitre 27.*

Démon (*daemon*)
Programme qui tourne en arrière-plan pour rendre un service. *Chapitre 12.*

Dépôt (APT) (*repository*)
Serveur qui publie des paquets signés et leur liste. *Chapitre 7.*

Dépôt (Git) (*repository*)
Le dossier `.git`, qui garde l'historique d'un projet. *Chapitre 27.*

Dépôt nu (*bare repository*)
Dépôt Git sans dossier de travail, qui sert seulement à recevoir et à distribuer les commits. *Chapitre 27.*

DHCP (*Dynamic Host Configuration Protocol*)
Protocole par lequel une machine reçoit son adresse IP du réseau, pour une durée donnée. *Chapitres 17 et 18.*

Distant (*remote*)
Dans Git, un autre dépôt avec lequel on échange des commits, en général nommé `origin`. *Chapitre 27.*

Distribution
Système complet bâti autour du noyau Linux : outils GNU, système d'init, gestionnaire de paquets, applications, réglages et mises à jour. *Chapitre 1.*

DNS (*Domain Name System*)
L'annuaire qui traduit les noms de domaine en adresses IP. *Chapitre 17.*

Docker
Outil qui construit des images et fait tourner des conteneurs. *Chapitre 28.*

Docker Compose
Outil qui décrit une application de plusieurs conteneurs dans un fichier `compose.yaml` et la lance d'une commande. *Chapitre 28.*

Docker Hub
Le registre d'images utilisé par défaut par Docker. *Chapitre 28.*

Dockerfile
La recette qui fabrique une image. *Chapitre 28.*

Dossier de travail (*working tree*)
Dans Git, les fichiers tels qu'on les modifie. *Chapitre 27.*

Dotfiles
Fichiers de configuration du dossier personnel, dont le nom commence par un point. *Chapitre 33.*

dpkg
L'outil qui installe un fichier `.deb` et tient le registre des paquets installés. *Chapitre 7.*

E

Empreinte (*fingerprint*)
Résumé court d'une clé, qui permet de la vérifier. *Chapitre 19.*

enforce
Mode d'AppArmor où ce que le profil ne prévoit pas est refusé. *Chapitre 22.*

Ensemble (*set*)
Dans nftables, liste d'adresses que le noyau consulte très vite, même longue. *Chapitre 21.*

Entrée standard (*stdin*)
Le canal 0 d'un programme : ce qu'il lit, le clavier par défaut. *Chapitre 5.*

Environnement de bureau (*desktop environment*)
La partie graphique du système. Ubuntu utilise GNOME. *Chapitres 1 et 33.*

Environnement virtuel (*virtual environment*)
Dossier qui contient un Python et des modules propres à un projet : `python -m venv .venv`. *Chapitre 33.*

Espace d'échange (*swap*)
Zone du disque où le noyau range des pages de mémoire peu utilisées quand la mémoire vive manque. *Chapitre 14.*

Espaces de noms (*namespaces*)
Mécanisme du noyau qui cache à des processus ce qui ne les concerne pas : autres processus, réseau, montages. La base des conteneurs. *Chapitre 28.*

etckeeper
Outil qui fait de `/etc` un dépôt Git et enregistre seul les changements apportés par `apt`. *Chapitre 27.*

Étiquette (Git) (*tag*)
Nom durable donné à un commit, en général un numéro de version. *Chapitre 27.*

Étiquette (Prometheus) (*label*)
Couple nom-valeur qui distingue les séries d'une même mesure. *Chapitre 26.*

Exportateur (*exporter*)
Programme qui expose des mesures que Prometheus vient relever. *Chapitre 26.*

F

Fail2ban
Service qui surveille les journaux et bannit par le pare-feu les adresses qui échouent trop souvent. *Chapitre 21.*

Faits (*facts*)
Informations qu'Ansible relève sur chaque machine avant d'agir : système, mémoire, interfaces, disques. *Chapitre 29.*

FHS (*Filesystem Hierarchy Standard*)
La norme qui fixe l'organisation des dossiers d'un système Linux. *Chapitre 4.*

Flatpak
Format d'applications de bureau confinées, distribuées surtout par Flathub. *Chapitres 7 et 33.*

fstab
Le fichier `/etc/fstab`, qui liste les systèmes de fichiers à monter au démarrage. *Chapitre 14.*

G

GID (*group identifier*)
Le numéro d'un groupe. *Chapitre 6.*

Git
Système de suivi de versions, qui garde l'historique complet dans chaque copie du projet. *Chapitre 27.*

GNOME
L'environnement de bureau d'Ubuntu, en version 46 dans la 24.04. *Chapitres 1 et 33.*

GNU
Projet lancé en 1983 pour écrire un système d'exploitation libre. Ses outils, avec le noyau Linux, forment le cœur des distributions. *Chapitre 1.*

GPT (*GUID Partition Table*)
La table de partitions actuelle, à préférer au MBR. *Chapitre 14.*

Grafana
Outil qui dessine des tableaux de bord à partir de mesures. *Chapitre 26.*

Grappe (*cluster*)
Pour PostgreSQL, un dossier de données servi par une instance du serveur, qui contient plusieurs bases. Ubuntu crée `16/main`. *Chapitre 24.*

Groupe de contrôle (*cgroup*)
Mécanisme du noyau qui regroupe des processus pour mesurer et limiter leur consommation. systemd en crée un par service. *Chapitre 12.*

Groupe de volumes (*volume group*)
Dans LVM, la réserve d'espace formée d'un ou plusieurs volumes physiques. *Chapitre 15.*

Groupe principal (*primary group*)
Le groupe donné par défaut aux fichiers que crée un utilisateur. Les autres groupes dont il est membre sont ses groupes secondaires. *Chapitre 6.*

GRUB (*GRand Unified Bootloader*)
Le chargeur d'amorçage d'Ubuntu. *Chapitre 11.*

H

Handler
Dans Ansible, tâche lancée une seule fois, à la fin, si une autre l'a prévenue par `notify`. Typiquement, redémarrer un service. *Chapitre 29.*

HTTP, HTTPS (*Hypertext Transfer Protocol*)
Le protocole du Web. HTTPS est HTTP chiffré par TLS. *Chapitre 23.*

HWE (*Hardware Enablement*)
Mécanisme qui apporte aux versions LTS des noyaux plus récents. *Chapitre 1.*

Hyperviseur (*hypervisor*)
Logiciel qui simule des ordinateurs pour faire tourner des machines virtuelles. *Chapitre 2.*

I

Idempotence
Propriété d'une opération qui, appliquée deux fois, donne le même résultat qu'une seule. *Chapitre 29.*

Image (*image*)
Modèle en lecture seule d'un conteneur : un système de fichiers minimal et la commande à lancer. *Chapitre 28.*

Index (*staging area*)
Dans Git, la zone où `git add` prépare le prochain commit. *Chapitre 27.*

initramfs
Petit système de fichiers chargé en mémoire avec le noyau, avec de quoi trouver et monter la vraie racine. *Chapitre 11.*

Inode
Fiche qui décrit un fichier : type, taille, propriétaire, droits, dates, emplacement des données. Le nom n'y figure pas. *Chapitre 4.*

Instantané (*snapshot*)
État figé d'un volume, ou d'une sauvegarde, à un moment donné. *Chapitres 15 et 16.*

Intégration continue (*continuous integration*)
Vérifications lancées automatiquement à chaque envoi de commits. *Chapitre 27.*

Interface réseau (*network interface*)
Point d'attache d'une machine à un réseau, physique comme `enp0s3` ou virtuel comme `lo`. *Chapitre 17.*

Inventaire (*inventory*)
La liste des machines gérées par Ansible, rangées en groupes. *Chapitre 29.*

IPv4, IPv6
Les deux versions du protocole IP, aux adresses de 32 bits (`192.168.1.50`) et de 128 bits. *Chapitre 17.*

J

Jinja2
Le langage de modèles d'Ansible : des variables entre doubles accolades dans un texte. *Chapitre 29.*

journald
Le service de systemd qui recueille les messages du noyau et des services. `journalctl` le lit. *Chapitre 13.*

L

Lien physique (*hard link*)
Second nom pour les mêmes données, c'est-à-dire le même inode. *Chapitre 4.*

Lien symbolique (*symbolic link*)
Petit fichier qui contient le chemin d'un autre. *Chapitre 4.*

Livepatch
Service d'Ubuntu Pro qui applique les correctifs les plus graves au noyau en marche, sans redémarrage. *Chapitre 7.*

logrotate
Outil qui renouvelle, compresse et supprime les vieux fichiers journaux. *Chapitre 13.*

LTS (*Long Term Support*)
Version d'Ubuntu publiée tous les deux ans et maintenue cinq ans, dix avec Ubuntu Pro. *Chapitre 1.*

LUKS (*Linux Unified Key Setup*)
Le format de chiffrement d'un périphérique entier. *Chapitre 15.*

LVM (*Logical Volume Manager*)
Couche entre les disques et les systèmes de fichiers, qui permet d'agrandir, de déplacer et de figer les volumes. *Chapitre 15.*

Lynis
Outil d'audit qui passe la machine en revue, plus de deux cent cinquante contrôles, et liste avertissements et suggestions. *Chapitre 20.*

M

Machine virtuelle (*virtual machine*)
Ordinateur complet simulé par un hyperviseur, avec son propre noyau. *Chapitres 2 et 28.*

main, universe, restricted, multiverse
Les quatre dépôts d'Ubuntu : `main` (maintenu par Canonical), `universe` (par la communauté), `restricted` (pilotes propriétaires), `multiverse` (logiciels soumis à des restrictions de licence). *Chapitres 1 et 7.*

Masque (*netmask*)
Le nombre de bits de la partie réseau d'une adresse, noté après une barre oblique : `/24`. *Chapitre 17.*

MBR (*Master Boot Record*)
L'ancienne table de partitions, limitée aux disques de 2 Tio. *Chapitre 14.*

Minuterie (*timer*)
Unité systemd `.timer` qui lance un service à des dates ou à intervalles réguliers. *Chapitre 13.*

mise
Outil qui installe les versions de langages demandées par chaque projet, décrites dans `mise.toml`. *Chapitre 33.*

Mode de secours (*rescue mode*)
La cible `rescue.target` : systèmes de fichiers montés, presque aucun service, un shell root. *Chapitre 11.*

Modèle (Ansible) (*template*)
Fichier Jinja2 dont Ansible remplit les variables pour produire un fichier de configuration. *Chapitre 29.*

Modèle (systemd) (*template unit*)
Unité dont le nom finit par `@`, lancée sous plusieurs instances : `alerte@nginx.service`. *Chapitre 16 et annexe B.*

Module (Ansible) (*module*)
Programme qui réalise une action précise. Chaque tâche en appelle un. *Chapitre 29.*

Moindre privilège (*least privilege*)
Principe : chaque compte et chaque programme ne reçoit que les droits nécessaires à sa tâche. *Chapitre 20.*

Montage lié (*bind mount*)
Dossier de l'hôte monté dans un conteneur. *Chapitre 28.*

Mot de passe d'application (*app password*)
Mot de passe propre à une application, qu'on révoque sans toucher à celui du compte. *Chapitre 32.*

N

NAT (*Network Address Translation*)
Traduction d'adresses : la box fait sortir les machines du réseau privé sous son adresse publique. *Chapitre 17.*

Netfilter
Le mécanisme de filtrage des paquets du noyau Linux. *Chapitre 21.*

Netplan
Le point d'entrée de la configuration réseau d'Ubuntu : un fichier YAML, traduit pour systemd-networkd ou NetworkManager. *Chapitre 18.*

NetworkManager
Le gestionnaire réseau d'Ubuntu Desktop : Wi-Fi, VPN, réseaux qui changent. *Chapitre 18.*

NFS (*Network File System*)
Protocole de partage de fichiers entre machines Linux. *Chapitre 25.*

nftables
Le langage de règles actuel de Netfilter, successeur d'iptables. UFW écrit ses règles en nftables. *Chapitre 21.*

Noyau (*kernel*)
Le cœur du système : il gère le matériel, la mémoire et les processus, et reste en mémoire jusqu'à l'extinction. *Chapitre 1.*

O

OOM killer (*out of memory*)
Mécanisme du noyau qui arrête un processus quand la mémoire est épuisée. Il laisse une trace dans `journalctl -k`. *Chapitres 26 et 30.*

P

Paire de clés (*key pair*)
Une clé privée et une clé publique liées par les mathématiques. *Chapitre 19.*

Paquet (*package*)
Un fichier `.deb` : un logiciel, sa description et la liste de ses dépendances. *Chapitre 7.*

Pare-feu (*firewall*)
Filtre qui décide quels paquets réseau peuvent entrer, sortir ou traverser la machine. *Chapitre 21.*

Partition
Une zone d'un disque, décrite par la table de partitions. *Chapitre 14.*

Passerelle (*gateway*)
Le routeur auquel une machine confie tout ce qui sort de son réseau local. *Chapitre 17.*

peer
Méthode d'accès locale de PostgreSQL : le rôle doit porter le nom du compte Unix qui se connecte. *Chapitre 24.*

Phrase secrète (*passphrase*)
Le mot de passe qui protège une clé privée. *Chapitre 19.*

PID (*process identifier*)
Le numéro d'un processus. Le PID 1 est systemd. *Chapitre 12.*

Playbook
Fichier YAML d'Ansible qui énumère des tâches à appliquer à des groupes de machines. *Chapitre 29.*

Point de montage (*mount point*)
Le dossier par lequel on accède à un système de fichiers. *Chapitre 14.*

Port
Nombre de 1 à 65 535 qui désigne, sur une machine, le service auquel on s'adresse. *Chapitre 17.*

Poste de contrôle (*control node*)
La machine d'où l'on commande Ansible, la seule où il est installé. *Chapitre 29.*

PPA (*Personal Package Archive*)
Dépôt hébergé sur Launchpad, la plateforme de développement d'Ubuntu. *Chapitre 7.*

Processus (*process*)
Un programme en train de s'exécuter : du code en mémoire, des données, des fichiers ouverts, une identité. *Chapitre 12.*

Profil (*profile*)
Pour AppArmor, la liste de ce qu'un programme confiné a le droit de toucher. *Chapitre 22.*

Prometheus
Outil de supervision qui relève des mesures à intervalles réguliers, les range avec leur heure et évalue des règles d'alerte. *Chapitre 26.*

R

RAID (*Redundant Array of Independent Disks*)
Assemblage de disques en une grappe vue comme un seul. En miroir (RAID 1), il survit à la perte d'un disque. *Chapitre 15.*

Rebond (*jump host, bastion*)
Machine exposée par laquelle on passe pour atteindre celles d'un réseau privé. *Chapitre 19.*

Registre (*registry*)
Un dépôt d'images de conteneurs. *Chapitre 28.*

Relais (*reverse proxy*)
Serveur placé devant une application, qui reçoit les requêtes et les lui transmet. Nginx dans ce livre. *Chapitre 23.*

Résolveur (*resolver*)
Serveur qui interroge les serveurs DNS successifs pour les machines et garde les réponses quelque temps. Sur Ubuntu, `systemd-resolved` en tient un, sur `127.0.0.53`. *Chapitre 17.*

restic
Outil de sauvegarde chiffrée, qui garde des instantanés successifs dans un dépôt. *Chapitre 16.*

Rôle (Ansible) (*role*)
Ensemble rangé de tâches, de modèles, de variables et de handlers, réutilisable d'un projet à l'autre. *Chapitre 29.*

Rôle (PostgreSQL) (*role*)
Un compte de PostgreSQL. *Chapitre 24.*

root
Le superutilisateur, UID 0, qui a tous les droits. Sur Ubuntu, son mot de passe est verrouillé : on passe par `sudo`. *Chapitre 6.*

rsyslog
Service qui reçoit une copie des messages de journald et les trie dans des fichiers texte de `/var/log`. *Chapitre 13.*

S

Samba
Logiciel qui fait d'un serveur Linux un serveur de fichiers pour Windows, dans le protocole SMB. *Chapitre 25.*

Secure Boot
Mécanisme de l'UEFI qui ne démarre que des programmes signés. Ubuntu le prend en charge. *Chapitre 2.*

SGID (*set group ID*)
Droit spécial. Sur un dossier, les fichiers créés dedans prennent le groupe du dossier. *Chapitre 6.*

Shebang
La première ligne d'un script, `#!/bin/bash`, qui désigne son interpréteur. *Chapitre 10.*

Shell
Le programme qui lit les commandes tapées dans un terminal et les exécute. Bash sur Ubuntu. *Chapitre 1.*

Shell de connexion (*login shell*)
Le premier shell d'une session, par SSH ou sur une console. Il lit `~/.profile`. *Chapitre 8.*

Signal
Message très court envoyé à un processus, un simple numéro. `SIGTERM` demande l'arrêt, `SIGKILL` l'impose. *Chapitre 12.*

SMART
Le journal d'état qu'un disque tient sur lui-même. `smartctl` le lit. *Chapitres 14 et 31.*

Snap
Le format de paquets de Canonical, isolés du système et mis à jour automatiquement. *Chapitres 1 et 7.*

Socket
Point de communication entre programmes, local ou réseau. Une unité `.socket` écoute à la place d'un service, et ne le démarre qu'à la première connexion. *Chapitre 12.*

Sortie d'erreur (*stderr*)
Le canal 2 d'un programme, où il écrit ses messages d'erreur. *Chapitre 5.*

Sortie standard (*stdout*)
Le canal 1 d'un programme, où il écrit ses résultats. *Chapitre 5.*

SQL (*Structured Query Language*)
Le langage commun des bases de données relationnelles. *Chapitre 24.*

SSH (*Secure Shell*)
Le protocole de connexion à distance chiffrée. *Chapitre 19.*

Sticky bit
Droit spécial qui, sur un dossier où tout le monde écrit, interdit de supprimer les fichiers des autres. *Chapitre 6.*

sudo
Commande qui en exécute une autre avec les droits de root, après avoir demandé le mot de passe de l'utilisateur. *Chapitres 3 et 6.*

SUID (*set user ID*)
Droit spécial : un programme SUID s'exécute avec les droits de son propriétaire. *Chapitre 6.*

Superbloc (*superblock*)
Le bloc qui décrit tout un système de fichiers. ext4 en garde des copies. *Chapitre 31.*

Surcharge (*drop-in*)
Fichier `.conf` qui modifie une unité systemd sans la recopier. *Chapitre 12 et annexe B.*

Surface d'attaque (*attack surface*)
Tout ce qu'un attaquant peut viser. On la réduit en n'installant et n'exposant que le nécessaire. *Chapitre 20.*

systemd
Le système d'init et le gestionnaire de services d'Ubuntu. *Chapitres 11 et 12.*

systemd-networkd
Le moteur réseau d'Ubuntu Server, qui applique des réglages fixes. *Chapitre 18.*

Système d'exploitation (*operating system*)
Le logiciel qui fait l'intermédiaire entre le matériel et les applications. *Chapitre 1.*

Système d'init (*init system*)
Le premier programme lancé par le noyau, qui démarre tout le reste. systemd sur Ubuntu. *Chapitres 1 et 11.*

Système de fichiers (*file system*)
L'organisation de l'intérieur d'une partition : où est chaque fichier, à qui il appartient, quels blocs sont libres. ext4 par défaut. *Chapitre 14.*

Système de suivi de versions (*version control system*)
Logiciel qui enregistre l'état d'un ensemble de fichiers à chaque étape. *Chapitre 27.*

T

Table de partitions (*partition table*)
Description du découpage d'un disque, écrite à son début : GPT ou MBR. *Chapitre 14.*

Tâche (*task*)
Dans Ansible, une étape d'un playbook, qui appelle un module. *Chapitre 29.*

TCP (*Transmission Control Protocol*)
Protocole de transport avec connexion : il numérote, fait accuser réception et renvoie ce qui s'est perdu. *Chapitre 17.*

TestDisk, PhotoRec
Outils de récupération : le premier retrouve des partitions perdues, le second des fichiers d'après leur signature. *Chapitre 31.*

TLS (*Transport Layer Security*)
Le protocole qui chiffre les connexions, sous HTTPS notamment. *Chapitre 23.*

TTL (*time to live*)
La durée de validité d'une réponse DNS, en secondes. *Chapitre 17.*

Tube (*pipe*)
Liaison, notée `|`, entre la sortie d'une commande et l'entrée de la suivante. *Chapitre 5.*

Tunnel
Connexion transportée dans une autre, chiffrée. `ssh -L` en ouvre un vers un service interne. *Chapitre 19.*

U

UDP (*User Datagram Protocol*)
Protocole de transport sans connexion ni garantie, plus léger que TCP. Le DNS l'utilise. *Chapitre 17.*

UEFI (*Unified Extensible Firmware Interface*)
Le micrologiciel des PC depuis 2012 environ. Il sait lire un système de fichiers et lance le chargeur d'amorçage. *Chapitre 11.*

UFW (*Uncomplicated Firewall*)
L'outil de pare-feu d'Ubuntu. *Chapitre 21.*

UID (*user identifier*)
Le numéro d'un utilisateur. 0 pour root. *Chapitre 6.*

umask
Masque des droits retirés aux fichiers et dossiers que l'on crée. *Chapitre 6.*

unattended-upgrades
Le paquet qui installe seul les mises à jour de sécurité. *Chapitre 7.*

Unité (*unit*)
Tout ce que gère systemd : service, socket, minuterie, montage, cible. Chacune est décrite par un fichier. *Chapitre 12 et annexe B.*

UUID (*universally unique identifier*)
Identifiant tiré au hasard à la création d'un système de fichiers, qui le désigne même si le disque change de nom. *Chapitre 14.*

V

Variable d'environnement (*environment variable*)
Variable qu'un processus transmet à ceux qu'il lance : `PATH`, `HOME`, `LANG`. *Chapitre 8.*

Volume (Docker) (*volume*)
Espace de stockage qui survit au conteneur. *Chapitre 28.*

Volume logique (*logical volume*)
Dans LVM, l'espace taillé dans un groupe de volumes, que l'on formate comme une partition. *Chapitre 15.*

Volume physique (*physical volume*)
Dans LVM, un disque ou une partition confié à LVM. *Chapitre 15.*

W

WebDAV
Extension de HTTP qui permet de déposer, déplacer et supprimer des fichiers à distance. *Chapitre 25.*

WireGuard
Réseau privé virtuel qui relie des appareils à un serveur par des connexions chiffrées, d'où qu'ils se connectent. *Chapitre 32.*

WSL (*Windows Subsystem for Linux*)
Fonction de Windows 10 et 11 qui fait tourner un Ubuntu en ligne de commande, intégré au système. *Chapitre 2.*

Y

YAML
Format de fichier texte structuré par l'indentation, utilisé par Netplan, cloud-init, Docker Compose et Ansible. *Chapitres 18 et 29.*

Z

ZFS
Système de fichiers et gestionnaire de volumes, proposé par l'installateur d'Ubuntu. *Chapitre 2.*

Zombie
Processus terminé dont le parent n'a pas encore relevé le code de retour. *Chapitre 12.*

zsh
Shell interactif proche de Bash, à la complétion et à l'historique plus riches. *Chapitre 33.*

Annexe D — Ressources pour continuer

Ce livre est un point de départ. Voici où trouver la suite : la documentation de référence, les livres qui vont plus loin, les sites et les émissions pour se tenir au courant, les communautés où poser ses questions, et les certifications pour faire reconnaître ce que l'on sait. Tout a été vérifié à l'automne 2026 ; les adresses changent, les noms restent.

D'abord, la machine

La documentation la plus juste est celle qui accompagne les programmes installés, car elle correspond exactement à leur version (chapitre 3).

Où	Quoi
`man commande`, `man 5 fichier`	Le manuel, la référence
`commande --help`	Le résumé des options
`apropos mot`	Les pages de manuel qui parlent d'un sujet
`/usr/share/doc/paquet/`	Exemples, notes de version, fichiers d'exemple (ceux de Netplan, par exemple)
`man systemd.directives`	L'index de toutes les directives de systemd

La documentation en ligne

Adresse	Contenu
`documentation.ubuntu.com/server/`	La documentation officielle d'Ubuntu Server : tutoriels, guides pratiques, références, explications

Adresse	Contenu
`documentation.ubuntu.com/release-notes/`	Les notes de chaque version
`ubuntu.com/about/release-cycle`	Le calendrier des versions et la durée de leur suivi
`ubuntu.com/security/notices`	Les avis de sécurité publiés pour les paquets d'Ubuntu
`doc.ubuntu-fr.org`	La documentation de la communauté francophone
`wiki.archlinux.org`	Le wiki d'Arch Linux, l'un des plus complets ; presque tout s'y applique à Ubuntu
`man7.org/linux/man-pages/`	Les pages de manuel du noyau et de la bibliothèque C, à jour, sur le site de Michael Kerrisk
`debian-handbook.info`	*Le cahier de l'administrateur Debian*, gratuit et en français : Ubuntu dérive de Debian

Des livres pour aller plus loin

La plupart des références sont en anglais. Les deux premiers livres du tableau s'adressent à qui veut devenir administrateur ; les suivants approfondissent un sujet.

Livre	Auteurs, éditeur	Pour quoi
UNIX and Linux System Administration Handbook, 5e édition	Evi Nemeth, Garth Snyder, Trent Hein, Ben Whaley, Dan Mackin ; Addison-Wesley, 2017	La somme du métier, écrite par des administrateurs pour des administrateurs
Mastering Ubuntu Server, 5e édition	Jay LaCroix ; Packt	L'administration d'Ubuntu Server, dans une édition consacrée à la 26.04
LINUX — Maîtrisez l'administration du système, 7e édition	Philippe Banquet, Sébastien Rohaut ; ENI, 2024	Un manuel complet, en français
Linux System Administration for the 2020s	Kenneth Hitchcock ; Apress, 2022	L'administration d'un parc moderne : automatisation, gestion par le code

Livre	Auteurs, éditeur	Pour quoi
How Linux Works, 3e édition	Brian Ward ; No Starch Press, 2021	Ce qui se passe sous la surface : démarrage, noyau, réseau, stockage
The Linux Programming Interface	Michael Kerrisk ; No Starch Press, 2010	Le fonctionnement interne de Linux, vu par le programmeur
The Linux Command Line	William Shotts ; No Starch Press, gratuit sur `linuxcommand.org`	Le meilleur cours sur la ligne de commande
Pro Git	Scott Chacon, Ben Straub ; Apress, gratuit en français sur `git-scm.com/book/fr/v2`	Git, de la première commande aux rouages internes

Chaque chapitre propose en outre, dans sa rubrique « Pour aller plus loin », les lectures propres à son sujet.

Se tenir au courant

Adresse	Contenu
`lwn.net`	L'hebdomadaire de référence sur le noyau et l'écosystème, en anglais, très technique
`phoronix.com`	Les nouveautés du noyau, des pilotes et du matériel, avec des mesures de performance
`omgubuntu.co.uk`	L'actualité d'Ubuntu et de son bureau
`linuxfr.org`	L'actualité du logiciel libre en français, avec les journaux et les forums de ses lecteurs

Deux émissions francophones, encore actives à l'automne 2026 :

- ***Libre à vous !***, l'émission de l'April, l'association de promotion et de défense du logiciel libre. Diffusée chaque mardi à 15 h 30 sur radio Cause Commune, puis en podcast, sur `libreavous.org`. Les libertés informatiques et leur actualité politique et juridique.
- ***Projets Libres !***, le podcast officiel de LinuxFr.org, animé par Walid Nouh, sur `projets-libres.org`. Des entretiens avec ceux qui font les logiciels libres : leur histoire, leur modèle, leur gouvernance.

Poser ses questions

Où	Pour quoi
`forum.ubuntu-fr.org`	Le forum de la communauté francophone, très actif
`discourse.ubuntu.com`	Le forum officiel de la communauté Ubuntu, en anglais, avec une catégorie d'entraide
`askubuntu.com`	Questions et réponses, en anglais, sur le modèle de Stack Exchange
Matrix, `#community:ubuntu.com`	La messagerie officielle de la communauté. La documentation d'Ubuntu recommande le client Element
`linuxfr.org`	Les forums et les journaux de LinuxFr.org
`agendadulibre.org`	Les rencontres autour du logiciel libre, surtout en France : permanences, ateliers, conférences

Les anciens salons IRC d'Ubuntu, sur le réseau Libera Chat, existent toujours, mais Matrix est désormais le canal officiel.

Une question bien posée obtient une réponse rapide. Indiquez la version d'Ubuntu (`lsb_release -a`), ce que vous vouliez faire, la commande exacte, le message d'erreur exact, et ce que vous avez déjà essayé. Copiez du texte, pas une photo de l'écran. Relisez avant d'envoyer : une adresse IP publique, un nom de domaine ou un mot de passe glissés dans un journal n'ont rien à faire sur un forum.

Les certifications

Une certification ne remplace pas l'expérience, mais elle la rend visible, et sa préparation oblige à couvrir les sujets qu'on aurait laissés de côté.

Certification	Organisme	Forme de l'examen	À savoir
Canonical Academy SysAdmin	Canonical	Trois examens pratiques, passés à son rythme : *Using Linux Terminal*, *Using Ubuntu Desktop*, *Using Ubuntu Server* ; un quatrième, *Using DevOps Principles*, est en test	Chaque examen réussi donne un badge ; réussir tous ceux de la filière donne la qualification. Le seul titre de ce tableau centré sur Ubuntu

Certification	Organisme	Forme de l'examen	À savoir
LPIC-1	Linux Professional Institute	Deux examens, 101-500 et 102-500 (version 5.0), de 90 minutes et 60 questions à choix multiples ou à compléter	Indépendant de la distribution. Valable cinq ans. Pas proposé en français
LFCS	Linux Foundation	Un examen pratique de deux heures, en ligne et surveillé, sur une ligne de commande Linux	Indépendant de la distribution. Valable deux ans
CompTIA Linux+	CompTIA	Examen XK0-006, depuis juillet 2025 : 90 minutes, jusqu'à 90 questions, à choix multiples et en situation	En anglais. Fait une large place aux conteneurs, à la sécurité, à l'automatisation (Ansible) et au dépannage
RHCSA	Red Hat	Examen EX200, entièrement pratique, sur Red Hat Enterprise Linux 10, sans accès à Internet	Tout se fait sur une vraie machine, et chaque réglage doit survivre au redémarrage

Le RHCSA mérite un mot. Ses objectifs recouvrent une grande partie de ce livre : shell et scripts, droits, processus, journaux, partitions et LVM, montages par UUID, NFS, minuteries systemd, comptes, clés SSH. Les différences tiennent aux outils propres à Red Hat : `dnf` et les paquets RPM au lieu d'APT, `firewalld` au lieu d'UFW, SELinux au lieu d'AppArmor, XFS comme système de fichiers par défaut. Ce sont d'autres commandes pour les mêmes idées. Un lecteur qui a fait les ateliers de ce livre a fait le plus gros du chemin, à condition de s'entraîner sur une machine RHEL, Rocky Linux ou AlmaLinux.

Par où commencer ? Pour faire reconnaître des compétences sur Ubuntu, la filière de Canonical. Pour un premier titre indépendant de la distribution, LPIC-1 ou LFCS ; le second, entièrement pratique, ressemble davantage au travail réel. Pour viser les entreprises qui tournent sous Red Hat, le RHCSA. Les tarifs varient selon les pays et changent souvent : consultez les sites des organismes.

Annexe E — Installer vite

L'aide-mémoire du jour où il faut une Ubuntu 24.04 tout de suite : sur une machine, dans une machine virtuelle, sous Windows, dans un conteneur ou chez un hébergeur, puis les dix premières minutes d'un serveur exposé à Internet. Chaque étape renvoie au chapitre qui l'explique et l'a vérifiée.

Choisir sa voie

Contexte	Pour	Ce qu'il faut	Chapitre
Machine physique	Un poste de travail, un serveur chez soi	Une clé USB de 8 Go, l'image Desktop ou Server	2
Machine virtuelle	Apprendre, essayer sans risque	VirtualBox, KVM ou Multipass, 4 Go de mémoire libres	2
WSL	La ligne de commande sous Windows	Windows 10 (version 2004 et suivantes) ou 11	2
Conteneur	Un environnement jetable, une application	Docker	28
Hébergeur	Un serveur sur Internet	Un compte chez l'hébergeur, une clé SSH	2, 19 à 21
Installation automatisée	Plusieurs machines identiques	Un fichier `user-data`	2, 32

L'image

Téléchargez-la sur `ubuntu.com/download` ou `releases.ubuntu.com/24.04/`, jamais ailleurs, avec le fichier `SHA256SUMS` du même dossier. Prenez la version ponctuelle la plus récente (24.04.5 à l'automne 2026).

```
$ sha256sum -c SHA256SUMS --ignore-missing
ubuntu-24.04.5-live-server-amd64.iso: OK
```

Sur un Mac à puce Apple, prenez l'image **arm64**.

Sur une machine physique

```
# Repérer la clé par sa taille et son modèle
lsblk -o NAME,SIZE,MODEL,TRAN

# L'écrire (ici sda : vérifiez deux fois)
sudo umount /dev/sda*
sudo dd if=ubuntu-24.04.5.1-desktop-amd64.iso of=/dev/sda bs=4M status=progress oflag=sync
```

> **DANGER · dd ne demande jamais confirmation**
>
> Un mauvais nom après `of=` efface le disque du système. balenaEtcher (Windows, macOS, Linux) et Rufus (Windows) font le même travail par une interface graphique.

Démarrez sur la clé avec la touche du menu de démarrage : F12 chez Dell et Lenovo, F9 chez HP, Échap ou F8 chez Asus.

Ubuntu Desktop, les choix qui comptent :

- Applications : la *sélection par défaut*.
- Logiciels propriétaires : les deux cases cochées sur un poste de travail.
- Disque : *Effacer le disque*, puis *Fonctions avancées*, **LVM et chiffrement** sur un portable. Notez la phrase secrète ailleurs : elle ne se récupère pas.
- Compte : un nom de machine en minuscules, sans accent ni espace.

Ubuntu Server, les choix qui comptent :

- *Network connections* : une adresse fixe se règle ici, en notation `192.168.1.0/24` pour le sous-réseau.
- *Guided storage* : *Use an entire disk* avec LVM. Le volume `ubuntu-lv` ne reçoit qu'une partie du disque : chapitre 15 pour l'agrandir.
- *Upgrade to Ubuntu Pro* : *Skip for now*.
- *SSH configuration* : cocher *Install OpenSSH server* ; *Import SSH key* récupère vos clés publiques depuis GitHub ou Launchpad.
- *Featured server snaps* : rien.

Dans une machine virtuelle

VirtualBox : *Nouvelle*, l'image, *Ignorer l'installation sans surveillance*, 4096 Mo de mémoire, 2 processeurs, 40 Go de disque alloué dynamiquement. Puis l'installateur, comme ci-dessus.

KVM, sur un poste Linux :

```
grep -E -c '(vmx|svm)' /proc/cpuinfo    # plus de 0 : le processeur sait virtualiser
sudo apt install qemu-kvm libvirt-daemon-system virtinst virt-manager
sudo usermod -aG kvm,libvirt $USER      # puis rouvrir la session
```

Multipass (`sudo snap install multipass`), sans installateur :

```
multipass launch 24.04 --name lab1 --cpus 2 --memory 4G --disk 20G
multipass shell lab1
multipass delete lab1 && multipass purge
```

Une fois le système à jour, prenez un instantané : vous y reviendrez après chaque exercice raté.

Sous Windows

Dans PowerShell, en administrateur :

```
wsl --install -d Ubuntu-24.04
```

Après le redémarrage, lancez « Ubuntu 24.04 » depuis le menu Démarrer et créez le compte, propre à cet Ubuntu. WSL suffit pour les chapitres 3 à 10 ; à partir du chapitre 11, passez à une machine virtuelle.

Dans un conteneur

```
$ docker run --rm ubuntu:24.04 bash -c \
    'grep PRETTY_NAME /etc/os-release; command -v systemctl || echo "pas de systemd"'
PRETTY_NAME="Ubuntu 24.04.5 LTS"
pas de systemd
```

`docker run -it --rm ubuntu:24.04 bash` ouvre un shell dans un système neuf, effacé à la sortie. L'image est réduite à l'essentiel et n'a pas de systemd : elle sert à essayer une commande ou à construire une application, pas à apprendre l'administration d'un serveur (chapitre 28).

Chez un hébergeur : les dix premières minutes

Avant de commander, créez votre clé sur le poste, avec une phrase secrète, et collez la partie publique dans le formulaire de l'hébergeur :

```
ssh-keygen -t ed25519 -C "gary@portable"
cat ~/.ssh/id_ed25519.pub
```

1. Un compte à soi. Si l'hébergeur vous donne `root` :

```
ssh root@203.0.113.42
apt update && apt full-upgrade -y
adduser gary
usermod -aG sudo gary
rsync --archive --chown=gary:gary ~/.ssh /home/gary
```

Dans un second terminal, vérifiez que `ssh gary@203.0.113.42` fonctionne avant d'aller plus loin. Gardez une session ouverte jusqu'à la fin : c'est elle qui vous sauvera d'une erreur de configuration.

2. SSH : des clés seulement, jamais root. Le fichier du chapitre 19, `/etc/ssh/sshd_config.d/10-durcissement.conf` :

```
# Connexion par clé uniquement, jamais en root, réservée à un groupe
PasswordAuthentication no
PermitRootLogin no
AllowGroups acces-ssh
MaxAuthTries 3
X11Forwarding no
```

```
sudo groupadd acces-ssh
sudo usermod -aG acces-ssh gary
sudo sshd -t && sudo systemctl reload ssh
```

`sshd -t` ne dit rien quand la configuration est correcte. Ouvrez une nouvelle connexion pour vérifier avant de fermer l'ancienne.

3. **Le pare-feu** (chapitre 21) :

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow OpenSSH
sudo ufw enable
sudo ufw status verbose
```

Ajoutez ensuite les services réellement offerts, par exemple `sudo ufw allow 'Nginx Full'` pour un site web.

4. **Fail2ban**, qui surveille SSH dès son installation :

```
sudo apt install fail2ban
sudo fail2ban-client status
```

5. **Les mises à jour de sécurité automatiques**, actives d'origine. Vérifiez-le :

```
systemctl is-enabled unattended-upgrades
sudo unattended-upgrade --dry-run -v 2>&1 | head -n 6
```

6. **Le nom et l'heure** (chapitre 18) :

```
sudo hostnamectl set-hostname web1
sudo timedatectl set-timezone Africa/Douala
```

Le chapitre 20 va plus loin : politique de mots de passe, réglages du noyau, audit, et la mesure de tout cela avec Lynis.

L'installation automatisée

Pour installer plusieurs machines de la même façon, on écrit les réponses de l'installateur dans un fichier `user-data`. Voici celui du chapitre 32, qui a installé le serveur personnel sans une question :

```
#cloud-config
autoinstall:
  version: 1
  locale: fr_FR.UTF-8
  keyboard:
    layout: fr
  timezone: Africa/Douala
  identity:
```

```
    hostname: perso
    username: gary
    # mkpasswd -m sha-512
    password: "$6$FugzxAXDKRleb/
t2$4iN9xBFHvctSO2PbGoMq.XtYAMqz0LeBu2WxiL23tn440U.qFiJZypoVNDKRRRsPsZvxWxe7h
Mw2OHUTCVlHO0"
  ssh:
    install-server: true
    allow-pw: false
    authorized-keys:
      - ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIPgNVDIjKQsmt8dDpBT49TVl5I8/33ZXLd3lbTojso7Y
gary@vostro
  network:
    version: 2
    ethernets:
      lan:
        match:
          name: "en*"
        dhcp4: true
  storage:
    layout:
      name: lvm
      sizing-policy: all
  updates: security
  shutdown: poweroff
```

Clé	Effet
`identity`	Nom de la machine, compte, mot de passe chiffré (`mkpasswd -m sha-512`, du paquet `whois`)
`ssh`	Le serveur SSH, sans mot de passe, avec votre clé publique
`network`	La carte désignée par motif de nom : la machine reste joignable même si le nom change après l'installation
`storage`	LVM sur tout le disque
`updates`, `shutdown`	Les mises à jour de sécurité, puis l'extinction

Le fichier se place sous le nom `user-data`, avec un fichier `meta-data` vide, sur une clé USB dont le volume s'appelle `CIDATA`. L'installateur demande encore une confirmation, sauf si l'on ajoute `autoinstall` à la ligne du noyau dans le menu de démarrage. Il peut aussi le lire sur un serveur web du réseau :

```
autoinstall ds=nocloud-net\;s=http://192.168.1.10/
```

La barre oblique inversée devant le point-virgule est indispensable.

La liste de contrôle

Vérification	Attendu
`lsb_release -a`	Ubuntu 24.04
`sudo apt update`	Se termine sans erreur
`ping -c 3 ubuntu.com`	Des réponses : le réseau et le DNS fonctionnent
`whoami`, puis `sudo whoami`	Votre identifiant, puis `root` après votre mot de passe
Sur un serveur : `ssh root@serveur` depuis le poste	Refusé
Sur un serveur : `sudo ufw status`	`Status: active`, seuls les services voulus ouverts
Sur un serveur : `sudo fail2ban-client status sshd`	La prison `sshd` active
Dans une machine virtuelle	Un instantané « installation propre »

Table des matières

Avant-propos vii

PARTIE I FONDATIONS

1 Linux, Ubuntu et vous 3
- 1.1 Qu'est-ce qu'un système d'exploitation ? 4
- 1.2 D'Unix à Ubuntu : une brève histoire 7
- 1.3 Logiciel libre et open source 11
- 1.4 L'écosystème Linux aujourd'hui 14
- 1.5 Pourquoi Ubuntu ? 17
- 1.6 Ubuntu 24.04 LTS « Noble Numbat » 20
- 1.7 Quelle édition choisir ? 22

2 Installer Ubuntu 24.04 27
- 2.1 Préparer le terrain 28
- 2.2 Créer une clé USB d'installation 31
- 2.3 Installer Ubuntu Desktop 33
- 2.4 Installer Ubuntu Server 36
- 2.5 Installer dans une machine virtuelle 38
- 2.6 Ubuntu sous Windows avec WSL 40
- 2.7 Ubuntu chez un hébergeur 41
- 2.8 L'installation automatisée 43
- 2.9 Le double amorçage avec Windows 44
- 2.10 Après l'installation 45

3 Premiers pas dans l'environnement 51
- 3.1 Le bureau GNOME 52
- 3.2 Ouvrir un terminal 55
- 3.3 Lire l'invite de commande 56
- 3.4 Les commandes de tous les jours 57
- 3.5 Trouver de l'aide 61
- 3.6 Les éditeurs de texte 64
- 3.7 Les logiciels fournis 66
- 3.8 Installer un premier logiciel 67

PARTIE II L'ENVIRONNEMENT UBUNTU

4 Le système de fichiers 77
- 4.1 « Tout est fichier » 78
- 4.2 L'arborescence standard 79

4.3 Chemins absolus et chemins relatifs ... 85
4.4 Fichiers cachés et liens ... 86
4.5 Les types de fichiers ... 89
4.6 L'inode ... 90

5 Les commandes essentielles du shell ... 97
5.1 Lister et visualiser ... 98
5.2 Créer, copier, déplacer, supprimer ... 99
5.3 Lire des fichiers ... 102
5.4 Trouver des fichiers et des commandes ... 104
5.5 Connaître l'état de la machine ... 106
5.6 Redirections et tubes ... 108
5.7 Les motifs de noms de fichiers ... 112
5.8 L'historique et l'édition de la ligne ... 113
5.9 Alias et variables d'environnement ... 115
5.10 Les archives ... 116

6 Utilisateurs, groupes et permissions ... 123
6.1 Utilisateurs et groupes ... 124
6.2 Les trois fichiers des comptes ... 125
6.3 Gérer les comptes ... 127
6.4 Gérer les groupes ... 129
6.5 Les permissions ... 130
6.6 La notation octale et la notation symbolique ... 131
6.7 chmod, chown, chgrp ... 133
6.8 Les droits spéciaux ... 134
6.9 Les droits par défaut : umask ... 136
6.10 Les ACL : au-delà de trois catégories ... 137
6.11 sudo ... 139
6.12 Changer d'identité ... 140

7 La gestion des paquets ... 145
7.1 Pourquoi un gestionnaire de paquets ... 146
7.2 Les dépôts d'Ubuntu ... 148
7.3 La liste des dépôts ... 149
7.4 APT au quotidien ... 150
7.5 PPA et dépôts tiers ... 153
7.6 Installer un fichier .deb ... 155
7.7 Snap ... 156
7.8 Flatpak ... 157
7.9 AppImage ... 158
7.10 Compiler depuis les sources ... 159
7.11 Les mises à jour automatiques ... 160

PARTIE III MAÎTRISE DU SHELL BASH

8 Le shell en profondeur ... 169
8.1 Qu'est-ce qu'un shell ? ... 170
8.2 Les fichiers de configuration ... 171
8.3 Les variables ... 173
8.4 PATH : comment Bash trouve une commande ... 175
8.5 De la ligne tapée à la commande exécutée ... 176
8.6 Les guillemets ... 179
8.7 L'historique ... 180
8.8 La complétion ... 181
8.9 L'invite ... 182
8.10 tmux ... 183

9 Traiter du texte en ligne de commande ... 189
9.1 grep : chercher ... 190
9.2 Les expressions régulières ... 191
9.3 sed : transformer ... 194
9.4 awk : exploiter des colonnes ... 196
9.5 Découper, trier, compter ... 198
9.6 Comparer ... 200
9.7 Encodages et fins de ligne ... 202
9.8 Mettre en forme ... 203
9.9 jq : lire du JSON ... 203
9.10 yq : lire du YAML ... 204

10 Écrire des scripts Bash ... 211
10.1 Anatomie d'un script ... 212
10.2 Variables et tableaux ... 213
10.3 Lire des entrées ... 215
10.4 Les conditions ... 217
10.5 Les boucles ... 219
10.6 case ... 222
10.7 Les fonctions ... 223
10.8 Codes de retour et gestion des erreurs ... 224
10.9 Déboguer ... 227
10.10 De bonnes habitudes ... 227
10.11 Des scripts que l'on peut relancer ... 228
10.12 Quand quitter Bash ... 229

PARTIE IV SYSTEMD ET GESTION DU SYSTÈME

11 Le démarrage du système ... 237
11.1 Les six étapes du démarrage ... 238

11.2 GRUB ... 240
11.3 Secure Boot ... 242
11.4 L'initramfs ... 243
11.5 Le noyau et ses modules ... 244
11.6 Les cibles de systemd ... 245
11.7 Démarrer en mode de secours ... 247
11.8 Mesurer le temps de démarrage ... 248

12 Processus et services ... 253
12.1 Qu'est-ce qu'un processus ? ... 254
12.2 Observer les processus ... 256
12.3 Les signaux ... 260
12.4 Régler les priorités ... 262
12.5 Premier plan et arrière-plan ... 263
12.6 systemd et ses unités ... 265
12.7 Piloter les services avec systemctl ... 266
12.8 Anatomie d'une unité de service ... 269
12.9 Retoucher une unité sans la modifier ... 271
12.10 Mesurer et plafonner les ressources ... 272
12.11 Cloisonner un service ... 274
12.12 Lancer une commande comme un service ... 276

13 Journaux et planification ... 285
13.1 Où Ubuntu écrit ses journaux ... 286
13.2 Lire le journal avec journalctl ... 287
13.3 Écrire dans le journal ... 290
13.4 La place occupée par le journal ... 291
13.5 Renouveler les fichiers avec logrotate ... 292
13.6 rsyslog, pour trier et centraliser ... 294
13.7 Planifier avec cron ... 295
13.8 Une seule fois : at ... 299
13.9 Planifier avec systemd : les minuteries ... 299

PARTIE V STOCKAGE ET SYSTÈMES DE FICHIERS

14 Disques, partitions et systèmes de fichiers ... 311
14.1 Du disque au dossier ... 312
14.2 Reconnaître les disques ... 313
14.3 Deux tables de partitions : MBR et GPT ... 314
14.4 Partitionner ... 315
14.5 Choisir un système de fichiers ... 316
14.6 Créer le système de fichiers ... 317
14.7 Monter et démonter ... 318

14.8 /etc/fstab : les montages permanents ... 320
14.9 L'espace d'échange ... 323
14.10 Vérifier et surveiller ... 324

15 LVM, RAID et chiffrement ... 331

15.1 Pourquoi LVM ... 332
15.2 Trois étages ... 332
15.3 LVM est déjà là : agrandir la racine ... 333
15.4 Construire un groupe de volumes ... 334
15.5 Faire évoluer le groupe ... 335
15.6 Les instantanés ... 337
15.7 Le RAID logiciel ... 338
15.8 Chiffrer un disque avec LUKS ... 341
15.9 Empiler les couches ... 343

16 Sauvegarde et restauration ... 351

16.1 Trois copies, deux supports, un autre lieu ... 352
16.2 Que sauvegarder ? ... 353
16.3 Complète, incrémentale, différentielle ... 354
16.4 rsync : copier ce qui a changé ... 355
16.5 tar : tout dans un fichier ... 357
16.6 restic : la sauvegarde moderne ... 358
16.7 Sauvegarder un volume en cours d'utilisation ... 362
16.8 Automatiser et surveiller ... 363
16.9 Tester la restauration ... 363

PARTIE VI RÉSEAU SOUS UBUNTU

17 Les fondamentaux du réseau ... 373

17.1 Les couches ... 374
17.2 Les adresses IPv4 ... 375
17.3 IPv6 en bref ... 377
17.4 Les ports, TCP et UDP ... 378
17.5 Le DNS ... 379
17.6 Examiner la machine avec ip ... 380
17.7 Tester la liaison ... 382
17.8 Interroger le DNS ... 383
17.9 Ports et services ... 385
17.10 Regarder passer les paquets ... 386

18 Configurer le réseau sous Ubuntu ... 393

18.1 Qui fait quoi ... 394
18.2 Lire la configuration en place ... 395
18.3 Une adresse fixe ... 395

18.4 Appliquer sans se couper du serveur ... 397
18.5 Les serveurs DNS ... 399
18.6 Le nom de la machine ... 400
18.7 Sur un poste de travail : NetworkManager ... 401
18.8 Au-delà d'une carte ... 402

19 SSH ... 407
19.1 Ce que fait SSH ... 408
19.2 La première connexion ... 409
19.3 Les clés ... 410
19.4 Le fichier de configuration du client ... 412
19.5 Régler le serveur ... 413
19.6 Copier des fichiers ... 416
19.7 Les tunnels ... 417
19.8 Le rebond ... 419
19.9 Un second facteur ... 419

PARTIE VII SÉCURITÉ DU SYSTÈME

20 Durcir le système ... 427
20.1 Trois principes ... 428
20.2 Les mises à jour de sécurité ... 429
20.3 Les comptes et les mots de passe ... 430
20.4 sudo, au plus juste ... 431
20.5 Les fichiers sensibles ... 433
20.6 Moins de services, moins de ports ... 433
20.7 Les réglages du noyau ... 434
20.8 Les options de montage ... 435
20.9 Les périphériques USB ... 436
20.10 Un avertissement à l'entrée ... 437
20.11 Garder la trace ... 437
20.12 Mesurer : Lynis ... 438

21 Pare-feu et protection du réseau ... 445
21.1 Ce que fait un pare-feu ... 446
21.2 UFW ... 447
21.3 Sous le capot : nftables ... 451
21.4 Fail2ban ... 453

22 AppArmor et le confinement ... 461
22.1 Un second contrôle ... 462
22.2 L'état des lieux ... 463
22.3 Lire un profil ... 463
22.4 Voir un refus ... 465

22.5 Les modes ... 466
22.6 Les espaces de noms sous Ubuntu 24.04 ... 468
22.7 Écrire un profil ... 469

PARTIE VIII SERVICES SERVEUR ESSENTIELS

23 Serveurs web ... 479
23.1 Demande et réponse ... 480
23.2 Nginx sur Ubuntu ... 481
23.3 Anatomie d'une configuration ... 482
23.4 Servir un site statique ... 483
23.5 Relayer une application ... 486
23.6 Plusieurs sites sur un serveur ... 490
23.7 HTTPS ... 491
23.8 Vitesse et discrétion ... 496
23.9 Les journaux ... 498
23.10 Apache, pour ceux qui en héritent ... 500
23.11 Caddy et Traefik ... 502

24 Bases de données ... 509
24.1 Quelle base pour quel besoin ... 510
24.2 PostgreSQL ... 511
24.3 Ouvrir la base au réseau ... 516
24.4 Sauvegarder et restaurer ... 519
24.5 Régler la mémoire ... 522
24.6 MariaDB ... 523
24.7 Redis ... 526
24.8 SQLite ... 527

25 Partager des fichiers sur le réseau ... 533
25.1 Quatre façons de partager ... 534
25.2 NFS ... 534
25.3 Samba ... 538
25.4 SFTP, pour déposer à distance ... 543
25.5 WebDAV, quand seul le Web passe ... 544
25.6 Lequel choisir ... 546

26 Superviser un serveur ... 551
26.1 Ce qu'on surveille, et pourquoi ... 552
26.2 Lire la machine sur le moment ... 552
26.3 L'historique avec sar ... 557
26.4 Prometheus ... 557
26.5 Les alertes ... 560
26.6 Le signe de vie ... 564

26.7 Grafana ... 565
26.8 Surveiller depuis l'extérieur ... 568
26.9 D'autres outils ... 569

PARTIE IX AUTOMATISATION ET DEVOPS

27 Git et le suivi des versions ... 577
27.1 Pourquoi suivre les versions ... 578
27.2 Installer et configurer ... 578
27.3 Les trois zones ... 579
27.4 Le premier dépôt ... 580
27.5 Les branches ... 583
27.6 Faire vérifier ses scripts ... 586
27.7 Partager un dépôt ... 587
27.8 Les plateformes ... 590
27.9 Suivre /etc avec etckeeper ... 591

28 Les conteneurs avec Docker ... 597
28.1 Conteneur ou machine virtuelle ... 598
28.2 Le vocabulaire ... 599
28.3 Installer Docker Engine ... 599
28.4 Premiers pas ... 601
28.5 Docker et le pare-feu ... 602
28.6 Les volumes ... 603
28.7 Les réseaux ... 605
28.8 Construire une image ... 605
28.9 Docker Compose ... 607
28.10 Traefik, le relais qui lit les étiquettes ... 612
28.11 De bonnes habitudes ... 613

29 Automatiser avec Ansible ... 619
29.1 Décrire plutôt que faire ... 620
29.2 Installer Ansible ... 621
29.3 L'inventaire ... 622
29.4 Des commandes ponctuelles ... 623
29.5 Le premier playbook ... 624
29.6 Les modules essentiels ... 626
29.7 Un rôle : le durcissement du chapitre 20 ... 627
29.8 Les secrets avec Ansible Vault ... 636
29.9 L'idempotence, en pratique ... 637

PARTIE X DIAGNOSTIC, PERFORMANCE, RÉCUPÉRATION

30 Diagnostiquer un incident 645
30.1 Une méthode 646
30.2 Le triage 647
30.3 Les journaux 647
30.4 La performance 649
30.5 Le réseau 650
30.6 Ce que fait un programme 651
30.7 Des pannes, de bout en bout 654
30.8 Le retour d'expérience 659

31 Récupérer et sauver 665
31.1 Avant de toucher à quoi que ce soit 666
31.2 GRUB sans configuration 666
31.3 Le mot de passe perdu 668
31.4 Démarrer sur un système de secours 670
31.5 Un système de fichiers abîmé 674
31.6 Un disque qui flanche 675
31.7 Des fichiers effacés 677
31.8 Un volume LVM supprimé 678
31.9 Un RAID dégradé 679
31.10 Quand appeler à l'aide 680

PARTIE XI PROJETS GUIDÉS

32 Projet : un serveur personnel 687
32.1 Le projet 688
32.2 Le matériel 689
32.3 Installer sans intervenir 690
32.4 Durcir et préparer avec Ansible 693
32.5 Le disque des sauvegardes 694
32.6 Les services 695
32.7 Surveiller 702
32.8 Sauvegarder 703
32.9 Accéder de l'extérieur avec WireGuard 705
32.10 Entretenir 707

33 Projet : un poste de travail 711
33.1 Le bureau 712
33.2 Le terminal 713
33.3 L'environnement de développement 715
33.4 Mots de passe et fichiers 718
33.5 Sauvegarder le poste 719

33.6 Les réglages, versionnés 720
33.7 D'où installer quoi 721

ANNEXES

Annexe A — Aide-mémoire des commandes 729

Annexe B — Référence systemd 737

Annexe C — Glossaire 753

Annexe D — Ressources pour continuer 767

Annexe E — Installer vite 773

Maîtriser Ubuntu 24.04 LTS

Du premier démarrage à un serveur exposé à Internet, une seule version d'Ubuntu, suivie de bout en bout.

Installer Ubuntu 24.04 sur un poste ou un serveur, maîtriser la ligne de commande et écrire des scripts Bash, comprendre systemd, gérer disques, LVM, RAID et sauvegardes, configurer le réseau et SSH, durcir le système, déployer un site web, une base de données et la supervision, puis passer à Git, Docker et Ansible : ce livre accompagne le lecteur pas à pas, en expliquant le pourquoi avant le comment.

Chaque commande a été exécutée sur Ubuntu 24.04 LTS, et chaque sortie reproduite telle qu'elle s'est affichée. Trente-trois chapitres, des ateliers et des exercices, deux projets complets — un serveur personnel auto-hébergé et un poste de travail professionnel — et cinq annexes de référence.

Pour les débutants, les développeurs, les administrateurs Windows qui passent à Linux, les futurs administrateurs et ingénieurs DevOps, et tous ceux qui hébergent leurs propres services.

Livre gratuit, diffusé sous licence CC BY-NC-ND 4.0 : il se partage librement, il ne se vend pas.

WSC DIGITAL