Introduction à Linux

Pierre Senellart

Semaine *Informatique pratique*, septembre 2022

Unix

1969 première version d'UNICS (Unix) par Ken Thompson (rejoint ensuite par Dennis Ritchie, puis d'autres), simplification du système MULTICS, au sein d'AT&T Bell Labs

1977 développement (projet mené par Ken Thompson à l'Université de Californie à Berkeley) de la *Berkeley Software Distribution* (BSD), collection de logiciels pour Unix, remplaçant progressivement l'ensemble du système

1983 en parallèle, sortie d'Unix System V par AT&T, succès commercial

Deux branches qui ont donné lieu à de nombreux systèmes Unix :

BSD FreeBSD, NetBSD, OpenBSD (libres), MacOS X (Apple), iOS (Apple)

System V AIX (IBM), Solaris (Oracle), HP-UX (HP)

Linux

1983 Richard Stallman lance le projet GNU d'un système d'exploitation libre, compatible avec Unix ; focus sur le développement d'outils (compilateurs, éditeurs, shell, etc.) plutôt que sur le noyau du système

1987 Andrew Tanenbaum développe MINIX, système d'exploitation inspiré d'Unix mais minimaliste, à des fins d'enseignement (non libre à l'époque)

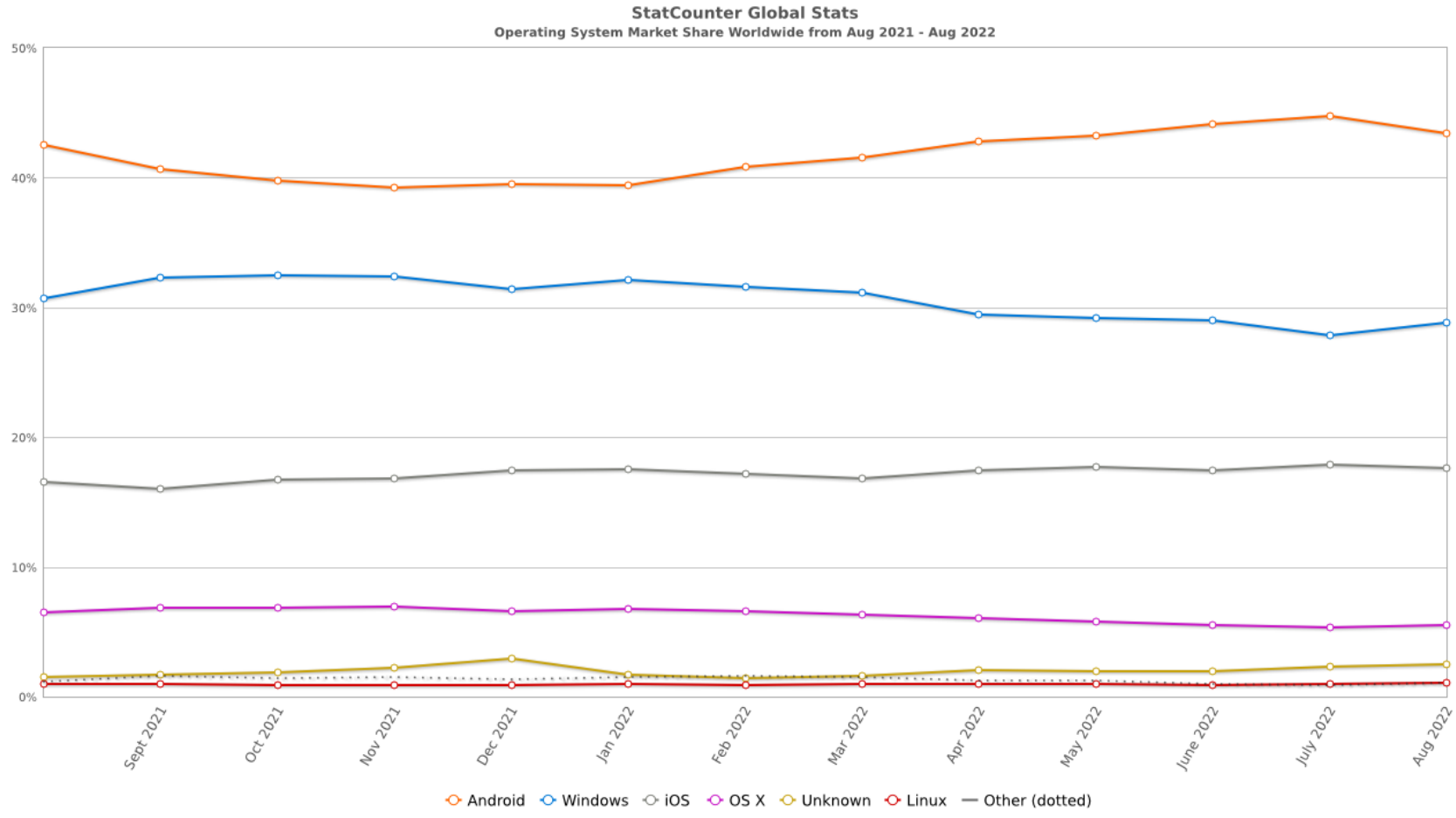
1991 Linus Torvalds développe le noyau du système d'exploitation Linux comme logiciel libre, inspiré par MINIX, et choisit GNU comme utilitaires pour le système, d'où le nom GNU/Linux parfois utilisé

2008 Android développé par Google, comme système d'exploitation dérivé de Linux (modification du noyau + outils annexes)

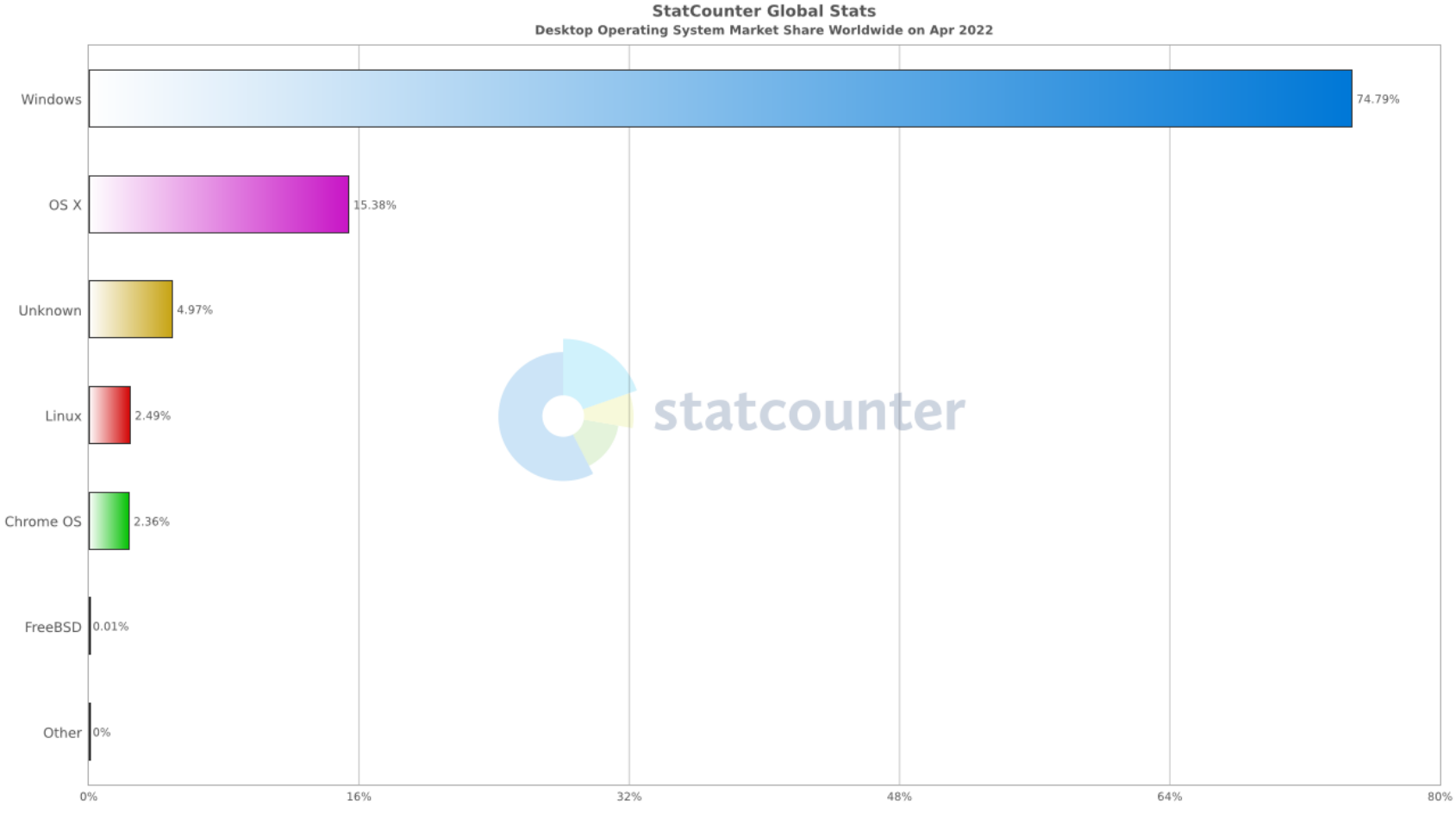
Unix vs Linux

- Contrairement à FreeBSD, Solaris, etc., Linux n'est pas directement un descendant d'Unix, et on l'oppose souvent aux systèmes Unix
- Mais séparation assez artificielle, il y a sans doute plus de similarité entre FreeBSD et Linux qu'entre FreeBSD et AIX
- POSIX (Portable Operating System Interface) : famille de standards tentant d'uniformiser l'interface utilisateur et de développement des systèmes Unix/Linux ; la plupart des systèmes Unix commerciaux sont certifiés POSIX ; les systèmes Unix libres et Linux sont largement conformes à POSIX
- Un environnement POSIX est même disponible sous Windows, via WSL (Windows Subsystem for Linux)

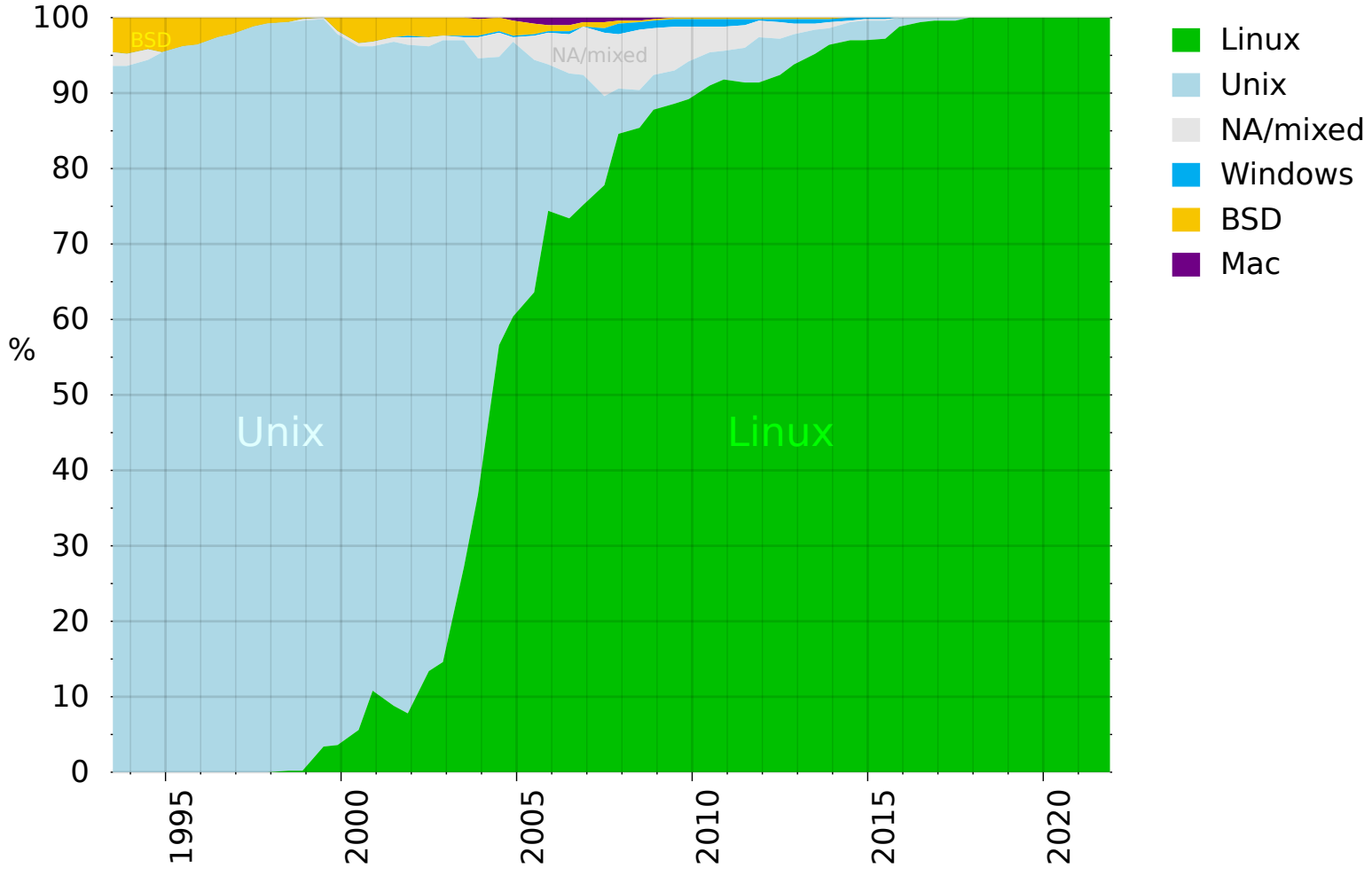
Part de marché OS grand public (accès Web)



Part de marché OS hors OS mobiles (accès Web)



Part de marché supercalculateurs (top 500)



Pourquoi apprendre à utiliser Linux ?

- Système libre, gratuit, installable sur une large gamme de machines
- Fournit un environnement (basé sur une utilisation en ligne de commande) efficace pour le développement logiciel, le traitement de fichiers, l'analyse de données
- Système utilisé dans les salles de TP !
- Système dominant sur les serveurs (Web, de calcul, etc.)
- Bon environnement d'apprentissage d'un système d'exploitation (code source disponible, accès à la documentation, facile à étendre)
- Se familiariser avec Linux aide pour comprendre les systèmes Unix, les systèmes dérivés de Linux

Distribution

- Distribution Linux : système d'exploitation incluant un noyau Linux, une collection d'utilitaires (souvent GNU), de logiciels de tout type, un système de mise à jours et d'installations de nouveaux logiciels (gestionnaire de paquets), un environnement graphique (pour les distributions pour ordinateur personnel), un installeur...
- Des centaines de distributions :

Ubuntu : la plus courante, dérivée de Debian, nouvelle toutes les six mois, `apt`

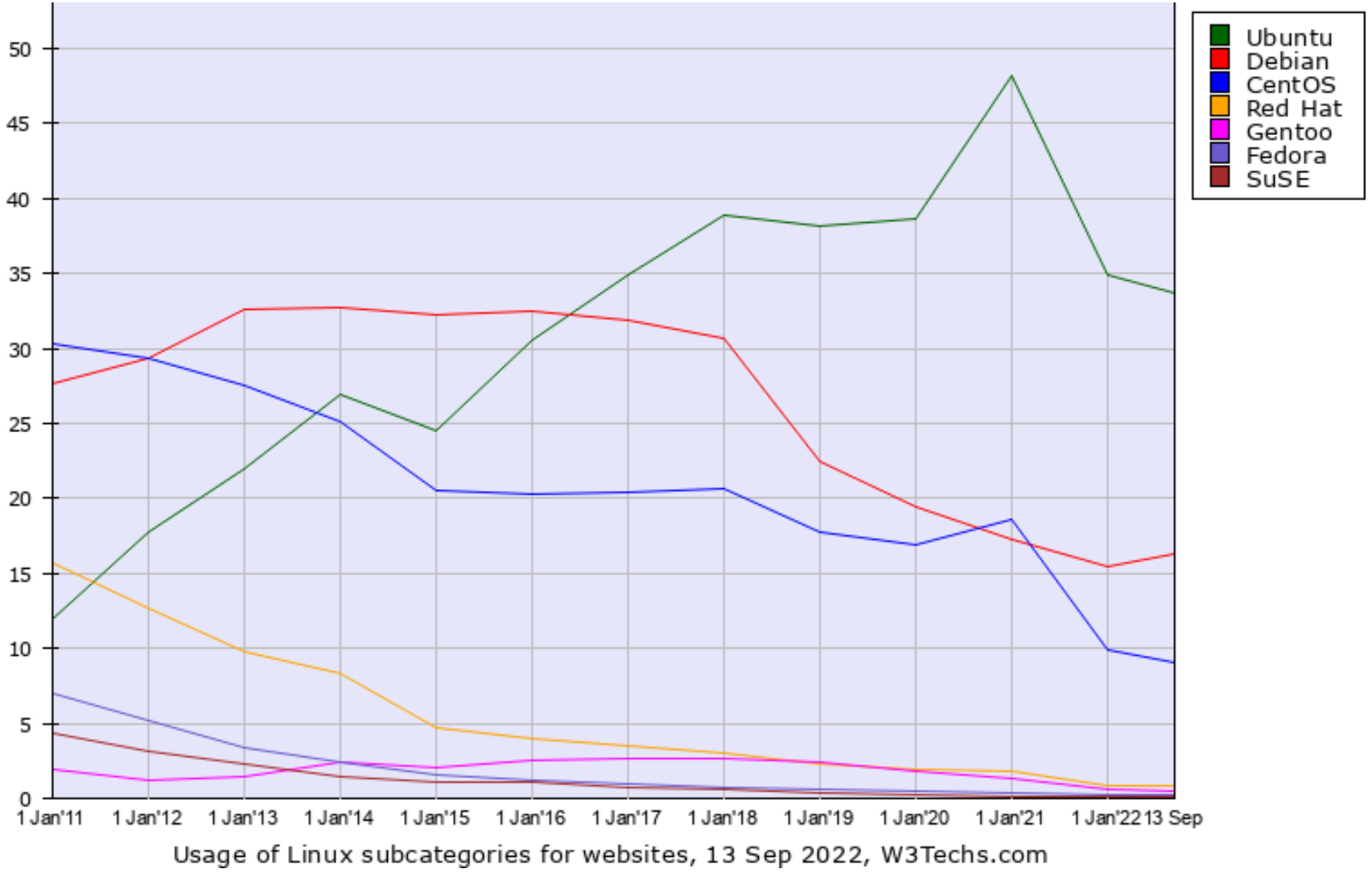
Debian : historique (1993), courante sur les serveurs, nouvelles versions toutes les quelques années, `apt`

Fedora : version gratuite d'une distribution commerciale, Red Hat Entreprise Linux, `yum`

Arch Linux : minimaliste, évolutions continues sans version à date fixe, `pacman`

Gentoo : paquets distibués via leur source, compilés localement, `emerge` (Portage)

Part de marché des distributions (accès Web)



Pour démarrer

- Distributions modernes facilement installables sur des ordinateurs de bureau ou des ordinateurs portables (via une clef USB contenant l'installeur)
- Peut s'installer à la place d'un Windows (ou d'un Mac) sur un espace dédié du disque, avec possibilité de démarrer chacun des deux systèmes
- Peut aussi être installé à l'intérieur d'une machine virtuelle, par exemple en utilisant VirtualBox, sans toucher le système d'origine
- Des install parties sont régulièrement organisées !

Plan

Vocabulaire de base

Processus : abstraction d'une instance d'un programme en cours d'exécution

Fichier : abstraction d'une collection de données (souvent, mais pas toujours, stockée sur un support physique, tel un disque dur), auquel on associe un nom

Système de fichier : organisation hiérarchique d'un ensemble de fichiers et spécification de comment cet ensemble de fichiers est stocké sur un certain média

Chemin : emplacement d'un fichier dans le système de fichier

Périphérique : n'importe quel composant matériel d'un ordinateur qui ne soit pas un de ses composants primaires sans lequel l'ordinateur ne peut fonctionner (processeur, mémoire vive, alimentation, etc.)

Espace noyau et espace utilisateur

- Noyau : cœur du système d'exploitation, fonctions de base permettant de gérer les processus de manière multitâche, d'accéder aux périphériques d'entrée–sortie et autre matériel, les accès aux fichiers, etc.
- Le système Linux découpe la mémoire (virtuellement) entre :

  Espace noyau : espace de la mémoire contenant le noyau et réservée au noyau

  Espace utilisateur : espace dans lequel l'ensemble des autres processus s'exécute
- Seule manière pour les processus de l'espace utilisateur de communiquer avec le noyau : appels systèmes
- Isolation assurée par le système, mais aussi par le processeur (anneau de protection)

Appels systèmes

- Utilisés dès qu'un processus doit communiquer avec le noyau pour accéder au système de fichiers, à un périphérique, établir une connexion réseau, communiquer avec un autre processus, etc.
- Voir `man syscalls`
- Généralement non portables, mais beaucoup suivent la spécification POSIX
- Souvent, ces fonctions ne sont pas appelées directement par le programmeur, mais sont appelées par des fonctions de plus haut niveau dans la bibliothèque standard d'un langage de programmation
- `strace` permet de voir l'ensemble des appels systèmes fait par un processus

Initialisation du système Linux (1/2)

- À l'initialisation d'un ordinateur, un mini-système d'exploitation (BIOS traditionnellement sur les PC, maintenant UEFI) dont le code (firmware) est stocké sur la carte-mère, se lance, initialise les composants de l'ordinateur, teste la présence de périphériques, etc.
- Le mini-système passe ensuite le contrôle à un chargeur d'amorçage (stocké sur le premier secteur d'un disque, ou parfois chargé depuis le réseau) tel `grub` (tout en restant accessible pour faire l'interface entre noyau du système d'exploitation et matériel)
- Le chargeur d'amorçage peut laisser le choix entre plusieurs systèmes d'exploitations, puis charge en mémoire le noyau correspondant

Initialisation du système Linux (2/2)

- Le noyau charge en mémoire un mini-système de fichiers contenant un système minimal (initial RAM disk) puis l'utilise pour finir de s'initialiser, d'accéder aux différents périphériques, etc.
- Le noyau passe ensuite le contrôle (mais reste dans l'espace noyau) au premier processus utilisateur : `init` traditionnellement ou, plus récemment `systemd`
- Ce processus lance un certain nombre de services (ou démons), c.-à-d., de programmes autonomes ; parmi les derniers, un environnement pour les utilisateurs (un environnement graphique ou un accès à une ligne de commande) – avec `systemd`, utiliser `systemctl` pour la liste des services chargés ; avec `init`, aller voir dans `/etc/rc*.d`

Utilisateurs

- Linux est un système multi-utilisateurs
- Chaque utilisateur est représenté par un login et un identifiant numérique (UID)
- Processus d'authentification (par ex., par mot de passe)
- Il existe aussi des groupes d'utilisateurs (nom de groupe, GID)
- Chaque utilisateur appartient à un ou plusieurs groupes
- Un utilisateur spécial : `root`, le superutilisateur (d'UID 0)
- Pour devenir superutilisateur, on peut se connecter comme tel, ou, ponctuellement, lancer des commandes préfixées de `sudo`

Environnement graphique

Sous Linux, un environnement graphique est formé de :

- un système de fenêtrage (X11 traditionnellement, parfois Wayland) qui se charge de fournir les éléments de base d'une interface graphique : affichage de fenêtres, gestion de la souris et du clavier, etc.
- un gestionnaire de fenêtres (metacity, KWin, i3, fvwm, xfwm, Xmonad, etc.) qui se charge de décorer les fenêtres, les positionner à l'écran, fournir une interface de base type menus ou raccourcis claviers
- éventuellement, un environnement de bureau (Gnome, KDE, XFCE, etc.) qui fournit un environnement graphique avec gestionnaire de fichiers, menu des applications, interface de configuration, etc.
- souvent, un gestionnaire d'identification (gdm, kdm, xdm) qui fournit une interface graphique pour se connecter à la machine

Copier–coller sous X11

- Deux manières sous X11 de copier–coller, avec deux tampons différents :

Tampon	Copier	Couper	Coller
Primary	Sélection à la souris	SHIFT+DEL	Clic molette ou SHIFT+INS
Clipboard	CTRL+c	CTRL+x	CTRL+v

- Dans les terminaux, comme CTRL+...peut être utilisé pour autre chose, on utilise CTRL+SHIFT+...pour Clipboard

Plan

Du programme au processus

- Un programme est un fichier contenant du code exécutable par le système d'exploitation, directement ou une fois interprété par un autre programme ; pour être lancé, ce fichier doit avoir la permission exécutable (voir plus loin) ; le noyau (via un des appel système `exec`) lance une instance de ce programme, qui devient un processus
- Si ce fichier commence par les deux caractères `#!` (shebang) ce qui suit jusqu'au premier caractère retour à la ligne est le chemin d'un autre exécutable utilisé pour interpréter le contenu de ce fichier
- Sinon, le fichier doit contenir du code machine directement exécutable par le système (en général au format ELF)
- Le noyau affecte un PID entier (identifiant de processus) au processus lancé (le premier processus lancé, `init`, a le PID 1)

Espace mémoire d'un processus

- Le noyau alloue au processus une portion de l'espace mémoire, pour y mettre :
  - le code du processus
  - l'environnement du processus (voir ci-après)
  - les arguments passés au processus
  - les variables globales du processus
  - la pile, utilisée pour les appels de fonction et les variables locales
  - le tas, utilisé pour l'allocation dynamique de mémoire
- le code des bibliothèques dynamiques chargées par ce processus est également chargé en mémoire (partagée avec d'autre processus utilisant la même bibliothèque)

Environnement

- Ensemble d'informations textuelles de type « VARIABLE=valeur » disponible à un processus
- Par défaut, l'environnement est propagé à l'ensemble des processus lancés par un processus donné
- Nombreuses variables d'environnement ayant un rôle particulier (pour les appels systèmes, les fonctions de bibliothèques classiques, le chargeur d'exécutable...)
- Aucune garantie qu'une variable d'environnement soit présente

Variables d'environnement importantes

Variable	Contenu
`PATH`	Chemins dans lesquels les exécutables sont recherchés
`HOME`	Chemin du répertoire personnel de l'utilisateur
`PWD`	Chemin actuel
`SHELL`	Shell actuel
`USER`	Identifiant de l'utilisateur actuel
`TZ`	Fuseau horaire

Locale

Variables d'environnements utilisés pour indiquer des préférences d'internationalisation du comportement des programmes :

Variable	Contenu
`LC_CTYPE`	Interprétation des octets en caractères, des classes de caractère
`LC_COLLATE`	Règles de tri et d'égalité de chaînes de caractère
`LC_MESSAGES`	Langue des messages
`LC_NUMERIC`	Formats numériques
`LC_TIME`	Format des dates
`LANG`	Valeur par défaut de ces variables
`LC_ALL`	Valeur imposée à toutes ces variables

Les valeurs possibles sont disponibles via `locale -a`, les valeurs actuelles effectives via `locale`. La locale `C` est la plus simple.

Descripteurs de fichiers

- Un processus va le plus souvent ouvrir des fichiers pour y lire ou écrire
- Chaque fichier ouvert est associé à un descripteur de fichier, un entier
- Trois descripteurs de fichiers standard, qui sont souvent pré-ouverts :

0	Entrée standard
1	Sortie standard
2	Sortie d'erreur

- Par défaut, un programme lit sur son entrée standard, écrit sur sa sortie standard, et affiche les messages d'erreur sur sa sortie d'erreur

Parent et héritage

- Chaque processus est lancé par un autre processus (sauf `init`/`systemd`), qui devient son parent
- Un processus hérite de son parent (par défaut, configurable) :
  - Son environnement (y compris sa locale)
  - L'identité de l'utilisateur qui l'a lancé, et les permissions dont dispose cet utilisateur
  - Les fichiers ouverts, y compris les descripteurs de fichiers standard (voir plus loin)
  - Le terminal auquel le processus est attaché, s'il existe (voir plus loin)

Parent et héritage

- Chaque processus est lancé par un autre processus (sauf `init`/`systemd`), qui devient son parent
- Un processus hérite de son parent (par défaut, configurable) :
  - Son environnement (y compris sa locale)
  - L'identité de l'utilisateur qui l'a lancé, et les permissions dont dispose cet utilisateur
  - Les fichiers ouverts, y compris les descripteurs de fichiers standard (voir plus loin)
  - Le terminal auquel le processus est attaché, s'il existe (voir plus loin)

En fait, un peu plus compliqué que ça : sous Linux, un processus ne peut en lancer un autre, il peut juste soit se dupliquer (fork, auquel cas toutes les ressources de ce processus sont héritées), soit disparaître pour être remplacé par un autre processus (exec, auquel cas les ressources sont libérées, sauf celles ci-dessus).

Signaux

Un processus peut recevoir un signal (un entier), susceptible d'influer sur son comportement :

Nom	Code	Signification
HUP	1	Le parent est mort
INT	2	Demande d'interrompre le processus (CTRL+c)
QUIT	3	Demande d'interrompre le processus (CTRL+\)
KILL	9	Tue le processus
TERM	15	Demande polie d'interrompre le processus
CONT	18	Demande de reprendre un processus arrêté
STOP	19	Arrête temporairement un processu
TSTP	19	Demande d'arrêter temporairement un proc. (CTRL+z)

Le comportement de la plupart des signaux peut être changé (p. ex., pour les ignorer, sauf KILL et STOP).

Code de retour

Un processus renvoie une fois terminé un code de retour (un entier) qui :

- vaut 0 quand tout s'est bien passé
- vaut une valeur différente quand une erreur s'est produite (l'entier précis peut indiquer le type d'erreur, dépend des commandes)

Plan

Système de fichiers racine

- L'ensemble de tous les fichiers est situé dans une hiérarchie arborescente, appelée système de fichiers racine
- Plusieurs systèmes de fichiers peuvent co-exister (par exemple, correspondant à différents disques), mais tous sont rattachés à ce système de fichiers racine (sous-arbres de l'arborescence)
- La racine de l'arborescence est désignée par /
- Un chemin absolu dans l'arborescence est désigné par la séquence des nœuds (fichiers) qui compose ce chemin, séparés par des /
- Par exemple : `/home/pierre/enseignement/linux/slides.tex`
- Par convention, un fichier dont le nom commence par un « . » est appelé ficher caché et non affiché par défaut par certains logiciels

Chemin relatif

Étant donné un chemin absolu x (par exemple, donné par la variable d'environnement PWD) :

- « . » désigne x
- « .. » désigne le chemin obtenu à partir de x en enlevant le dernier nœud de la séquence
- « a/b/c » désigne le chemin absolu « x/a/b/c »

Types de fichier

Tous les nœuds de la hiérarchie sont des fichiers, de 7 types différents :

`d`	Répertoire	nœuds internes de la hiérarchie
`-`	Fichier ordinaire	
`l`	Lien symbolique	pointe vers un chemin de la hiérarchie
`p`	Tube (pipe) nommé	communication uni-directionnelle
`s`	Connecteur (socket) Unix	communication bidirectionnelle
`b`	Périphérique bloc	disques, partitions, etc.
`c`	Périphérique caractère	périphériques d'entrée–sortie, etc.

- Les liens symboliques s'utilisent comme ce vers quoi ils pointent
- Les tubes nommés, les périphériques s'utilisent comme des fichiers standard (lecture, écriture) !
- Les connecteurs Unix s'utilisent comme des interfaces réseau (écoute, envoi de message, etc.)

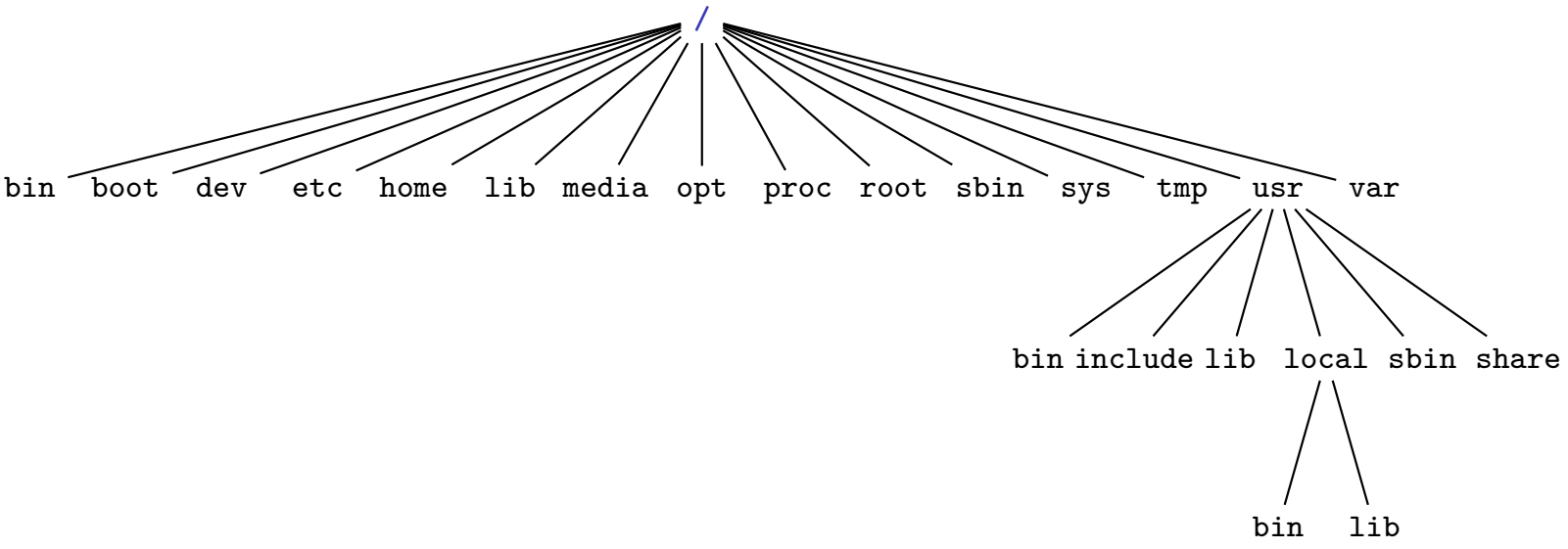
Propriétaire et permissions

- Chaque fichier :
  - appartient à un utilisateur u
  - appartient à un groupe g
  - a trois permissions possibles pour chacun des trois niveaux suivants : l'utilisateur u, les utilisateurs du groupe g, et tous les autres utilisateurs (o) :

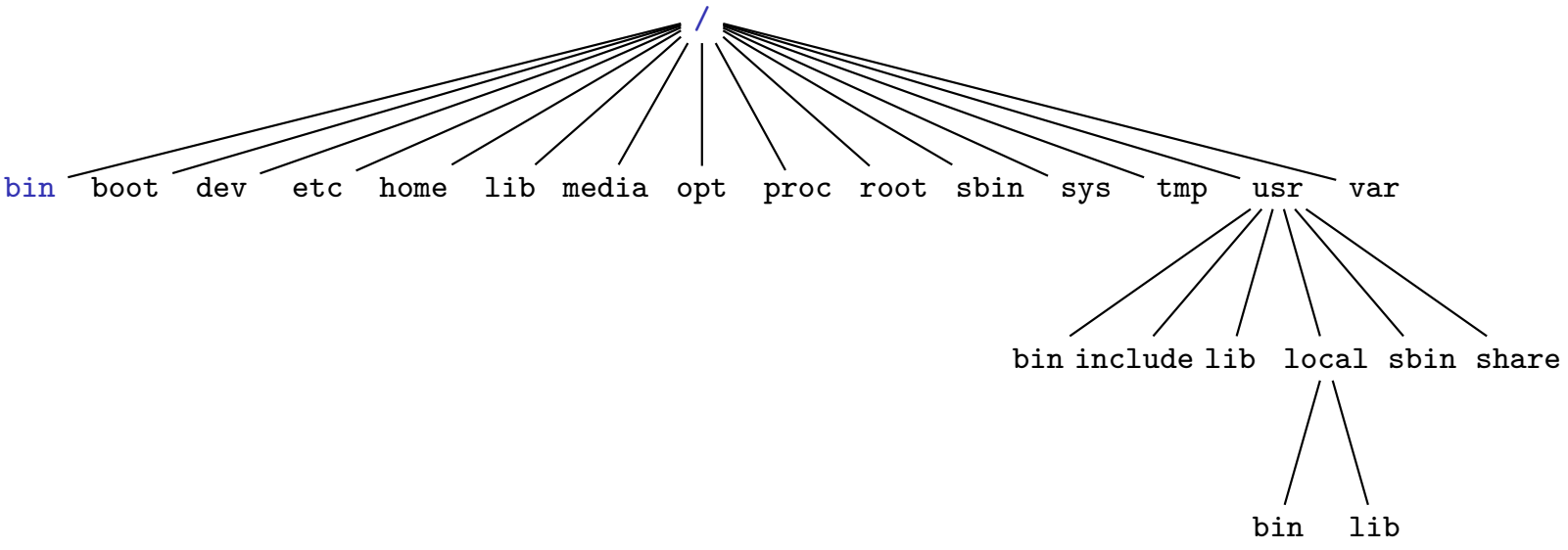
`r`	ce fichier peut-il être lu ?
`w`	ce fichier peut-il être modifié ?
`x`	ce fichier peut-il être exécuté ?

- Pour un répertoire, `r` veut dire lister les fichiers contenus, `w` ajouter ou supprimer un fichier contenu, `x` accéder aux fichiers contenus
- Le superutilisateur a toujours tous les droits.

Norme de hiérarchie du système de fichiers (FHS)

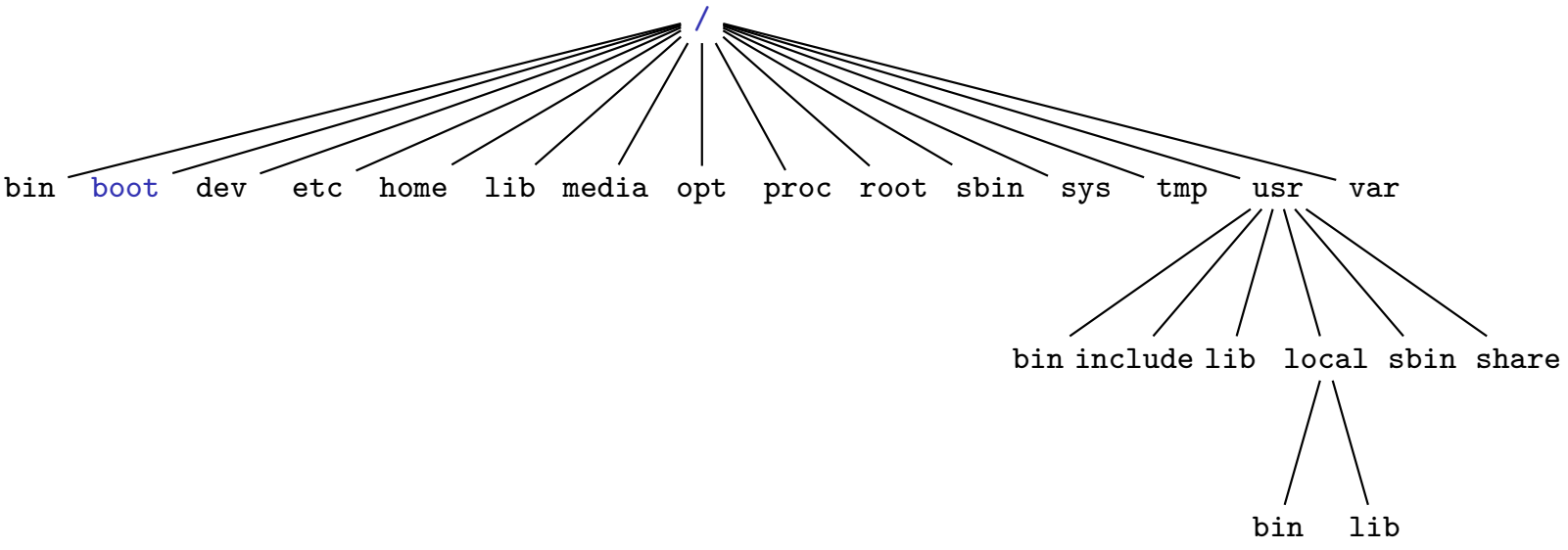


Norme de hiérarchie du système de fichiers (FHS)



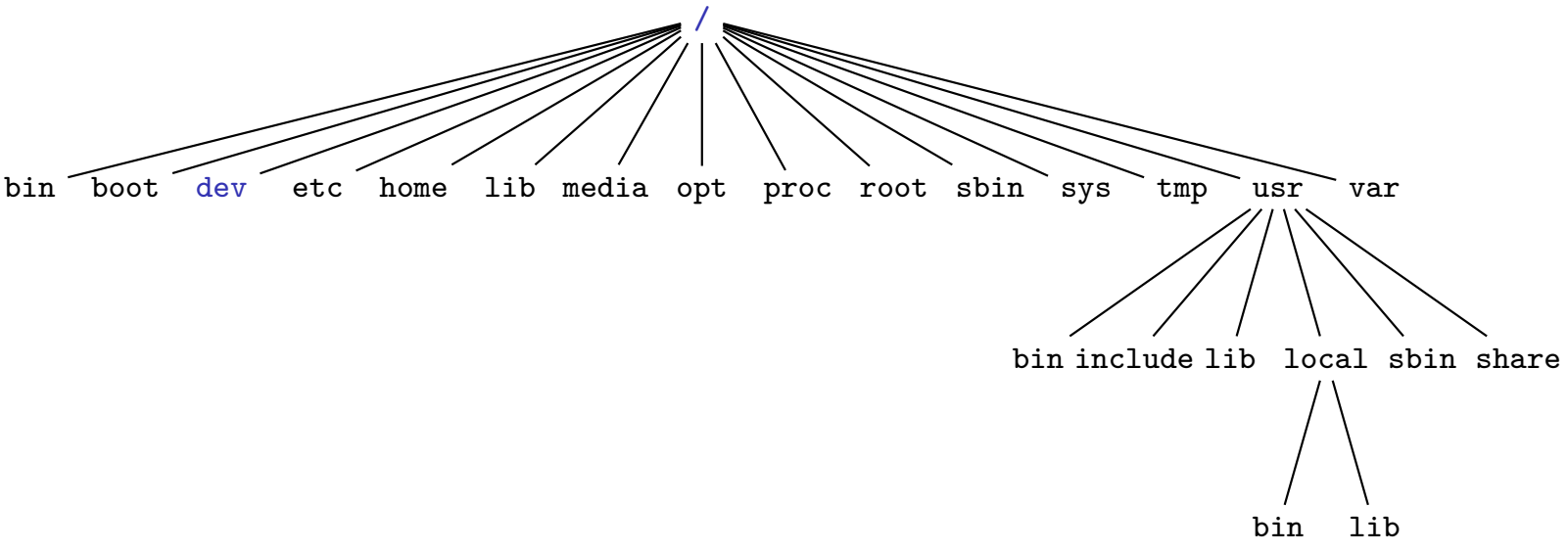
Exécutables de base

Norme de hiérarchie du système de fichiers (FHS)



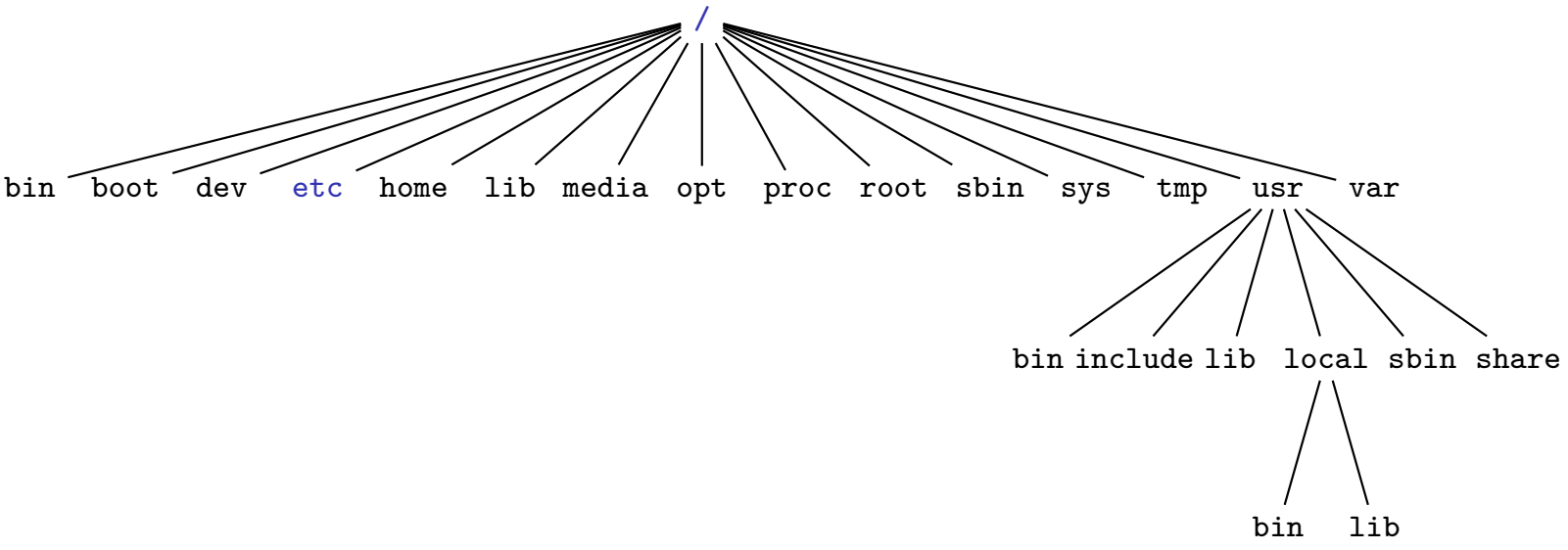
Images du noyau et de l'initial RAM disk

Norme de hiérarchie du système de fichiers (FHS)



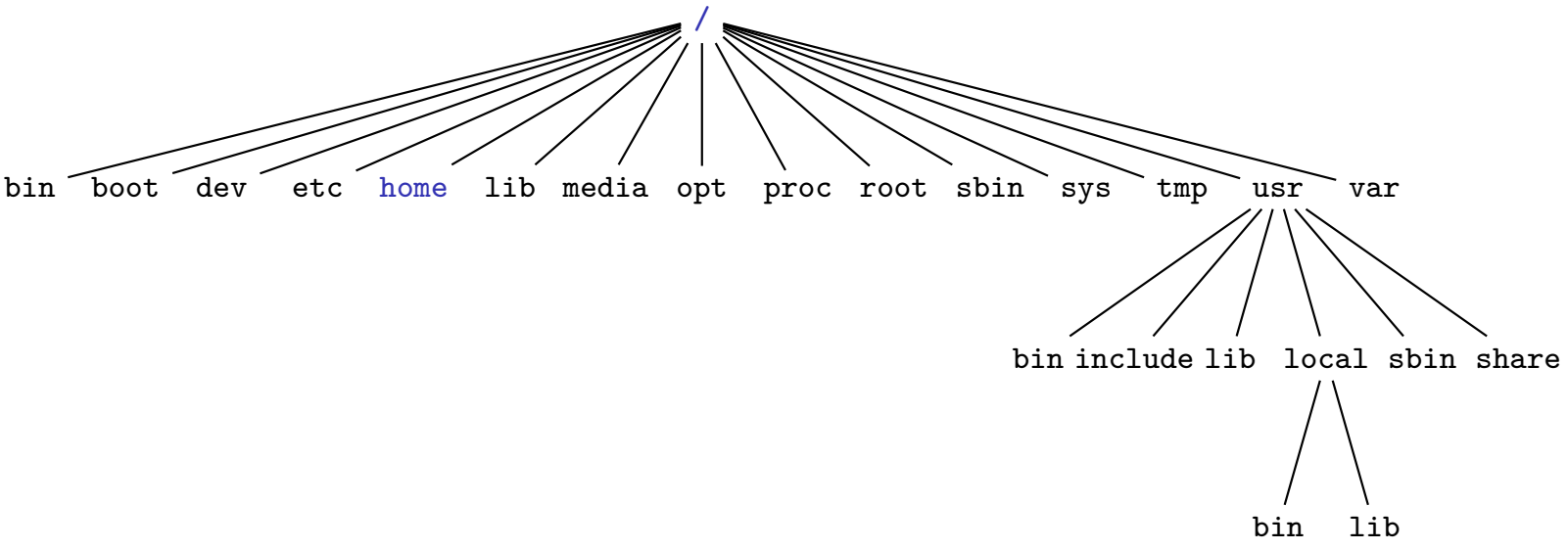
Périphériques

Norme de hiérarchie du système de fichiers (FHS)



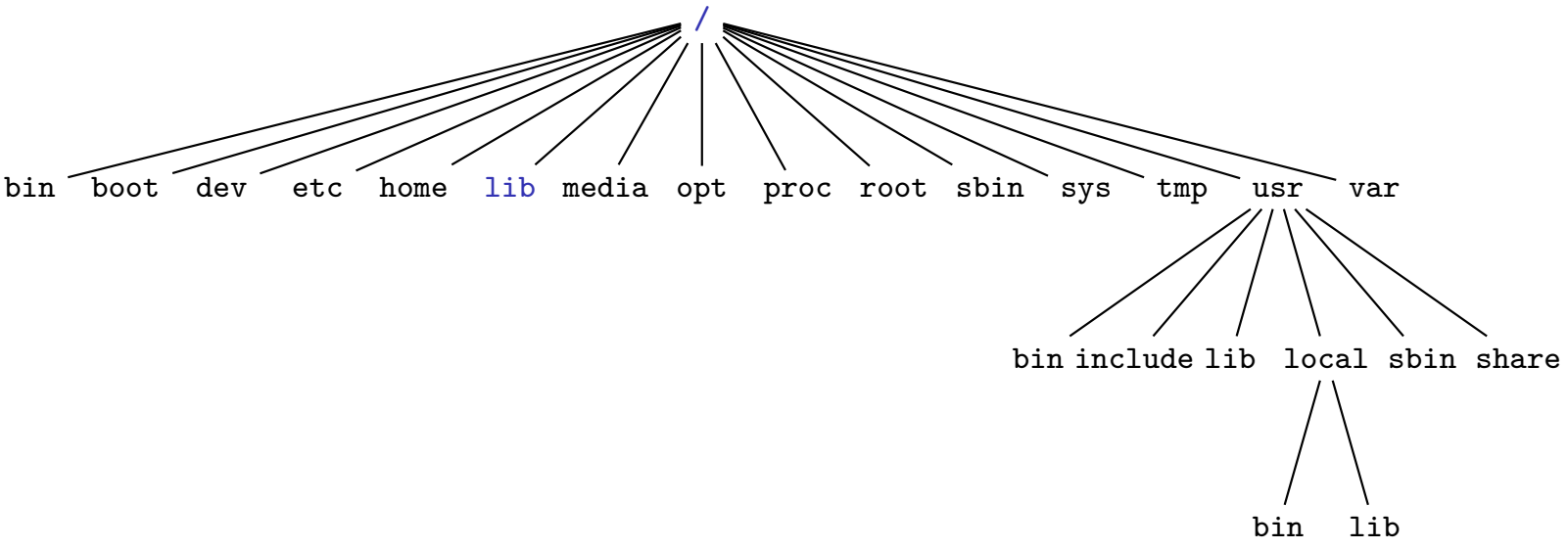
Fichiers de configuration

Norme de hiérarchie du système de fichiers (FHS)



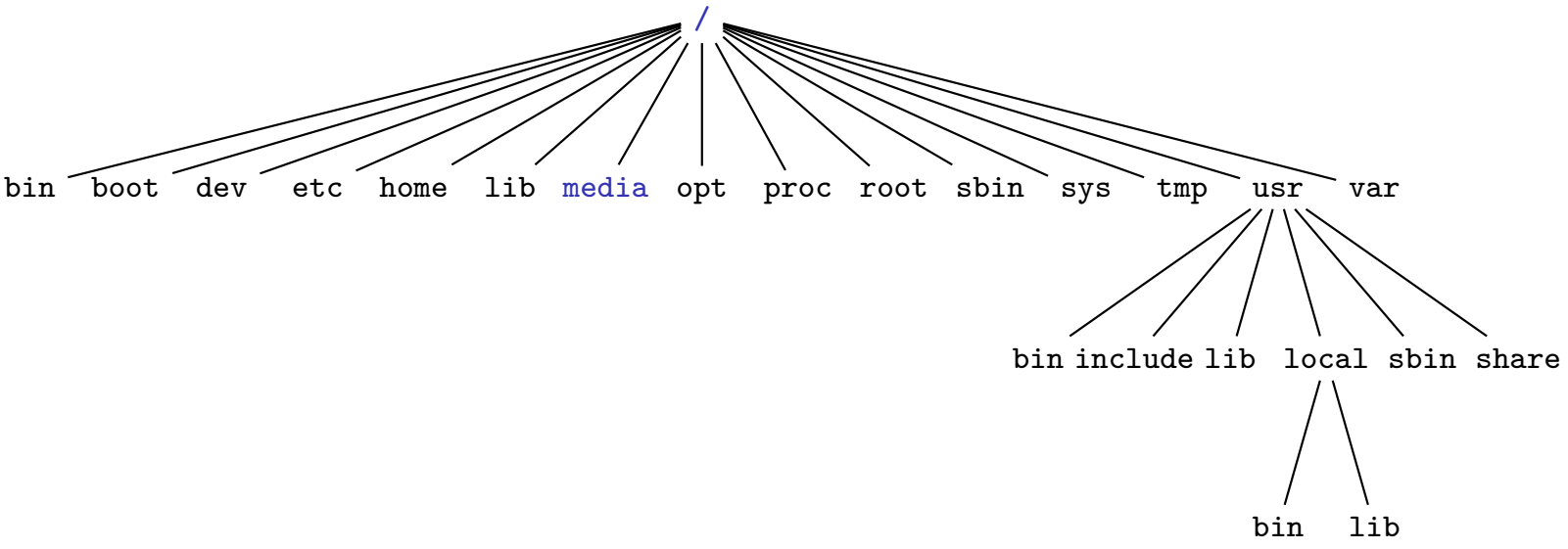
Répertoires des utilisateurs

Norme de hiérarchie du système de fichiers (FHS)



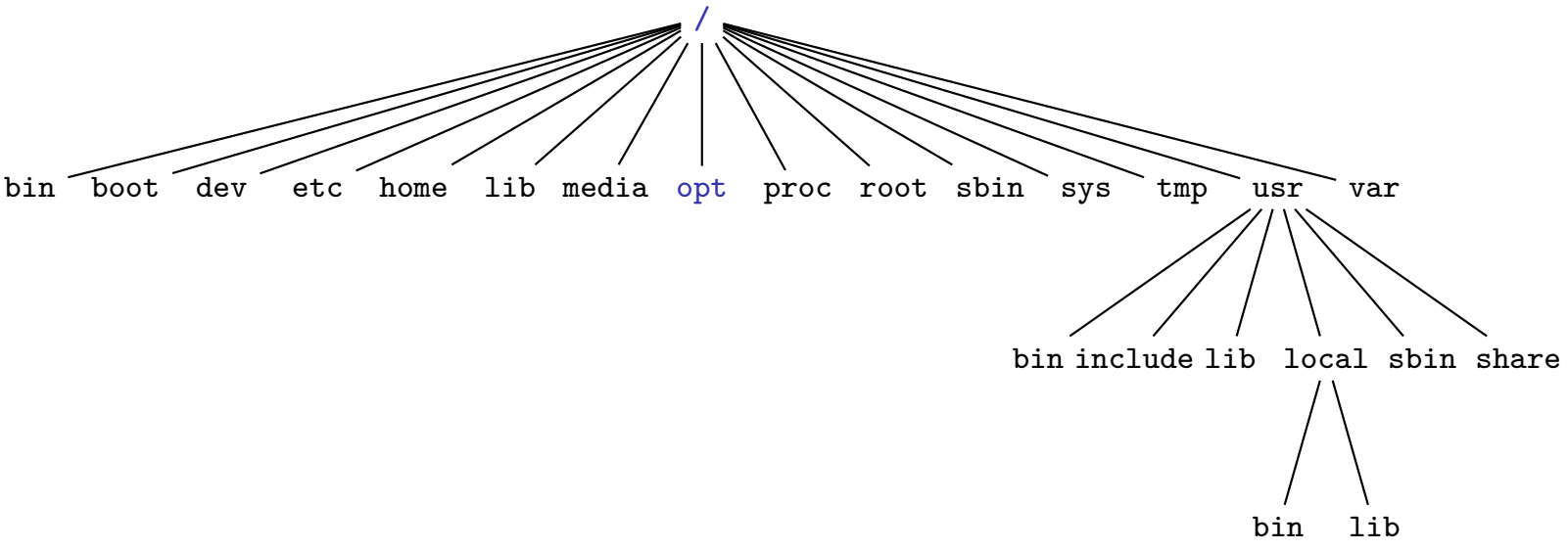
Bibliothèques de base

Norme de hiérarchie du système de fichiers (FHS)



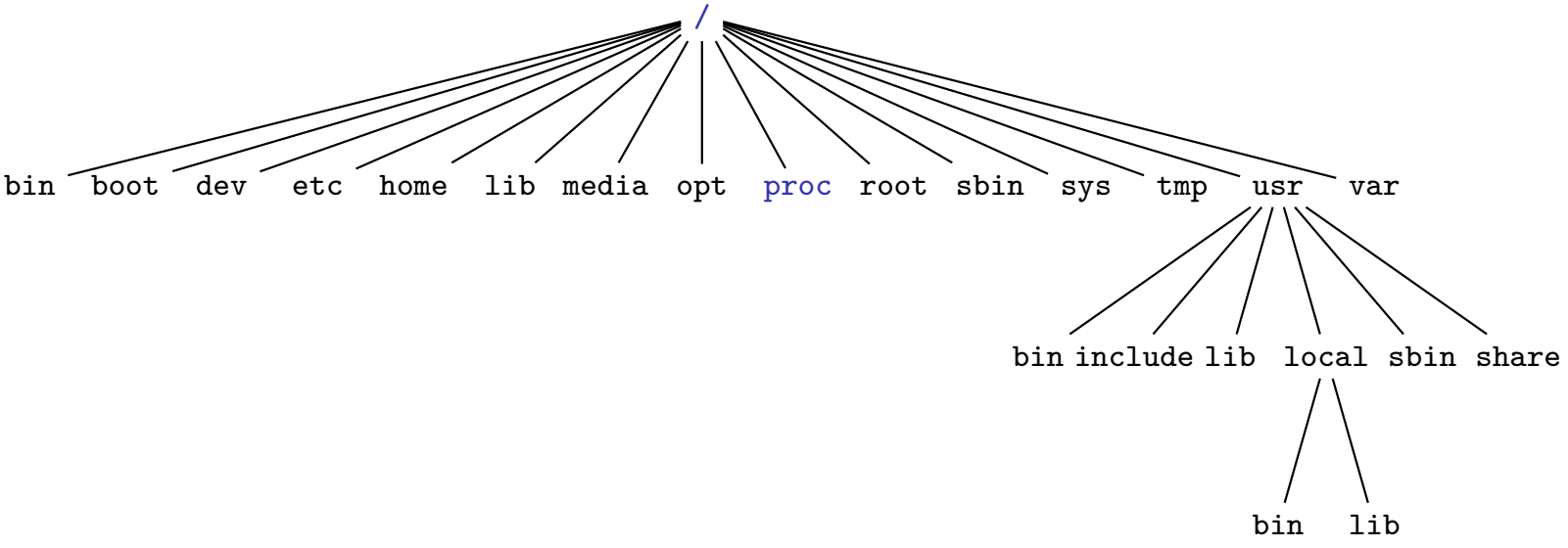
Point de montage pour disques occasionnels

Norme de hiérarchie du système de fichiers (FHS)



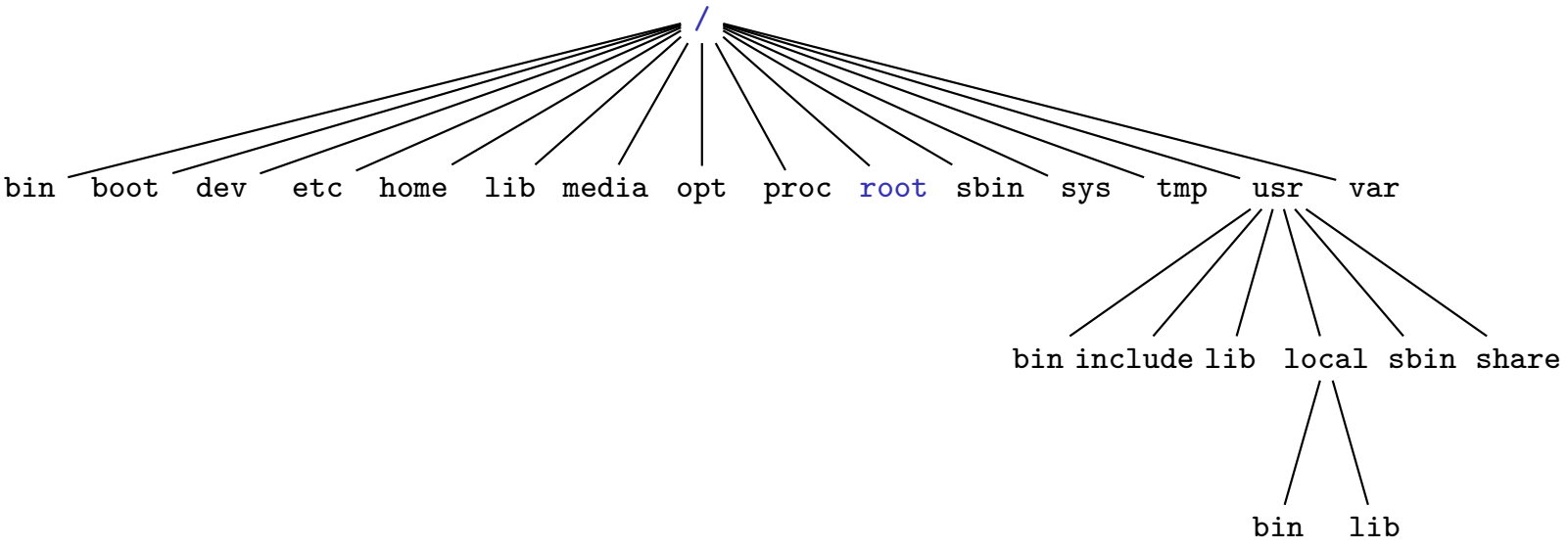
Logiciels non gérés par le système

Norme de hiérarchie du système de fichiers (FHS)



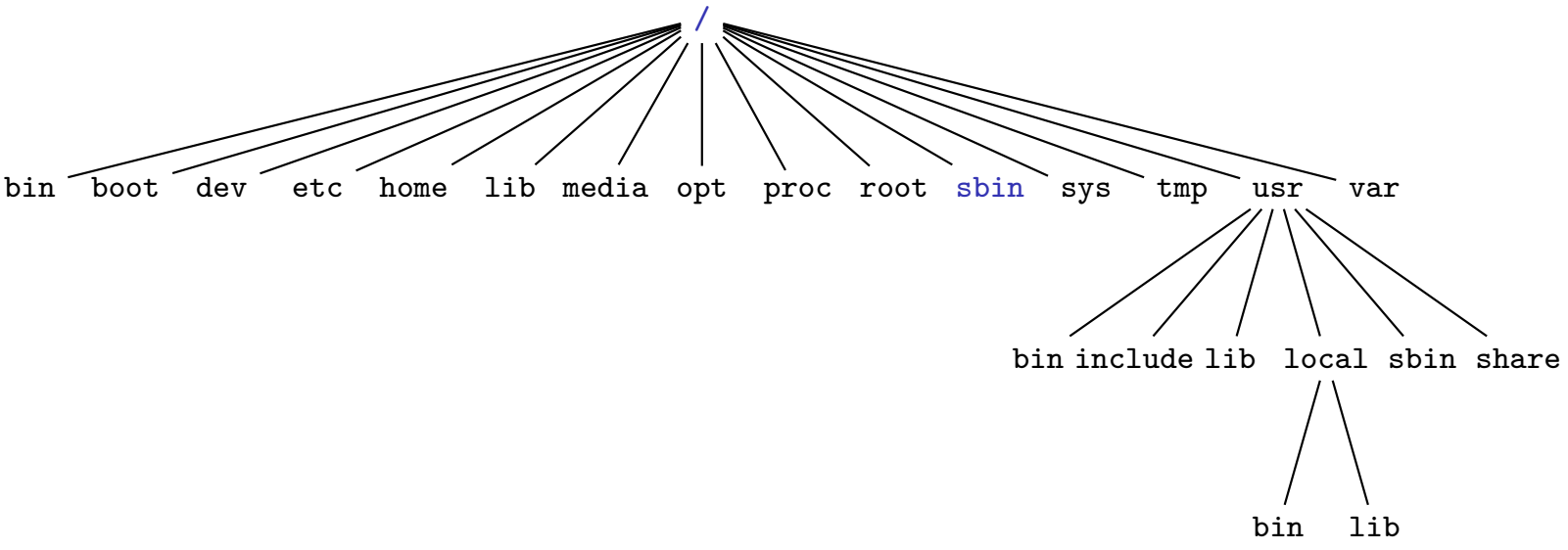
Informations sur les processus

Norme de hiérarchie du système de fichiers (FHS)



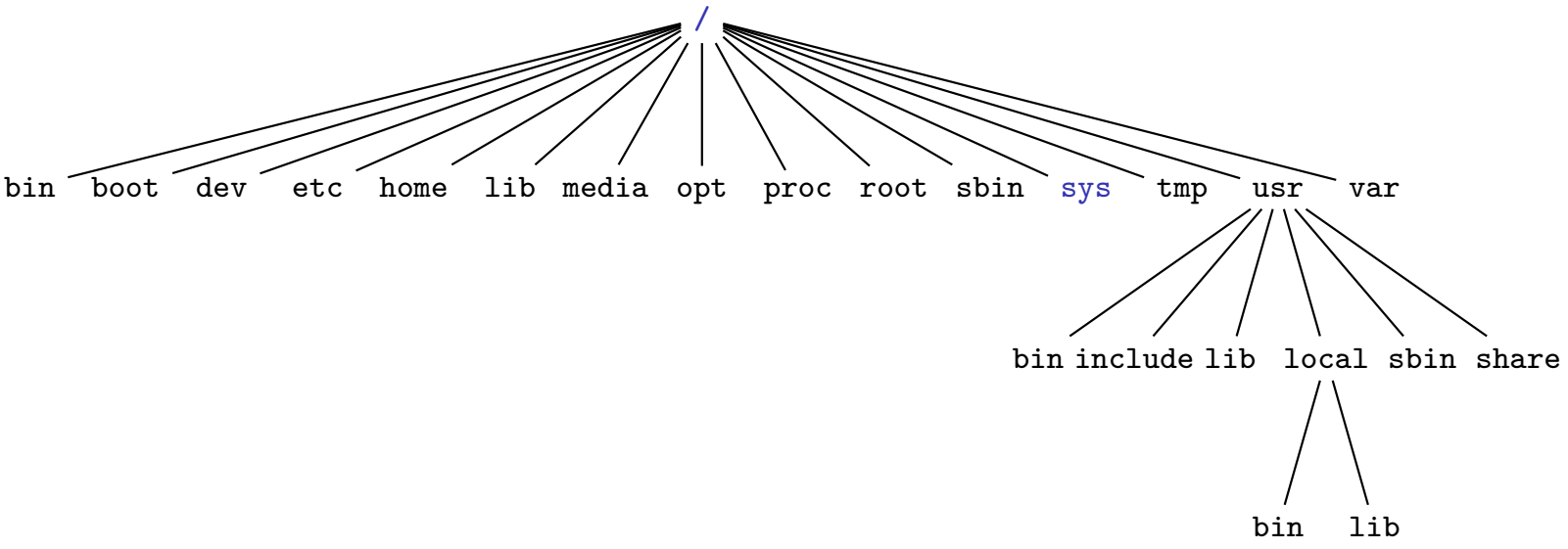
Répertoire principal du superutilisateur

Norme de hiérarchie du système de fichiers (FHS)



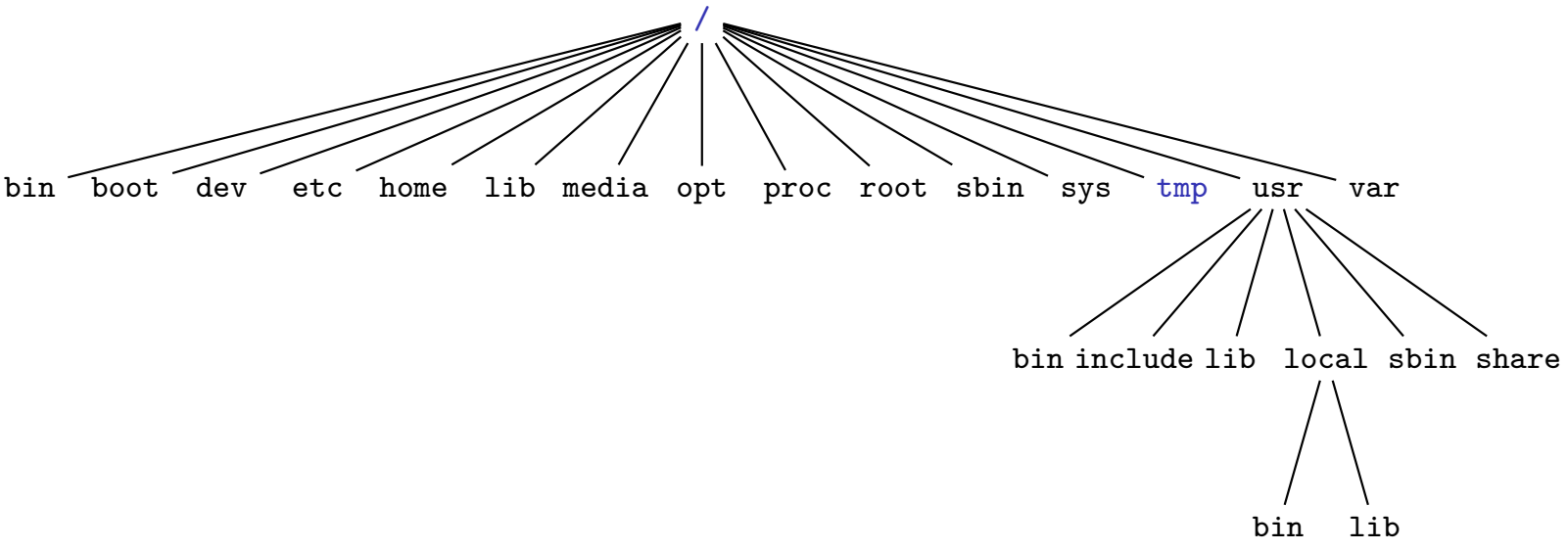
Exécutables systèmes de base

Norme de hiérarchie du système de fichiers (FHS)



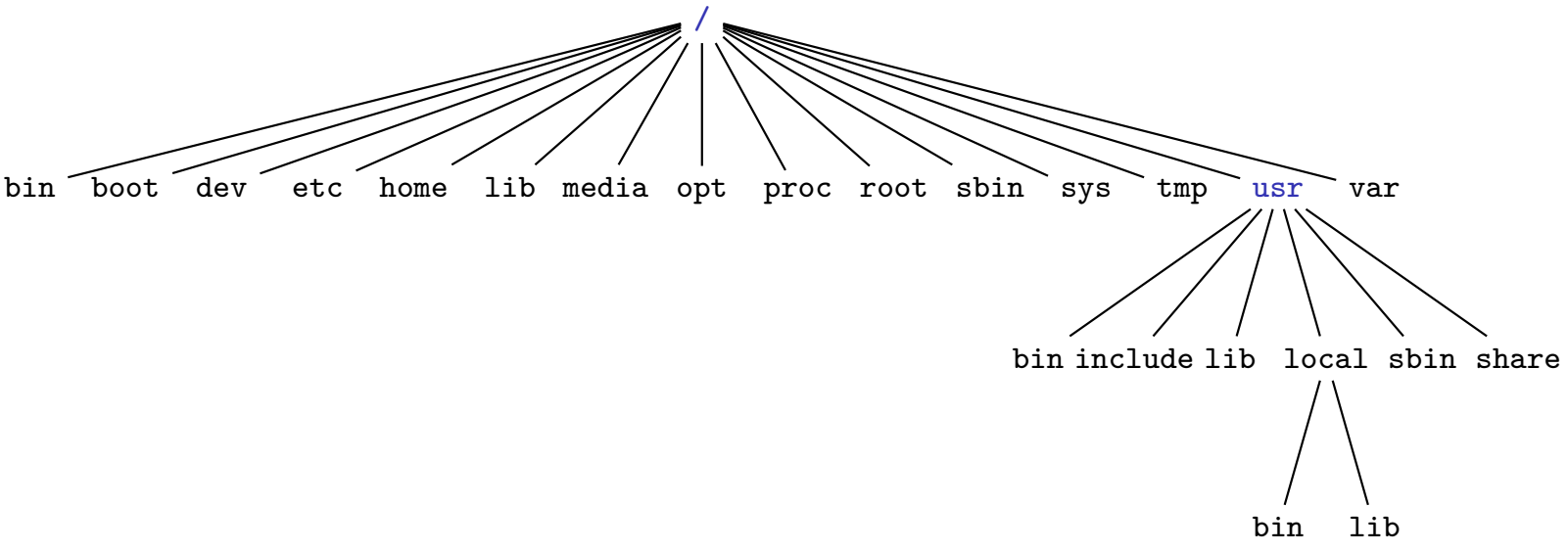
Informations sur le noyau et les périphériques

Norme de hiérarchie du système de fichiers (FHS)



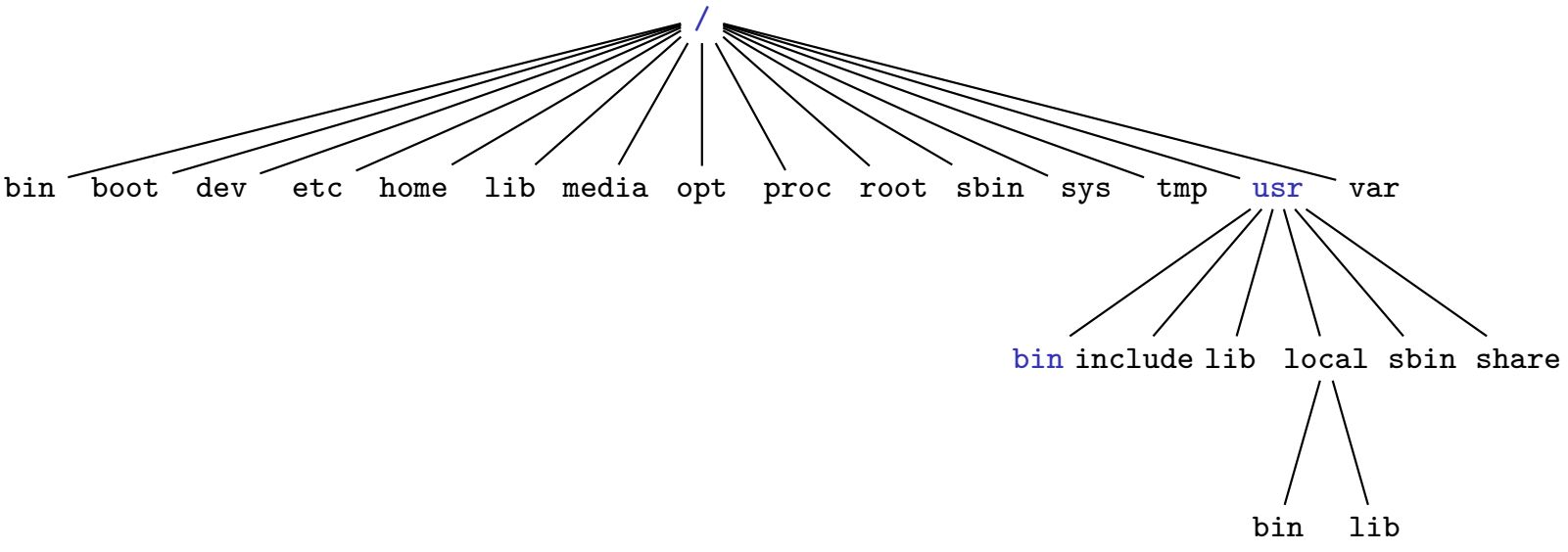
Fichiers temporaires

Norme de hiérarchie du système de fichiers (FHS)



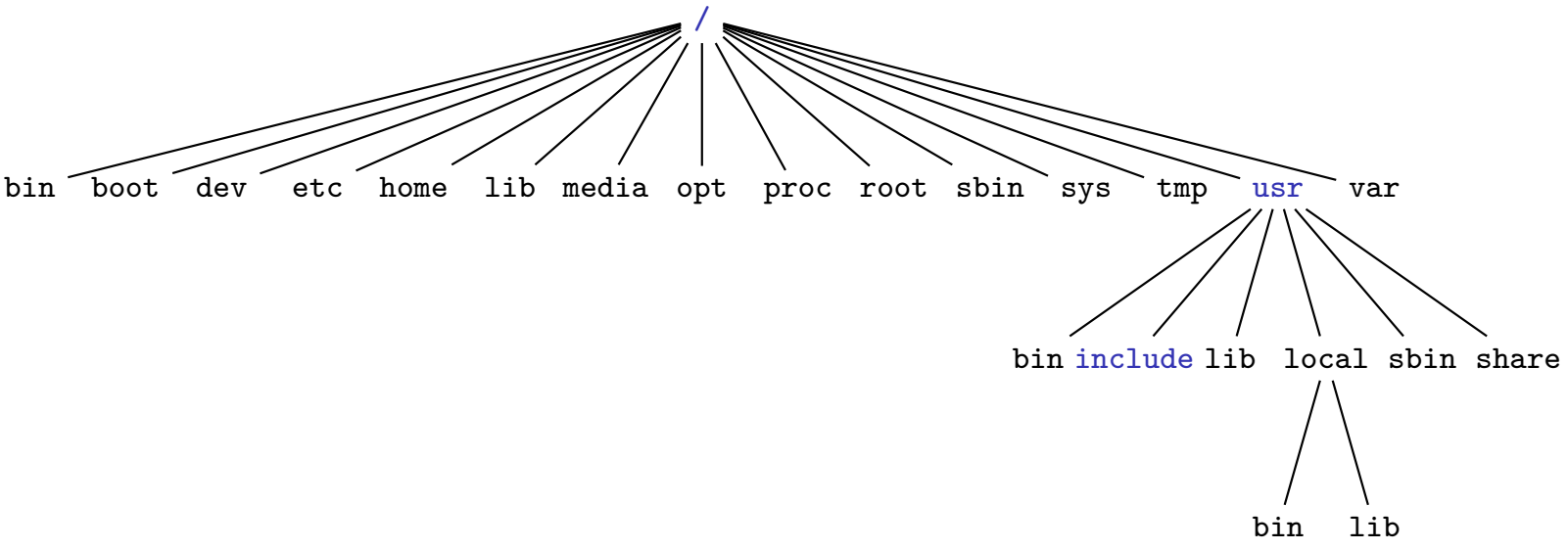
Hiérarchie de second niveau

Norme de hiérarchie du système de fichiers (FHS)



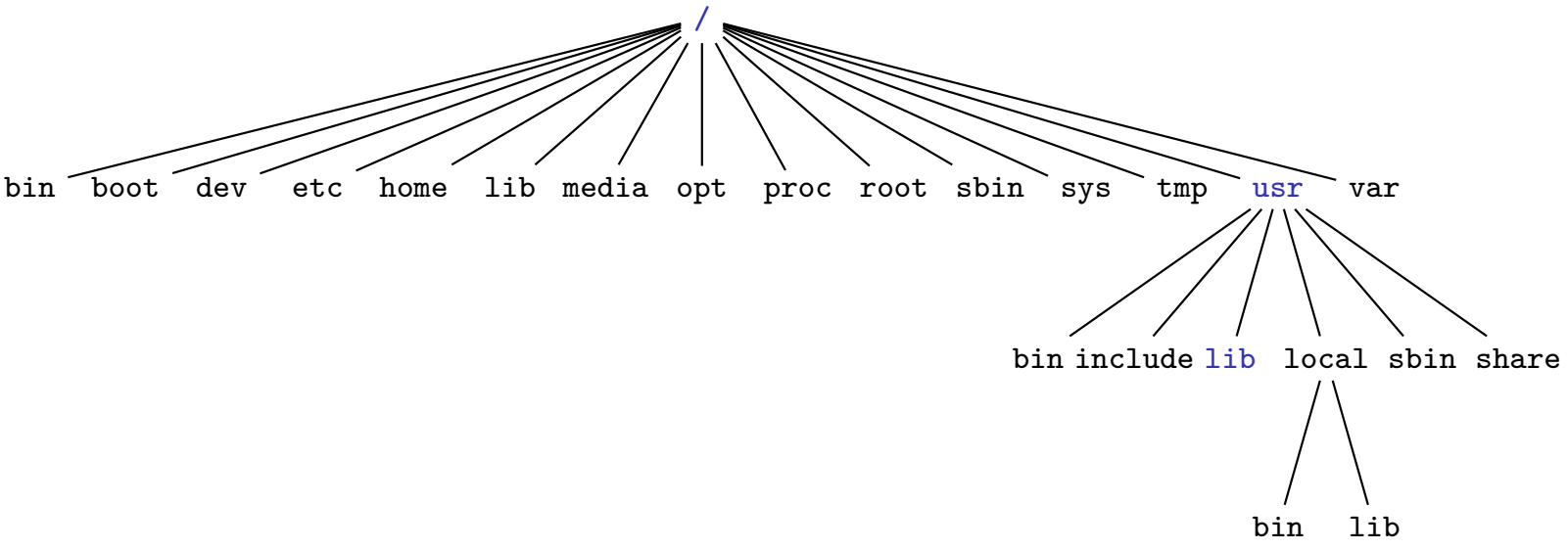
Exécutables standard

Norme de hiérarchie du système de fichiers (FHS)



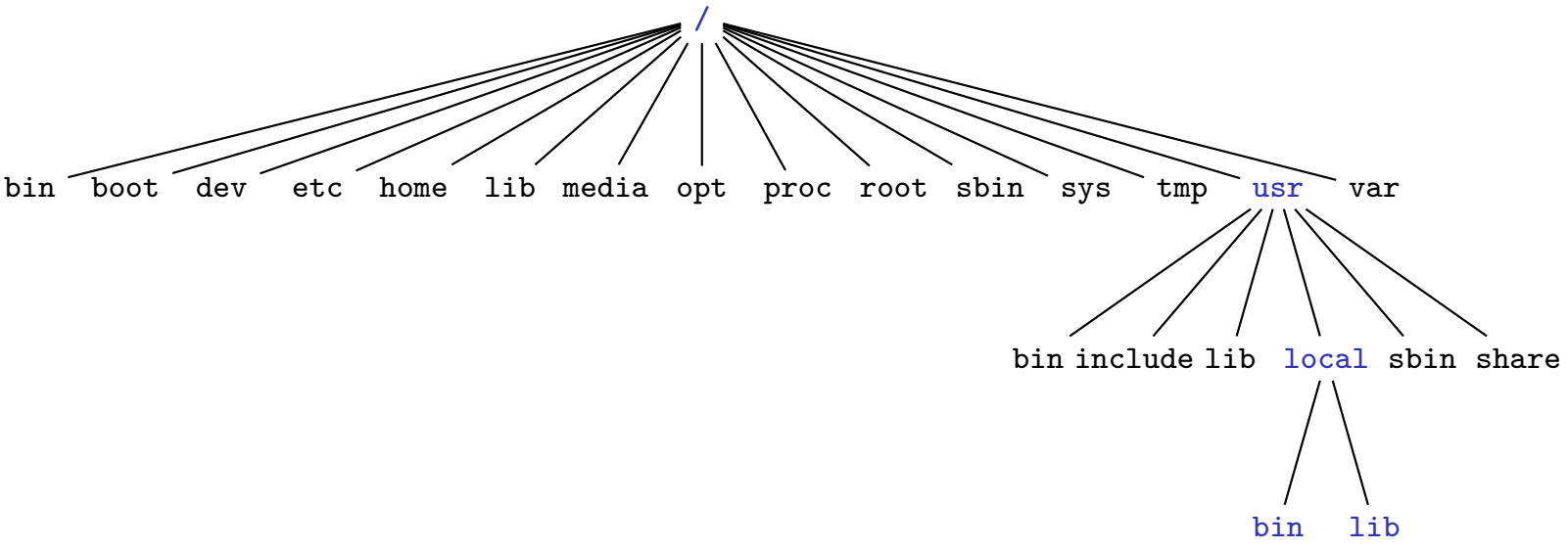
En-têtes de compilation

Norme de hiérarchie du système de fichiers (FHS)



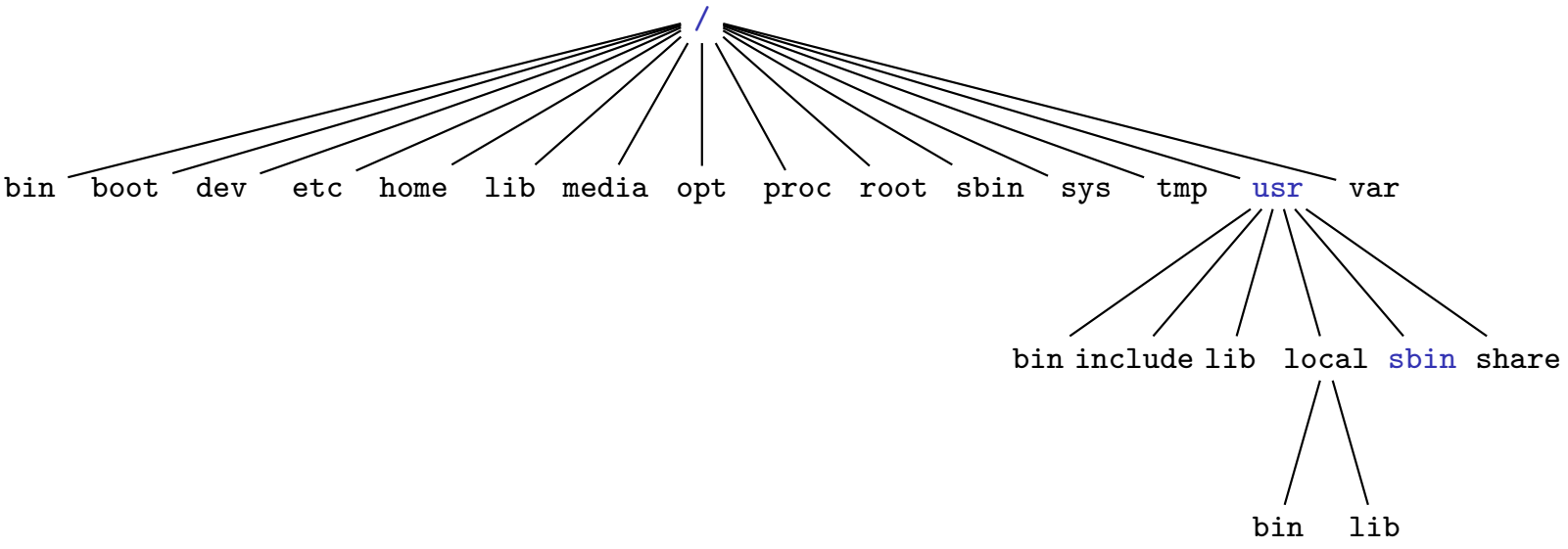
Bibliothèques standard du système

Norme de hiérarchie du système de fichiers (FHS)



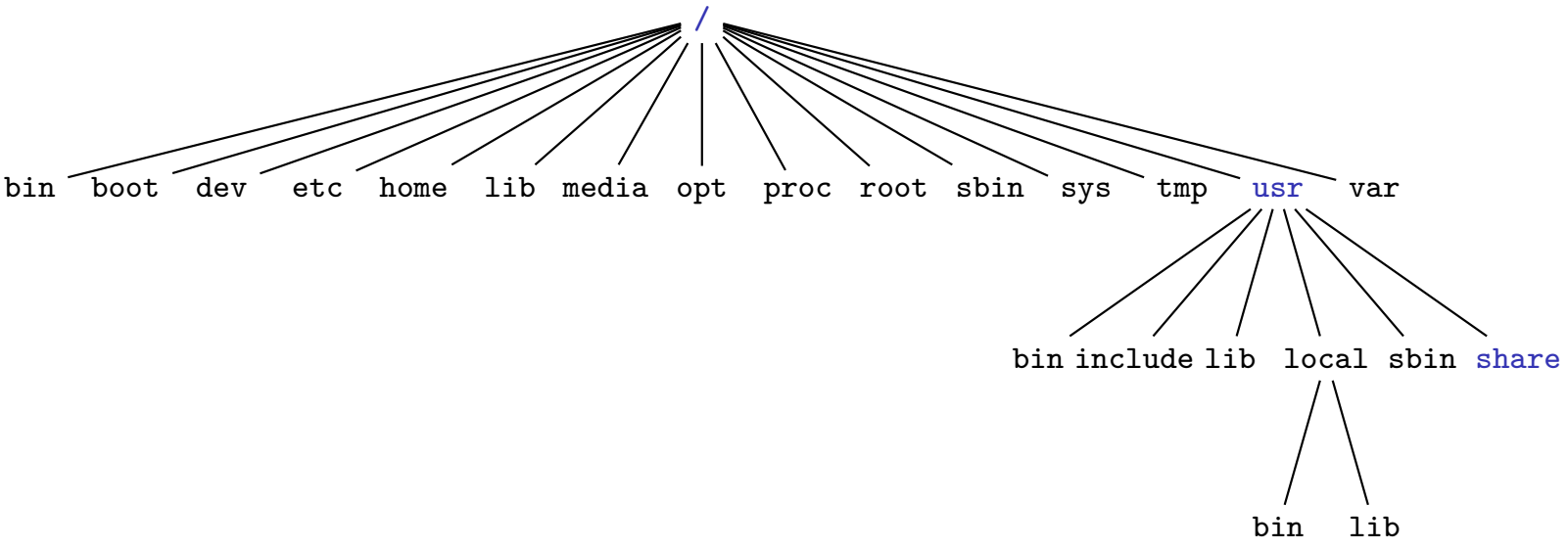
Hiérarchie de troisième niveau (non gérée par le système)

Norme de hiérarchie du système de fichiers (FHS)



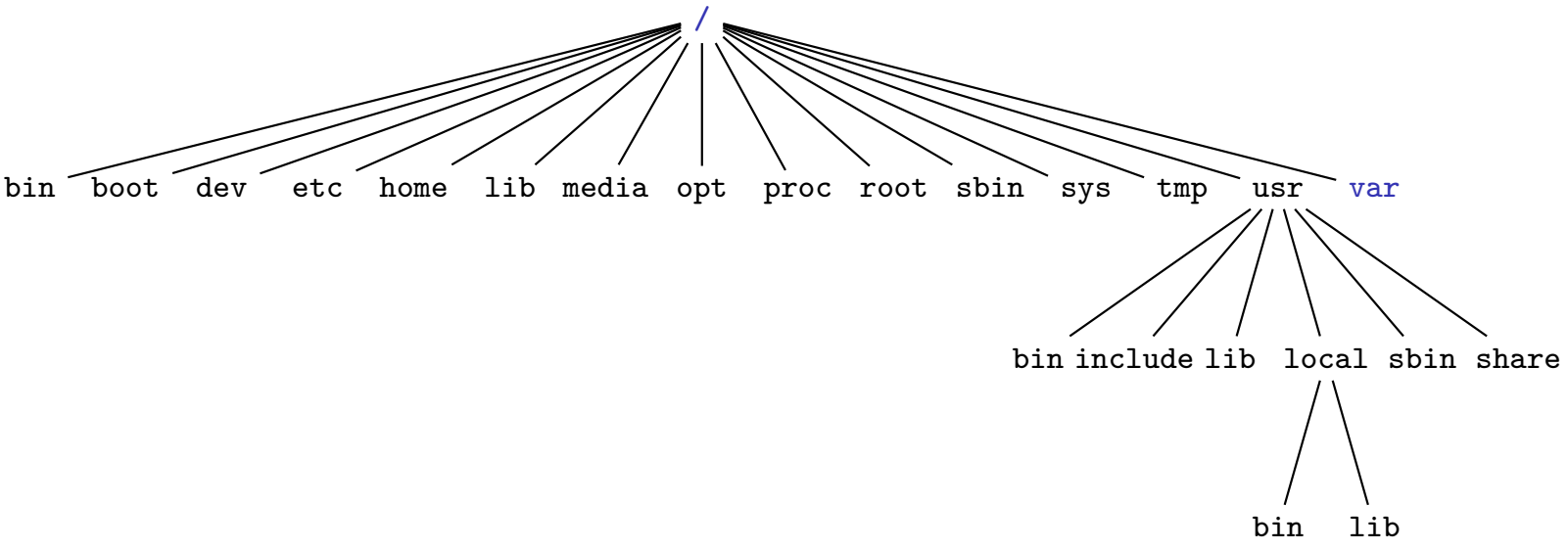
Exécutables systèmes standard

Norme de hiérarchie du système de fichiers (FHS)



Fichiers indépendants de l'architecture (documentation, etc.)

Norme de hiérarchie du système de fichiers (FHS)



Fichiers changeant fréquemment (logs, bases de données, etc.)

/dev

Contient des entrées pour les disques, les partitions, les périphériques d'entrée–sortie, etc., mais aussi :

`/dev/null` En écriture, fait disparaître ce qui y est écrit ; en lecture, se comporte comme un fichier vide.

`/dev/zero` En lecture, se comporte comme un fichier contenant une infinité d'octets de valeur 0.

`/dev/random` En lecture, se comporte comme un fichier contenant une infinité d'octets fournis par un générateur pseudo-aléatoire avec garantie cryptographique ; peut bloquer si le système manque d'entropie.

`/dev/urandom` En lecture, se comporte comme un fichier contenant une infinité d'octets fournis par un générateur pseudo-aléatoire sans garantie cryptographique.

Fichiers importants dans /etc

`/etc/passwd` Utilisateurs locaux avec leur login, UID, groupe principal, nom complet, répertoire principal, shell de login

`/etc/shadow` Hachage cryptographique des mots de passe des utilisateurs locaux

`/etc/group` Définition des groupes et des utilisateurs qui y sont rattachés secondairement

`/etc` contient aussi les fichiers de configuration de tous les logiciels installés sur le système.

Le système de fichiers `/sys`

- Comme `/proc`, système de fichiers virtuel contenant une hiérarchie de répertoires, fichiers, liens symboliques virtuels contenant de l'information sur le système
- Informations sur les périphériques et le noyau, présentée de manière structurée permettant à des programmes d'y accéder
- Possible de changer certains paramètres du noyau en écrivant dans un fichier de `/sys`

Le système de fichiers /proc

Étant donné le PID *p* d'un processus :

`/proc/p/cmdline` contient la ligne de commande

`/proc/p/environ` contient l'environnement du processus

`/proc/p/exe` est un lien vers l'exécutable

`/proc/p/fd` est un répertoire listant tous les fichiers ouverts (liens symboliques vers les vrais fichiers)

`/proc/p/fdinfo/i` contient de l'information sur le fichier ouvert numéro *i*

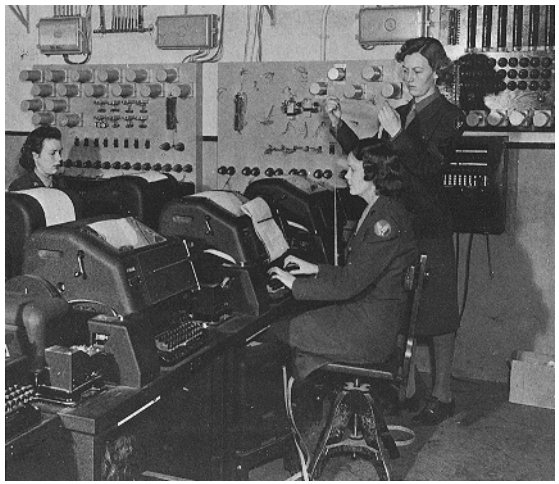
`/proc/p/maps` est un descriptif de l'espace mémoire

Pour des raisons historiques, `/proc` contient aussi d'autres informations, non liées aux processus, mais la plupart se retrouvent aussi, mieux présentée, dans `/sys`.

Plan

Terminal

- Historiquement, un téléscripteur, télétype, terminal est un appareil (avec un clavier et, suivant les époques, une imprimante ou un écran) connecté à un ordinateur



CC BY 2.0 Jason Scott

- Actuellement, un émulateur de terminal (ou terminal) est une émulation logicielle moderne (par Linux lui-même pour les consoles accessibles avec CTRL+ALT+Fx, ou plus courramment au sein d'un logiciel graphique tel que `xterm`, `gnome-terminal`, `konsole`) d'un tel terminal

Pseudo-terminal

- Un émulateur de terminal utilise un pseudo-terminal, une paire de périphériques de type caractère (`/dev/ptmx` et `/dev/pts/`*i*), qui sert à communiquer entre l'émulateur de terminal (qui reçoit les entrées et affiche les sorties) et les processus rattachés à ce terminal
- Entrée : clavier
- Sortie : affichage en mode texte (éventuellement avec couleurs)
- Un (ou plusieurs) processus est rattaché à ce terminal ; il peut lire l'entrée sur son entrée standard, et produire des affichages sur le terminal en écrivant sur ses sorties standard et d'erreur
- Au lancement du terminal, le shell de login de l'utilisateur est lancé, seul processus rattaché à ce terminal
- Le terminal envoie des signaux au(x) processus de premier plan rattaché à ce terminal (`INT` pour CTRL+c, `QUIT` pour CTRL+\, `TSTP` pour CTRL+z)

Historique du terminal

- Un terminal affiche un certain nombre de lignes
- Les émulateurs de terminaux modernes maintiennent un historique des lignes précédentes
- Accès par la molette de la souris, un ascenseur, ou SHIFT+PgUp et SHIFT+PgDn

Mettre le terminal en pause

- CTRL+s : pause l'affichage du terminal (utilise quand les lignes défilent vite)
- CTRL+q : reprend l'affichage du terminal

Plan

Shell

- Logiciel fournissant une interface type ligne de commande au sein d'un terminal
- Différents shells existent :

  `bash` shell le plus courant, par défaut sur de nombreuses distributions Linux, version moderne du Bourne shell développé pour Unix en 1979

  `zsh` alternative à bash, beaucoup de similarités, mais aussi beaucoup de fonctionnalités supplémentaire ; mon préféré !

  `tcsh` un shell assez différent, hérité du shell `csh` de BSD

  `ksh` des similarités à la fois avec `bash` et `tcsh`

- Focus sur `bash`, mais prenez le temps d'essayer d'autres shells, vous allez passer beaucoup de temps à interagir avec un shell, autant que ce soit le bon !

Lancer un exécutable

- Fonction première d'un shell : lancer des exécutables (ou commandes) avec leurs arguments (séparés d'espaces)
- Si le répertoire contenant l'exécutable est dans la variable d'environnement `PATH` (plusieurs chemins séparés par des « : »), il suffit d'indiquer son nom (et les arguments éventuels)
- Sinon, indiquer le chemin absolu ou relatif (commençant obligatoirement par « `./` » ou « `../` ») de l'exécutable
- Le processus peut être lancé :
  - au premier plan, par défaut ; on attend qu'il termine pour revenir au shell
  - à l'arrière-plan, en terminant la ligne de commande par un caractère « & » ; on revient au shell, mais le processus hérite toujours des entrées/sorties standard, liées au terminal, et sera tué quand son père (le shell) mourra
  - en le déconnectant du terminal et en l'empêchant d'être tué par le shell à sa mort, en préfixant la ligne de commande par « `nohup` » et en la terminant par « & »

Complétion automatique

- En appuyant sur TAB, le shell complète automatiquement en fonction du contexte :
  - en début de ligne de commande, les noms d'exécutables dans le `PATH`
  - les noms de fichiers
  - les chemins, absolus et fichiers
  - (parfois) complétion intelligente des arguments d'une commande en fonction de la commande lancée : options, noms de machine plutôt que noms de fichiers, etc. (en bash, à activer avec `. /etc/bash_completion`)

Édition de la ligne de commande

Nombreux raccourcis claviers (hérités de l'éditeur de texte Emacs, et utilisés dans beaucoup d'outils en ligne de commande) :

$\leftarrow$, $\rightarrow$	déplacer le curseur
CTRL+a	aller en début de ligne
CTRL+e	aller en fin de ligne
ALT+$\rightarrow$	aller au mot suivant
ALT+$\leftarrow$	aller au mot précédent
CTRL+w	effacer le mot précédent
CTRL+u	effacer tout ce qui précède le curseur
CTRL+k	effacer tout ce qui suit le curseur
CTRL+_	annuler la dernière édition (peut être répété)
CTRL+l	effacer l'écran

Historique des commandes

Le shell maintient un historique des commandes précédemment tapées (dans cette session, ou dans des sessions précédentes)

- CTRL+r suivi d'une chaîne de caractère recherche cette chaîne dans l'historique (CTRL+r à nouveau pour l'occurrence suivante)
- `!!` est remplacé à l'exécution de la commande par la ligne de commande précédente
- `!*` est remplacé à l'exécution de la commande par les arguments de la commande précédente
- `history` affiche l'historique des commandes, et `!`*i* est remplacé par la ligne de commande *i*

Appeler plusieurs commandes

- On peut séparer plusieurs commandes par un « ; ». Elles seront appelées les unes à la suite des autres.
- On peut aussi utiliser des opérateurs logiques :

  `a && b` lance *a* ; une fois *a* terminé, ne lance *b* que si *a* a renvoyé un code de retour 0

  `a || b` lance *a* ; une fois *a* terminé, ne lance *b* que si *a* a renvoyé un code de retour différent de 0
- On peut utiliser des parenthèses pour chaîner des commandes de manière complexe :

  `(a || (b; c)) && d` exécute *a*, puis (*b* puis *c*) si *a* échoue puis *d* si (soit *a* soit *c*) a retourné un code de retour 0

Redirections

On peut lancer une commande en pré-ouvrant des fichiers en entrée ou en sortie, à la fin de la ligne de commande :

i< *fichier* ouvre le fichier *fichier* en entrée, avec comme descripteur de fichier *i*

i> *fichier* ouvre le fichier *fichier* en sortie, écrasant son contenu, avec comme descripteur de fichier *i*

i>> *fichier* ouvre le fichier *fichier* en sortie, ajoutant du contenu à la fin du fichier, avec comme descripteur de fichier *i*

En particulier, < ou 0< fournit un fichier en entrée standard ; > ou 1> écrit la sortie standard dans un fichier ; 2> écrit la sortie d'erreur dans un fichier.
On peut aussi écrire 2>&1 pour rediriger la sortie d'erreur vers la sortie standard.

Tubes

- Si *a* et *b* sont deux commandes, alors *a* | *b* :
  - lance simultanément les commandes *a* et *b*
  - redirige la sortie standard de *a* vers un tube anonyme *x*
  - redirige l'entrée standard de *b* depuis *x*
- Cela a pour effet de fournir à *b* (sur son entrée standard) la sortie standard de *a*, au fur et à mesure qu'elle est produite
- Bloquant (avec tampon) : si *b* arrête de lire son entrée, *a* arrête de produire sa sortie (et vice-versa)
- Point central de la philosophie de la ligne de commande Linux, extrêmement pratique
- Peut être chaîné : *a* | *b* | *c* | *d* | ...

Jokers

On peut utiliser, dans des chemins ou noms de fichiers :

- `*` pour n'importe quel sous-chaîne d'un nom de fichier
- `?` pour n'importe quel caractère
- `[abc]` pour n'importe quel caractère parmi `a`, `b`, `c`

Le shell remplace ces jokers par l'ensemble des correspondances trouvées dans le répertoire en question.

Mais aussi...

- Substitution de processus : *a* `<(`*b*`)` lance la commande *b*, et fournit à la commande *a* un nom de fichier (un tube anonyme ou un tube nommé temporaire) qui, si lu, contient la sortie de la commande *b*
- Substitution de commande : *a* `` `b` `` ou *a* `$(`*b*`)` lance *b*, attend qu'elle termine, puis lance la commande *a* en lui fournissant en argument la sortie de *b*
- Arithmétique : `$((`*e*`))` évalue *e* comme une expression arithmétique
- Les variables d'environnement deviennent des variables du shell avec le même nom. La valeur d'une variable `TOTO` est obtenue avec `$TOTO`
- Dans un chemin, « `~` » est remplacé par le répertoire personnel de l'utilisateur

Protection des caractères spéciaux

- On peut entourer des chaînes de caractère par des « '...' » ou des « "..." » pour éviter l'interprétation de certains caractères spéciaux (espaces, &, <, etc.)
- À l'intérieur de « '...' », aucun caractère spécial n'est interprété
- À l'intérieur de « "..." », les caractères spéciaux « $ », « ` », « ! » et « \ » restent interprétés

Fichier de configuration

Un utilisateur peut paramétrer le shell `bash` (activer la complétion intelligente, initialiser des variables d'environnement, etc.) grâce à deux fichiers de configuration, dont le contenu est exécuté au démarrage du shell :

`$HOME/.bash_profile` pour le shell obtenu à la connexion d'un utilisateur (par exemple, via le gestionnaire de connexion)

`$HOME/.bashrc` pour tous les shells interactifs

Plan

Script shell

- Le shell est en fait un langage de programmation Turing-complet, cf. `https://pierre.senellart.com/other/languages/languages.xml`
- Utilisable en one-liner : mini-script qui tient sur une ligne de commande (potentiellement longue) !
- On peut aussi mettre des scripts shell dans des fichiers (souvent avec extension `.sh`), pour les exécuter
- Pratique pour certains traitements, mais ne pas en abuser ! Langage de programmation peu agréable, préférer un meilleur langage de script tel Python ou Perl
- On récupère l'ensemble des arguments du script avec `"$@"`, ou chacun des arguments avec `$1`, `$2`, etc.

Affectations, tests

```
annee=2022

if [[ $((annee % 400)) -eq 0; || \
  ( $((annee % 4)) -eq 0 && $((annee % 100)) -ne 0 ) ]]
then
  echo bissextile
else
  echo non-bissextile
fi
```

Il existe les commandes `true` et `false` qui renvoient toujours un code de retour nul et non nul, respectivement

Boucles for

```
for i in $(seq 1 10)
do
  for j in $(seq 1 10)
  do
    printf "%3d " $((i*j))
  done
  echo
done
```

Boucles while

```
j=0
while read i
do
  echo "Word $j: $i"
  j=$((j+1))
done < /usr/share/dict/words
```

Fonctions

```
somme() {
  $(($1+$2))
}

echo somme 17 25
```

- Les appels de fonctions ressemblent à des appels de commande
- Les arguments de fonction sont accessibles via $1, $2, etc.

Plan

`ls`

- Affiche le contenu du répertoire courant, ou d'un répertoire passé en argument
- -l affiche des détails sur chacun des fichiers (type, permissions, propriétaire, taille, date de dernière modification)
- -a pour aussi afficher les fichiers caché
- --color pour distinguer les types de fichier avec des couleurs

Répertoire courant : `pwd` et `cd`

- `pwd` affiche le répertoire courant
- `cd` change le répertoire courant en le répertoire passé en argument
- `cd`, sans argument, change le répertoire courant en le répertoire personnel de l'utilisateur
- `cd -` revient au répertoire précédent

Manipulation de fichiers : `cp`, `mv`, `rm`, `rmdir`

- `cp` *f* *g* copie le fichier *f* sous le nom *g* ou dans le répertoire *g* (s'il existe)
- `cp -r` copie un répertoire et son contenu
- `mv` déplace un fichier ou un répertoire
- `rm` supprime un fichier
- `rm -r` supprime un répertoire et son contenu ; dangereux !
- `rmdir` supprime un répertoire vide

Création de fichiers : `touch`, `mkdir`, `mkfifo`

- `touch` crée un fichier ordinaire ou met à jour sa date de modification
- `mkdir` crée un répertoire vide
- `mkfifo` crée un tube nommé

chmod

- Change les permissions d'un fichier
- `chmod` *x*+*p* *f* : on ajoute les permissions *p* (parmi `rwx`) aux utilisateurs *x* (parmi `ugo`) sur le fichier *f*
- `chmod` *x*−*p* *f* : on enlève les permissions *p* (parmi `rwx`) aux utilisateurs *x* (parmi `ugo`) sur le fichier *f*
- Si *x* est `a`, concerne tous les utilisateurs
- Si argument `-R`, s'applique à toute une hiérarchie

`find`

- Permet de trouver un fichier dans une hiérarchie
- On liste les critères :
  - `-type` type du fichier
  - `-name` nom du fichier
  - `-mtime` date de dernière modfication
  - `-size` taille du fichier

Plan

ps

- Affiche la liste des processus (par défaut, ceux de l'utilisateur, associés au même terminal)
- Plusieurs syntaxes différentes pour ses arguments, on en présente une seule
- Si on ajoute l'argument « `a` », affiche aussi ceux des autres utilisateurs, associés à un certain terminal
- Si on ajoute l'argument « `x` », affiche aussi ceux non associés à un terminal
- Si on ajoute l'argument « `u` », affiche plus d'informations, dont l'utilisateur
- Donc : `ps aux` affiche tous les processus, avec leurs utilisateurs

`kill` et `pkill`

- `kill -`*s* *p* envoie le signal *s* (numérique ou code type KILL) au processus de PID *p*
- `pkill -`*s* *m* fait de même, pour tous les processus dont le nom vérifie l'expression rationnelle *m*

Environnement : `which`, `env`, `export`, `unset`, `locale`

- `which` affiche le chemin complet d'un exécutable dans le `PATH`
- `env` affiche l'ensemble des variables d'environnement
- `export` ajoute une variable shell à l'environnement
- `unset` rend une variable shell non définie (en particulier, l'enlève de l'environnement)
- `locale` résume les variables d'environnement de locale

Plan

Tubes et manipulation de fichiers

- Le chaînage des tubes (pipeline) est particulièrement bien adapté au traitement de fichiers au format texte
- On chaîne des commandes qui produisent des données au format texte à d'autres qui les traitent, filtrent, découpent, trient, regroupent, etc.
- Permet des one-liners extrêmement puissants !

Produire une sortie : `echo`, `cat`, `tac`

- `echo` affiche sur sa sortie standard son argument, suivi d'un retour à la ligne
- `echo -n` ne produit pas de retour à la ligne
- `cat` affiche sur sa sortie standard le contenu de fichiers, ou le contenu de son entrée standard si aucun fichier en argument
- `tac` fonctionne comme `cat` mais produit les lignes dans l'ordre inverse

less

- Affiche un fichier ou son entrée standard page par page dans un terminal
- ESPACE ou PgDn pour aller à la page suivante
- « / » pour chercher du texte
- « q » pour quitter

`head` et `tail`

- `head -`*n* affiche les *n* premières lignes (d'un fichier ou de son entrée)
- `tail -`*n* affiche les *n* dernières lignes

`wc`

- Compte le nombre de :
  - `-l` lignes
  - `-w` mots
  - `-c` caractères

  d'un fichier ou de son entrée
- Par défaut, affiche les trois

sort et uniq

- `sort` trie un fichier ou son entrée ligne par ligne (attention à `LC_COLLAPSE`)
  - `-t` séparateur de champ (par défaut, espaces)
  - `-k` index du champ à utiliser pour le tri (par défaut, 1)
  - `-n` indique d'utiliser un tri numérique (par défaut, tri de chaînes via la locale)
- `uniq` renvoie les lignes uniques dans un fichier trié (attention à `LC_COLLAPSE`)
  - `-c` compte le nombre d'occurrences de chaque ligne

join

- Équivalent de l'opération de jointure (ou antijointure) dans une base de données : trouve pour chaque ligne d'un fichier trié une correspondance dans un autre fichier trié
- Possibilité d'indiquer un délimiteur (`-t`) et des numéros de champs (`-1` et `-2`) pour les deux fichiers
- Les deux fichiers doivent être triés selon ces champs (attention à `LC_COLLAPSE`)
- Avec `-v`, trouve les lignes d'un fichiers n'ayant pas de correspondance dans un autre fichier

cut et paste

- `cut -d` d `-f` i,j,k extrait les champs i, j et k d'un fichier dont les champs sont délimités par d
- `paste` produit un fichier dont la k-ième ligne est la concaténation (avec un délimiteur donné par `-d`) des k-ièmes lignes des fichiers d'entrée

Au-delà des fichiers texte

Cette approche de chaînage de tubes va au-delà des simples fichiers texte :

Fichiers compressés `zcat` (pour `gzip`), `bzcat` (pour `bzip2`)

Fichier JSON `jq` (syntaxe complexe)

Fichier XML `xpath` (syntaxe standard XPath)

Fichiers CSV `csvcut`, `csvjoin`, etc. du logiciel csvkit

Fichier sur le Web `curl` ou `wget -O -`

Page Web `w3m -dump`, `elinks -no-references -dump` ou `lynx -dump` pour en extraire le texte

Expressions rationnelles

- Langage permettant de décrire des motifs à rechercher dans une chaîne de caractères
- Par exemple : `(a|b)*#(a|b)*(#(a|b|#)*)?`
- Très utile pour indiquer des recherches dans des fichiers texte, des substitutions à appliquer, etc.

Syntaxe BRE et ERE

BRE	ERE	Sémantique
`.`	`.`	n'importe quel caractère
`[abc]`	`[abc]`	a ou b ou c
`[^abc]`	`[^abc]`	ni a ni b ni c
`[[:alpha:]]`	`[[:alpha:]]`	n'importe quelle lettre
`[0-9]`	`[0-9]`	un caractère entre 0 et 9
e`|`*f*	*e*`|`*f*	soit *e* soit *f*
e`*`	*e*`*`	*e* répétée $n \geq 0$ fois
e`\+`	*e*`+`	*e* répétée $n \geq 1$ fois
e`\?`	*e*`?`	*e* répétée $n \leq 1$ fois
`\(`*e*`\)`	`(`*e*`)`	capture de *e*
`\`*d*	`\`*d*	contenu de la *d*-ième capture
`^`	`^`	début de ligne
`$`	`$`	fin de ligne

grep et sed

- `grep` *e* recherche toutes les lignes dont une partie correspond à l'expression rationnelle *e*
- `grep -v` *e* recherche toutes les lignes dont aucune partie ne correspond à l'expression rationnelle *e*
- `grep -o` *e* affiche toutes les correspondances de l'expression rationnelle *e*
- `sed 's/`*e*`/`*r*`/'` remplace la première occurrence de *e* sur chaque ligne par *r* (qui peut contenir des références à des captures)
- `sed 's/`*e*`/`*r*`/g'` remplace chaque occurrence de *e* sur chaque ligne par *r* (qui peut contenir des références à des captures)
- Par défaut : BRE ; en ajoutant `-E`, ERE

Remarques sur `grep` et `sed`

- `sed` sait faire bien plus ! Il est d'ailleurs Turing-complet
- Un autre outil est `awk`, plus puissant que `sed` pour le traitement de fichiers avec des champs délimités, pour lequel on veut faire de l'agrégation, etc.
- À la fois `sed` et `awk` peuvent être remplacés par `perl`, un interpréteur d'un langage de programmation très puissant et concis, Perl, qui se prête bien aux one-liners

Plan

`man`

- Documentation intégrée à Linux, accessible par la commande `man`
- Divisée en sections :

1	Commandes
2	Appels systèmes
3	Fonctions de bibliothèques
4	Fichiers spéciaux
5	Formats de fichiers
8	Administration

(comparer `man 1 printf` et `man 3 printf`)

- man -k rechercher une page de `man` par mot-clef
- whatis court résumé sur une commande
- La plupart des commandes ont aussi une mini-aide intégrée, accessible avec l'argument `-h` ou `--help`

Références

- Le cours de second semestre *Systèmes d'exploitation*, où vous apprendrez à concevoir un système d'exploitation
- *Modern Operating Systems*, Andrew Tanenbaum & Herbert Bos, 4ème édition, 2014. Livre de référence sur les systèmes d'exploitation, par le créateur de MINIX.
- *The Linux Command Line : A Complete Introduction*, William E. Shotts, 2ème édition, 2019. Un bon livre sur la ligne de commande sous Linux. Actuellement en promotion avec plein d'autres livres sur Linux : `https://www.humblebundle.com/books/linux-no-starch-press-books`