



Introduction aux systèmes Linux

Plan

- Généralités sur SE Linux
- Historique
- Logiciel libre /normalisation
- Linux
 - ▣ Distributions
 - ▣ Structure linux
 - ▣ SGF linux
 - ▣ Notion d'utilisateur
 - ▣ Notion de Processus
- Interpréteur de commande Shell
 - ▣ Définition Shell
 - ▣ Variantes du Shell
 - ▣ Environnement du bash
- Commandes shell
 - ▣ Notion de commande et Rappels sur commandes
 - ▣ Type/Localisation
 - ▣ Echappement
 - ▣ Expansion des accolades

Qu'est qu'un Système d'Exploitation ?

□ Logiciel pour

- ▣ Gérer les ressources d'un ordinateur
- ▣ Utiliser les ressources d'un ordinateur

□ Objectifs

- ▣ Faciliter et simplifier l'utilisation d'un ordinateur.
- ▣ Créer une machine abstraite offrant un ensemble de services.
- ▣ Exploiter au mieux les capacités du matériel et gérer de manière optimale les ressources

Abstraction

- Cacher la complexité des machines pour l'utilisateur
- Abstraction du terme « Machine » selon Coy:
 - machine réelle = Unité centrale + périphériques
 - machine abstraite = machine réelle + système d'exploitation
 - machine utilisable = machine abstraite + application

Abstraction

- Cacher la complexité des machines pour l'utilisateur
- Abstraction du terme « Machine » selon Coy:
 - ▣ **machine réelle** = Unité centrale + périphériques
 - ▣ **machine abstraite** = **machine réelle** + système d'exploitation
 - ▣ **machine utilisable** = **machine abstraite** + application
- On distingue les notions
 - ▣ Processus, Jobs, Tâches
 - ▣ User
 - ▣ Fichiers

Types de SE

- **Systèmes mono-utilisateur mono-tâche**
 - ▣ un seul utilisateur et un seul programme exécuté à la fois
 - ▣ (ex : MS-DOS).
- **Systèmes mono-utilisateur multi-tâche**
 - ▣ un seul utilisateur qui peut exécuter plusieurs programmes en même temps
 - ▣ ex : Windows 95/98/Millennium/2000/XP.
- **Systèmes multi-utilisateur multi-tâche :**
 - ▣ plusieurs utilisateurs peuvent utiliser le même ordinateur et exécuter plusieurs programmes simultanément.
 - ▣ ex : Unix, Windows NT serveur / 2000 serveur / 2003 serveur / Linux (s)

Historique

- L'évolution des systèmes est liée à celle du matériel.
- Plusieurs générations d'ordinateurs. Classification depuis les années 70
- Au fil des générations l'évolution et le développement de la technologie dans l'industrie informatique a rendu les ordinateurs plus petits, plus rapides et moins chers.
- Andrew TANENBAUM
 - ▣ Première génération de 1945 à 1955 (tubes à vides
 - ▣ Deuxième génération de 1955 à 1965 (transistors et système batch)
 - ▣ Troisième génération de 1965 à 1980 (multiprogrammation et time-sharing, UNIX)
 - ▣ Quatrième génération depuis 1980 (LSI , VLSI et micro-ordinateur)

Historique

- Prolifération de systèmes divers DOS, Windows ...
- **UNIX**
 - ▣ Solaris (oracle)
 - ▣ BSD
 - ▣ macOS (dernières versions)
 - ▣ ...
- **Linux**
 - ▣ Ubuntu
 - ▣ Fedora
 - ▣ Debian
 - ▣ Android
- 5ème génération ? Technologie informatique basé sur l'intelligence artificielle. Le but ultime est de développer des compétences informatiques autonomes ?

UNIX / Linux Histoire

- 1964 : démarrage de Multics (MULTiplexed Information and Computing Service)
 - ▣ projet ambitieux de service pour les calculs intensifs
 - ▣ un système novateur : temps partagé, multitâche préemptif, système de fichier hiérarchique, multiutilisateur, sécurité.
 - ▣ Echec commercial
- 1969 : Les laboratoires Bell (filiales d'AT&T) se retirent de Multics
 - ▣ Ken Thompson et Dennis Ritchie, ingénieurs des laboratoires Bell, continuent leur réflexions sur un nouveau projet de système interactif simple et puissant.
 - ▣ Sur un PDP-7, Ken Thompson a commencé à écrire une version de système mono-utilisateur et qui a servi de base au système UNIX .
- En 1971, le nouveau système a été porté sur le mini-ordinateur PDP-11 et déployé pour dactylos du département des Brevets des laboratoires Bell, incluant des programmes d'édition et de formatage de texte.

UNIX / Linux Histoire

- 1972 : Difficulté de maintenir un système en langage d'assemblage,
 - ▣ Ritchie écrivait le langage C (langage de haut niveau basé sur un langage précédent, le B) ;
 - ▣ Thompson réécrivit le tout en C, ce qui permit d'améliorer la portabilité de l'OS sur différents environnements.
 - ▣ Ce système a acquis le nom Unics (Uniplexed Information and Computing Services)
 - ▣ un jeu de mots et l'orthographe se transforma rapidement en Unix par Kernighan.
- Juillet 1974 : Parution de L'article « Le système temps-partagé UNIX » de Ritchie et Thompson dans le CACM : le début du succès à grande échelle.
- 1975 : AT&T, dont dépendait Bell Labs, n'ayant pas le droit de commercialiser le produit, décide de distribuer le système Unix complet avec son code source à des fins éducatives dans les universités.
- Le code source de UNIX a été publié et développé en plusieurs versions (incompatibles)

UNIX / Linux Histoire

- Vers 1980 :
 - ▣ les développeurs de logiciels pouvaient avoir accès de partout au code source Unix, et l'adapter à leurs besoins
 - ▣ Unix a proliféré, et les variantes ont divergé.
 - ▣ Début d'une guerre es Unix
- Deux lignes majeures de Unix
 - ▣ Une de Berkeley
 - ▣ Une de AT&T,

UNIX / Linux Histoire

- En 1987, Unix System Laboratories, qui faisait partie des Laboratoires Bell à cette époque, commença à travailler avec Sun sur un système qui pourrait unifier les deux branches majeures de Unix.
- Le produit de leur collaboration, appelé Unix System V Release 4.0, fut sorti deux ans plus tard, et combinait des fonctionnalités de
 - ▣ System V Release 3, BSD, SunOS et du Xenix de Microsoft.
- Les branches les plus robustes actuellement sont AIX d'IBM, HP-UX de HP et Solaris de Sun.

UNIX / Linux Histoire

- ❑ Le code source de UNIX a été publié et développé en plusieurs versions (incompatibles)
- ❑ L'IEEE a développé un standard appelé POSIX auquel la plupart des versions actuelles de
- ❑ UNIX se conforment (ainsi que d'autres systèmes)
- ❑ En 1987, Andrew TANENBAUM a développé le MINIX un petit clone de UNIX à des fins pédagogiques.
- ❑ Linus Torvald a fait une version de production : LINUX

Historique (Unix)

- Au fil du temps le système a été modifié et a servi de base à d'autres systèmes d'exploitation de sorte qu'actuellement il en existe plusieurs variantes
- Officiellement UNIX, est une famille de systèmes d'exploitation multitâche et multi-utilisateur.
- Actuellement, Unix est à la fois une marque et une spécification, détenu par un consortium industriel appelé Open Group, et Seuls les Logiciels certifiés par l'Open Group peuvent porter le label Unix.

Logiciels libres

- Années 1960-1970
 - ▣ Développeurs de logiciels
 - ▣ Code source disponible.
 - ▣ Partage, collaboration, entraide, etc
- Début des années 1980
 - ▣ Sociétés de développement
 - ▣ Code source non disponible.
 - ▣ Confidentialité, restrictions d'utilisation, etc.
- Le débat est ouvert autour des logiciels libres

Les logiciels libres

- Concept introduit par Richard Stallman au début des années 1980
- Les logiciels « libres » se distinguent des logiciels « propriétaires » par le fait que les utilisateurs de ces logiciels disposent des droits suivants Les 4 libertés fondamentales
 - ▣ liberté d'exécuter le logiciel, pour tous les usages
 - ▣ liberté d'étudier le fonctionnement du logiciel → accès au code source
 - ▣ liberté de redistribuer le logiciel (y compris de le vendre)
 - ▣ liberté d'améliorer le logiciel et de publier ses améliorations
- Libre différent de gratuit : domaine public, freeware, shareware, ...
- «libre» s'oppose à «propriétaire» et à «logiciel privateur»

Logiciels libres

- Les logiciels « libres » se distinguent des logiciels « propriétaires » par le fait que les utilisateurs de ces logiciels disposent des droits suivants :
 - ▣ Liberté 0: Exécuter le programme pour tous les usages, ils peuvent l'utiliser librement;
 - ▣ Liberté 1: Étudier librement le fonctionnement du programme, et l'adapter à ses besoins (y compris au niveau de son code source (open source));
 - ▣ Liberté 2 : Redistribuer des copies à quelqu'un d'autre. (y compris de le vendre)
 - ▣ Liberté 3: Améliorer le programme et publier ses améliorations donc modifier et redistribuer les versions modifiées librement.
- Ces libertés sont souvent référencées comme les quatre libertés fondamentales des logiciels libres.

Logiciels libres

- Notons toute fois:
 - ▣ Libre différent de gratuit : domaine public, freeware, shareware, ...
 - ▣ «libre» s'oppose à «propriétaire» et à «logiciel privateur»
 - ▣ Les logiciels libres ne sont pas des logiciels libres de droits ni des logiciels du domaine publique.
 - ▣ Bien au contraire, ils sont protégés par le droit d'auteur, et les droits conférés aux utilisateurs ne le sont, que par une **licence** choisie par l'auteur du logiciel.

Licence Libre

- Contrat « juridique » qui accorde aux utilisateurs les libertés essentielles.
 - ▣ Utiliser le logiciel, pour tous les usages
 - ▣ Étudier le logiciel
 - ▣ Redistribuer des copies du logiciel
 - ▣ Modifier le logiciel et publier ses modifications
- Peut être généralisée à n'importe quelle œuvre numérique

Avantages des logiciels libres

- **Fonctionnalité et Rentabilité** : Possibilité de réutiliser tous les codes sources existants. Permettant d'aller plus loin vers des fonctionnalités les plus avancées dans leurs domaines respectifs.
- **Efficacité Fiabilité et sécurité** : Sur le long terme, la sélection des solutions techniques se révèlent plus performants et les plus efficaces. l'ouverture des sources et à la liberté de les consulter et de les modifier Les logiciels libres atteignent donc un très grande fiabilité et ceci est valable pour la sécurité, les faille détecté rapidement par un grand nombre d'utilisateurs avertis
- **Respect des standards et Garantie de la liberté** : La disponibilité des sources garantit la liberté. Les standards garantissent une interopérabilité parfaite avec les autres logiciels. En effet, pas des formats de fichiers opaques ou des protocoles de communication propriétaires. Ceci assure la Liberté de circulation des informations et la liberté d'expression de chacun ainsi que la liberté de choix.
- **Indépendance et pérennité** :La disponibilité des sources garantit aux utilisateurs la pérennité des logiciels libres qu'ils utilisent et leur indépendance vis à vis des éditeurs de ces logiciels.
- **Économiques** :Les logiciels libres sont souvent, de part la liberté de distribution que leur licence autorise, d'un prix de revient modeste.

Inconvénients des logiciels libres

- ❑ Complexité du modèle économique
- ❑ Fragmentation des solutions
- ❑ Difficulté d'intégration des solutions
- ❑ Légèreté et manque des documentations
- ❑ Faible compatibilité avec les logiciels propriétaires

Logiciels libres (Projet GNU) GNU is Not Unix

- GNU est l'un des projets libres les plus importants Lancé par Richard Stallman 1983
- Ce projet a pour but de recréer un système Unix complet, mais qui, à la différence des autres systèmes Unix, serait un système libre.
- Pour rappeler cette différence, l'acronyme « GNU » a été choisi en référence à la phrase « GNU's Not Unix »
- Supporté par la **FSF** depuis 1985 (Free Software Foundation)
- Créer une suite complète de logiciels
 - ▣ Mais le noyau tarde à arriver
 - ▣ Projet Hurd, initié en 1990 mais toujours inabouti à ce jour



GNU, FSF et GPL

- FSF (Free Software Foundation) a donné un cadre légal au projet GNU, en définissant la licence d'utilisation des programmes du projet GNU.
- Cette licence, nommé la GPL (General Public License), garantit la liberté de chacun et les droits d'auteurs des programmeurs.
- Tous les programmes du projet GNU sont bien entendu distribués sous cette licence.
- La FSF a également défini d'autres licences pour les autres aspects du projets,
 - ▣ Licence des librairies de programmes (LGPL, Lesser General Public License)
 - ▣ licence des documentations (FDL, Free Documentation License).
- Notons qu'il existe d'autres licences de logiciels libres

Normes Posix

- La diversité des versions autour des SE Unix ont démarré des projets de normalisation POSIX (Portable Operating System Interface) par le IEEE (Institute of Electrical and Electronics Engineers), formellement désignée par IEEE 1003.
- POSIX spécifie, dans 17 documents différents,
 - ▣ Les interfaces utilisateurs et les interfaces logicielles
 - ▣ La ligne de commande standard et l'interface de script qu'est le Bourne shell4.
 - ▣ Les autres commandes, services et utilitaires comprennent awk, echo, ed, et des centaines d'autres.
 - ▣ Les services d'entrées/sorties de base (fichiers, terminaux, réseau) doivent être présents ; le système doit supporter certains attributs spécifiques pour les fichiers.
 - ▣ POSIX définit aussi une interface de programmation standard, et celle-ci est prise en charge par la plupart des systèmes d'exploitation récents.

Open Source Licence

- Du logiciel libre est né l'open source, Fin des années 90 dont la philosophie est le droit d'obtenir le code source et le modifier pour son propre besoin.
- Ceci qui a engendré la constitution du Open Source Initiative (OSI) fondé en 1998
- Contrairement à FSF (Free Software Foundation) , OSI n'offre pas de licence mais des principes généraux
- Tout en étant bien séparées, ces deux mouvements peuvent aujourd'hui s'unir pour travailler ensemble, la communauté leur fait référence ensemble comme Free and Open Source Software (FOSS)
- Un Exemple : Open Source license : the BSD (Berkeley Software Distribution)

Libre Vs Open Source

- Le mouvement du libre est social, il soutient des valeurs philosophiques et politiques .
- L'open source met en avant la méthodologie de développement et de diffusion du logiciel.
- Deux visions qui arrivent au même point pratique. Pourtant, ne veulent pas être confondus.
 - ▣ Le libre est forcément open source. mais Il n'est pas forcément gratuit.
 - Confusion vient du fait que le mot “free” est traduit par libre mais aussi par gratuit.
 - ▣ Un logiciel peut être Open Source sans être libre (au sens stallman), il peut être soumis à des licences qui imposent d'autres restrictions.
- Toutes les licences reconnues libres par la FSF sont aussi reconnues open-source par l'OSI. Aussi bien les contraignantes comme la GPL que les plus permissives comme MIT, BSD ou Apache.

Linux

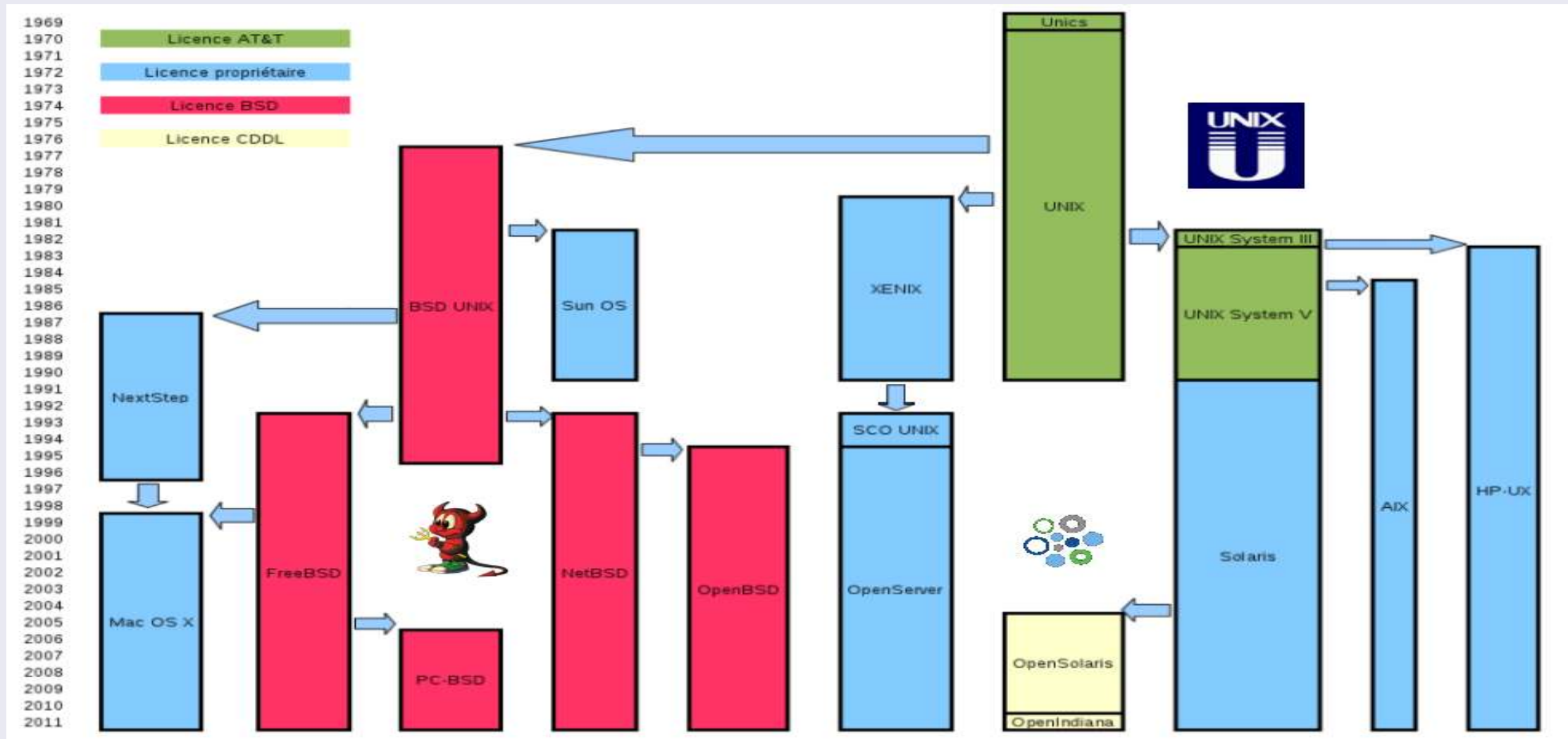
- ❑ Linux démarre en 1991 dans le cadre d'un projet universitaire par Linus Trovalds un informaticien d'origine Finlandaise à l'université d'Helsinki .
- ❑ Basé sur Minix un système d'exploitation de type Unix mais dédié à un usage éducatif.
- ❑ Linus développe donc un noyau d'un nouveau système d'exploitation qui sera baptisé Linux et annoncé en Août 1991
- ❑ Le projet GNU a fournis par ailleurs des composants systèmes qui ont pu être intégré avec le noyau linux pour donner lieu à un SE complet désormais GNU-Linux
- ❑ 1992 : passage à la licence libre GNU GPL

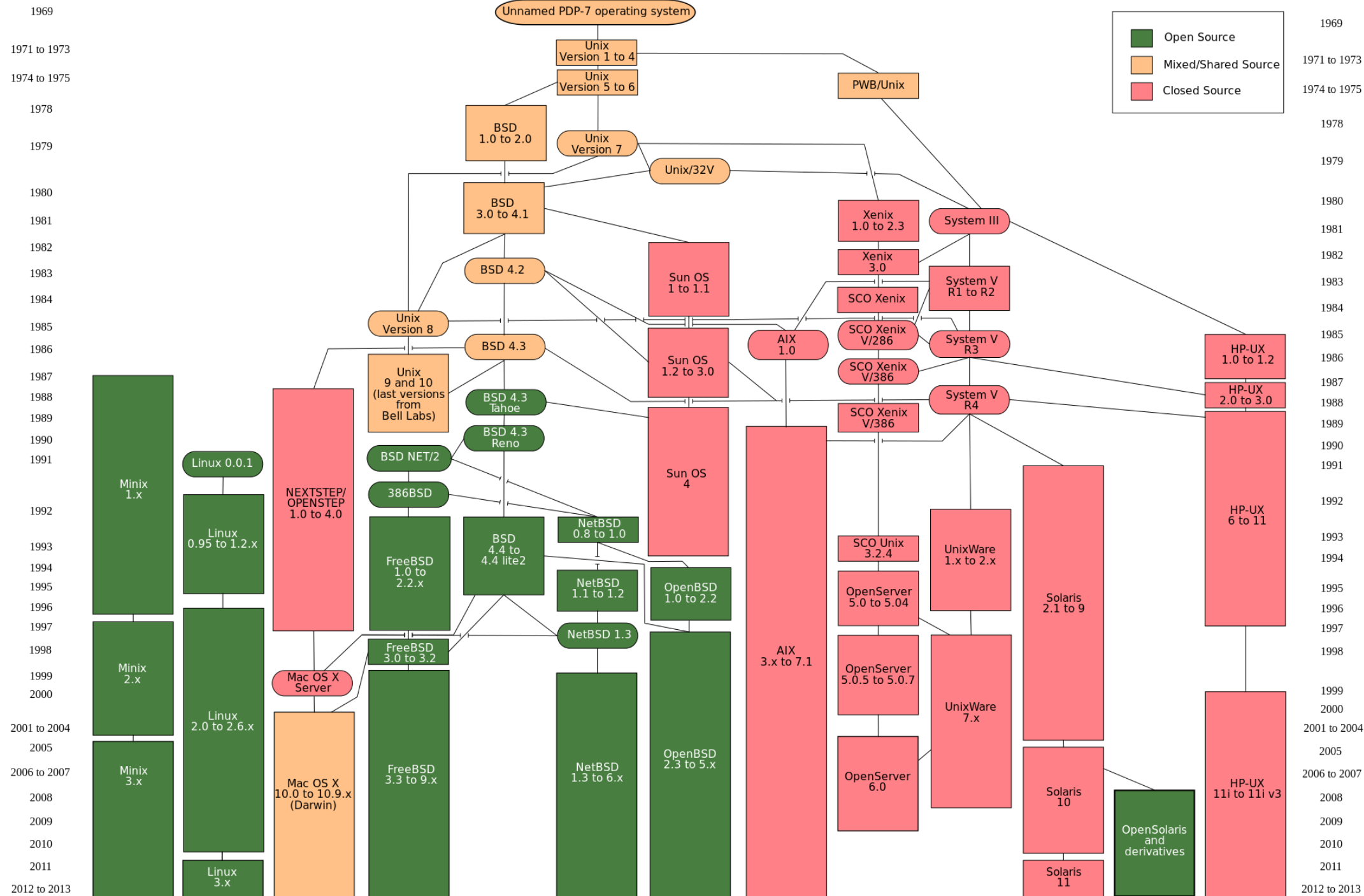
Linux

28

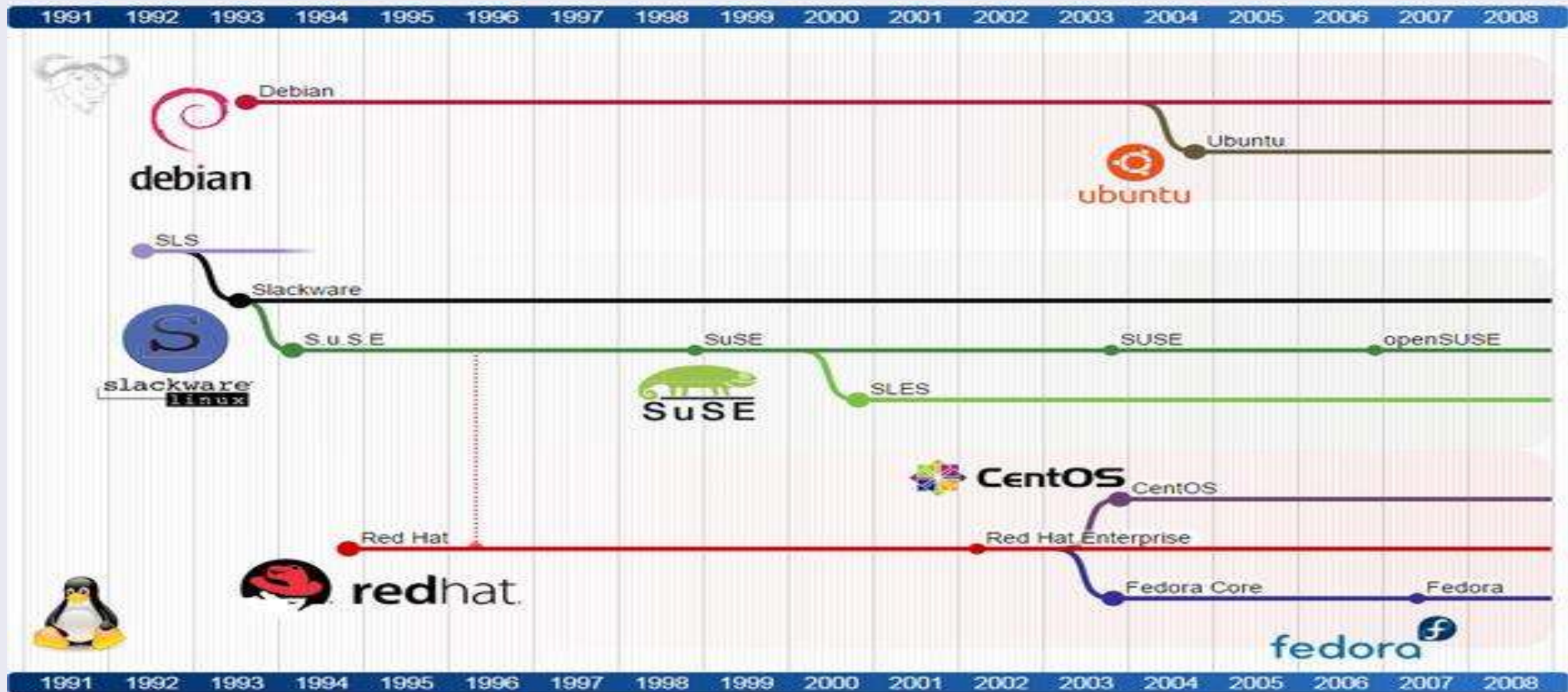
- ❑ GNU/Linux Désigne désormais une famille de systèmes d'exploitation Open source de type Unix fondé sur le noyau Linux :
- ❑ De nombreuses distributions existent
- ❑ Utilisé par plusieurs types de systèmes informatiques, PCs, serveurs, téléphones portables, systèmes embarqués ou encore superordinateurs.
- ❑ Android OS qui utilise le noyau Linux (mais pas GNU), équipe aujourd'hui 85 % des tablettes tactiles et smartphones
- ❑ Linux n'a pas été certifié Unix, **Linux n'est pas vraiment Unix**
c'est juste ... Comme Unix (Unix Like)

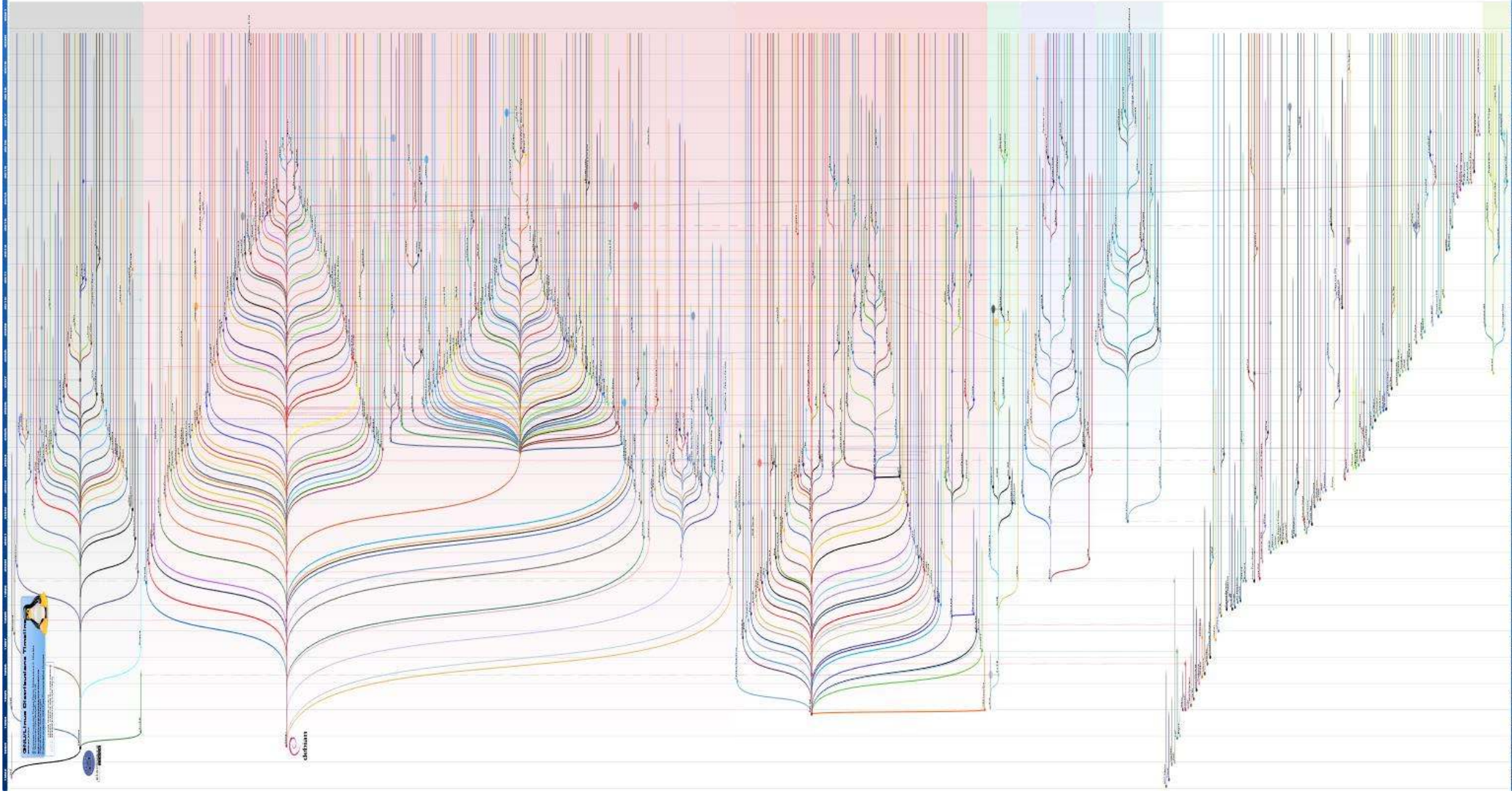
Famille Unix





Les principales familles de distributions Linux





Structure Linux

33

Système d'exploitation

- ❑ multi-tâches :
- ❑ multi-utilisateurs :
- ❑ multipostes :
- ❑ Outils / logiciels gratuits
- ❑ Environnement de développement
- ❑ Libre (et gratuit) !! :

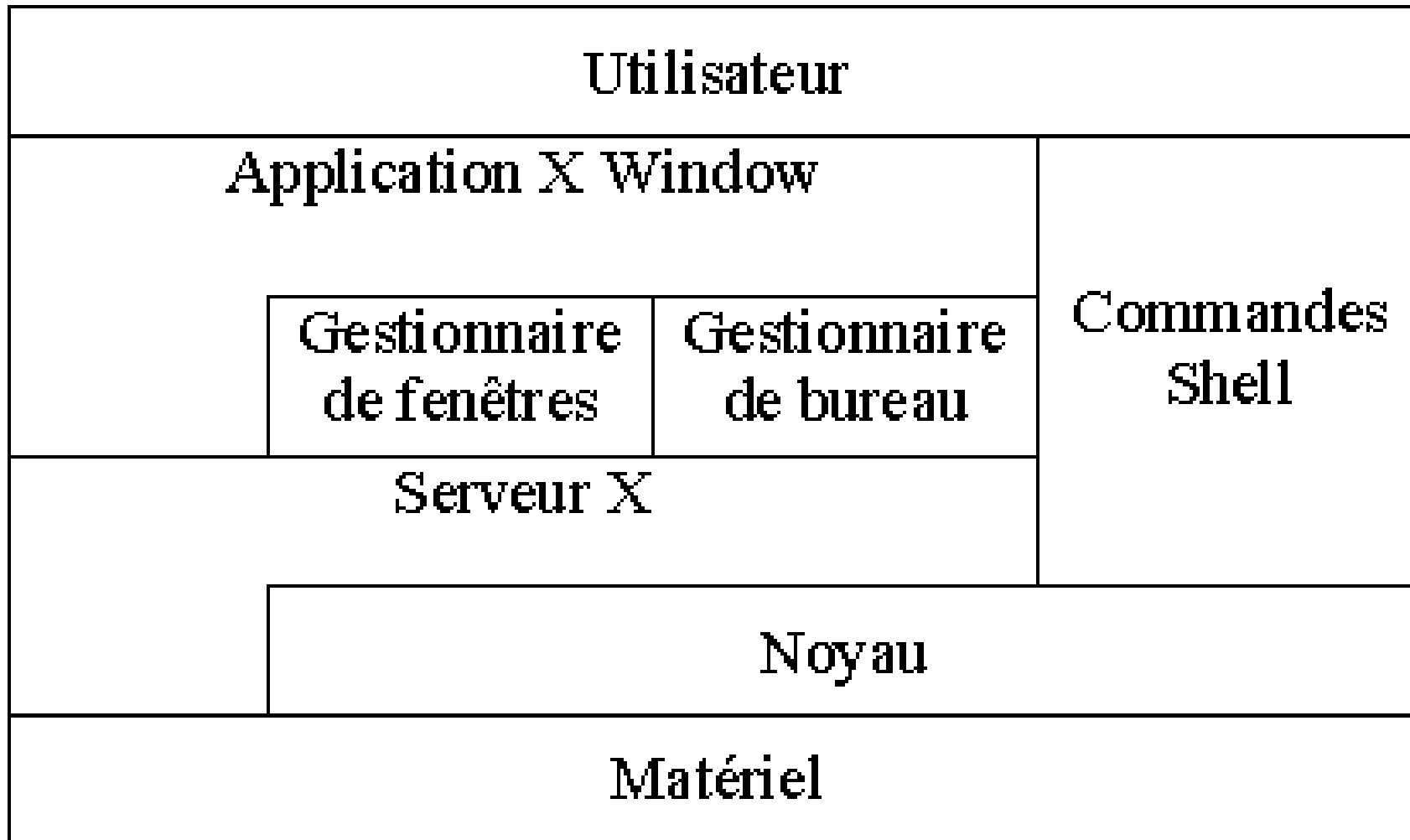
Structure du système Linux

34

- Kernel = Noyau
 - ▣ Gestion des Entrées/Sorties
 - ▣ Ordonnancement des tâches (scheduler)
 - ▣ Gestion de la mémoire virtuelle
- Shell
 - ▣ Interprétation des commandes utilisateur
 - ▣ Séquencement entre les programmes
- Application
 - ▣ Ajout et/ou retrait à l'aide de paquets
 - ▣ Développements modulaires pour partager les
 - ▣ bibliothèques entre applications

Structure Linux

35



Linux : SGF

- Linux est capable de gérer plusieurs systèmes de fichiers réels. La seule condition est qu'ils doivent tous fournir les services de base exigés par le système de fichiers virtuel ;
- Les applications peuvent utiliser plusieurs de ces systèmes de fichiers réels de manière uniforme, puisqu'elles n'utilisent que le système de fichiers virtuel. Cela simplifie leur programmation;
- Chaque système de fichiers réel étant indépendant des autres, il ne perturbe pas leur fonctionnement. En particulier, un système de fichiers corrompu ne corrompt pas les autres.

Linux : SGF

37

Applications					
Système de fichiers virtuel					
EXT2/3	ReiserFS	NFS	DOS	ISO9660	NTFS
Pilotes E/S					
Périphériques					

Linux : Fichiers

38

- La base du système de fichiers est le fichier individuel (file)
- Le premier caractère de chaque ligne renvoyé par la commande `ls -l` correspond au type du fichier
- On distingue plusieurs types de fichiers
 - ▣ - les fichiers ordinaires (*ordinary files*) ;
 - ▣ d les répertoires (*directories*) ;
 - ▣ Les fichiers spéciaux (special files ou devices).
 - l lien symbolique
 - p pipe nommé
 - s socket
 - c périphérique en mode caractère (clavier)
 - b périphérique en mode bloc(disque)

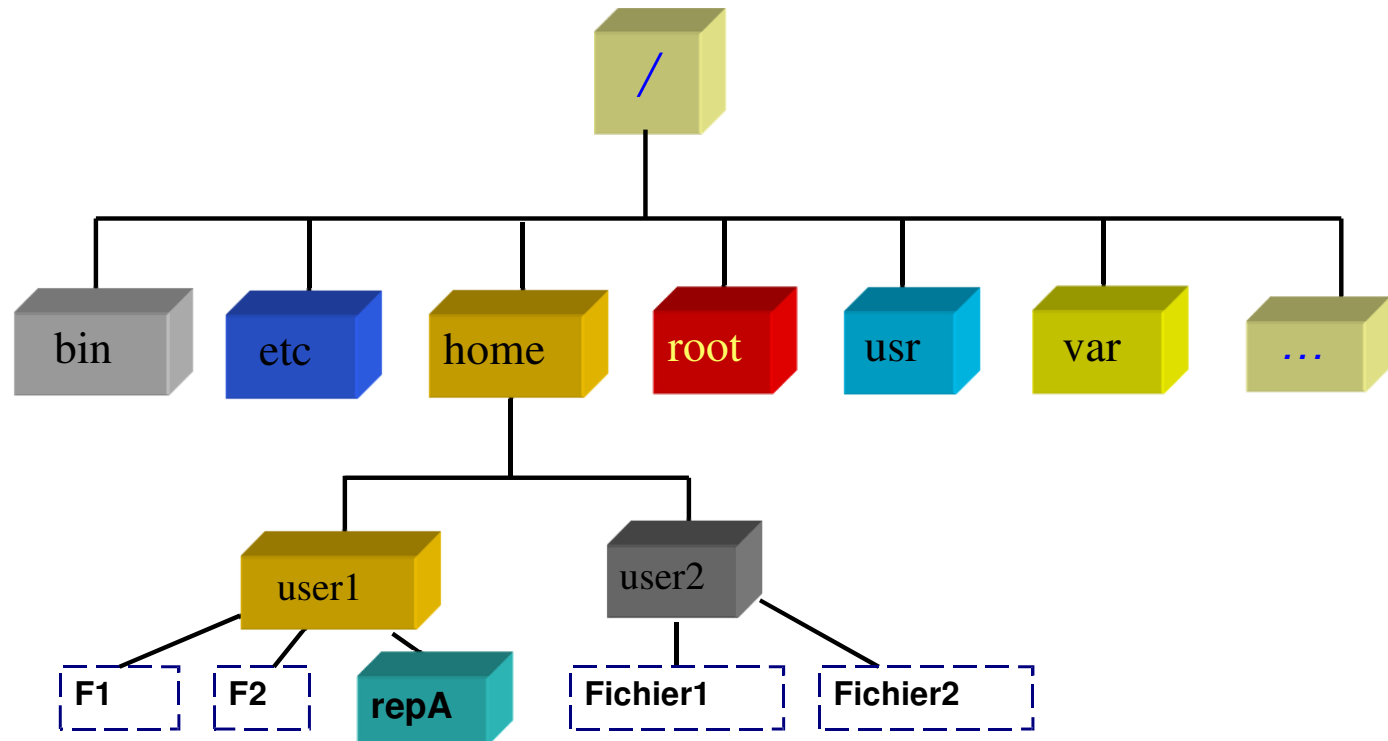
Linux: Répertoires

39

- Le répertoire courant ou répertoire de travail (working directory) est “celui où l’on se trouve”.
- Se déplacer dans le système de fichier revient à changer de répertoire courant. `cd nom-de-répertoire` (pour change directory).
- Pour connaître le chemin absolu du répertoire de travail : `pwd` (pour path working directory)
- Chaque usager dispose d’un répertoire d’accueil (home directory) qui lui appartient. Le nom de ce répertoire est contenu dans la variable `HOME`.
- Tout répertoire contient deux répertoires particuliers qui ont pour noms `.` et `..`.
 - ▣ `.` représente le répertoire de travail (répertoire courant),
 - ▣ `..` représente le répertoire parent du répertoire courant.
- `~` représente le répertoire d’accueil de l’utilisateur courant.

Linux : Organisation du système de fichiers

- Pour désigner un fichier dans l'arborescence de fichier, il faut indiquer le chemin qui mène à ce fichier dans l'arborescence



- **Chemin absolu** : Partant de la racine désigné par « / »
- **Chemin relatif** : chemin à partir du répertoire courant dans lequel on se trouve

Les utilisateurs et les groupes

Linux est multi-utilisateurs

- Chaque utilisateur a un login = nom identifiant unique
- Un utilisateur particulier : **root**
 - ▣ Administrer la machine
 - ▣ Configurer les comptes utilisateurs
 - ▣ Résoudre les problèmes systèmes
- Les utilisateurs sont organisés dans des **groupes**
 - ▣ s'ils partagent les mêmes fichiers
 - ▣ S'ils ont les mêmes responsabilités
 - ▣ S'ils appartiennent au même service

Notion de processus Linux

(Programme Vs Processus)

- Programme
 - ▣ Suite d'instructions permettant de réaliser un traitement.
 - ▣ Revêt un caractère statique

- Processus :
 - ▣ Par définition un processus est un programme en cours d'exécution, Une entité dynamique possède plusieurs états

- Unix/Linux système multi-tâches, l'unité de gestion est basé d'abord sur le processus
- Sur un système Unix/Linux Les processus sont organisé de manière hiérarchique

Notion de processus Linux

(Attributs des processus)

43

- UID : le propriétaire
- PID : le numéro du processus
- PPID : le numéro du processus père
- PRI : la priorité du processus
- NI : la valeur de NICE facteur de modification de priorité
- STIME: heure du lancement
- STAT l'état
- TTY le nom du device (terminal) vers lequel il est dirigé
- TIME le temps passé dans le cpu ;
- – le nom de la commande exécutée (COMMAND).

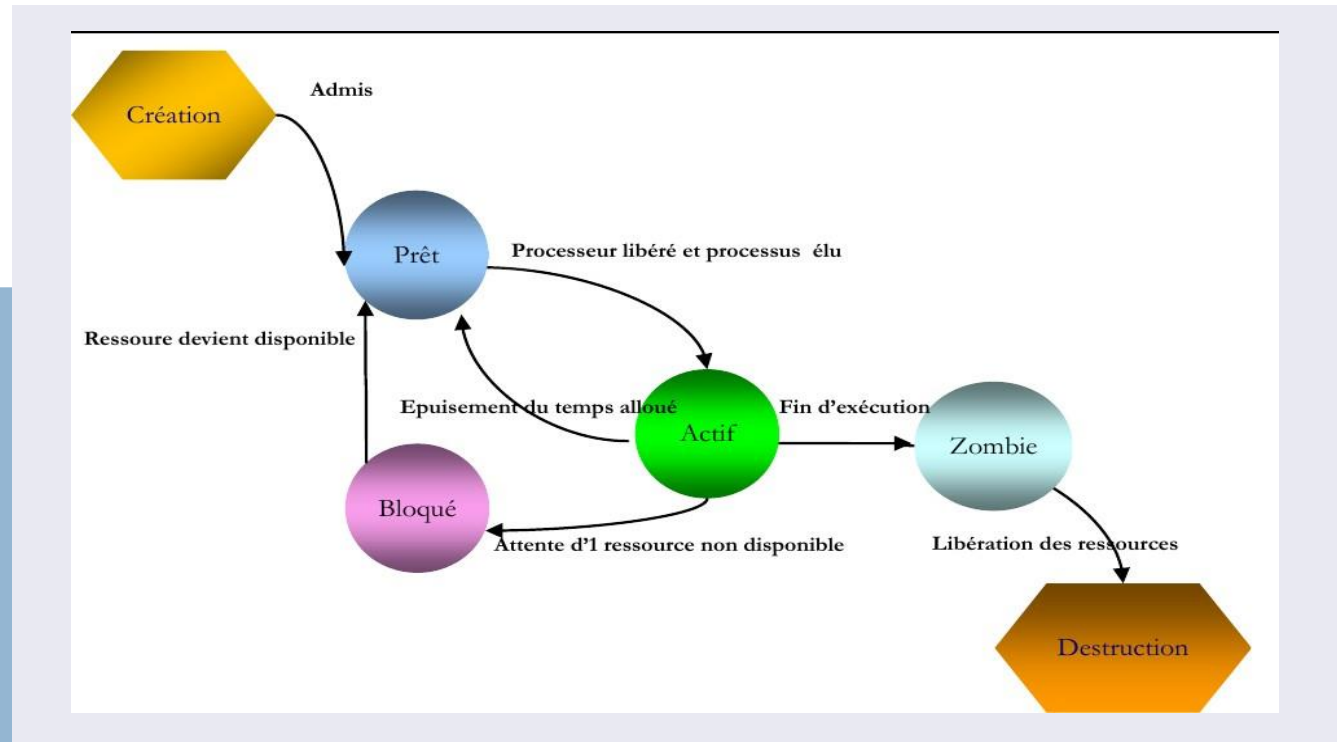
Notion de processus Linux

(cycle de vie d'un processus)

- ❑ Linux est un système multi-tâche : exécution en parallèle de plusieurs tâches
- ❑ Linux est un OS préemptif : A tout moment, il peut lancer ou arrêter n'importe quelle application
- ❑ La commande **ps** permet de connaître les processus actifs à un moment donné, associés à un terminal et lancés par l'utilisateur courant.

Plusieurs options possibles : `ps -l`,
`ps -ef`, `ps -elf`. `ps -aux`:

```
$ps -A ou $ps -e    Processus en exécution
ps -a              processus associé à un terminal
Ps -T              processus associé à ce terminal
ps -r              processus running
ps -x              processus de l'utilisateur
ps -He            voir avec hiérarchie les processus
ps -axjf
ps -g groupe
ps -o colonne (colonne cmd, user, pid ppid args ...)
```



Définition Shell

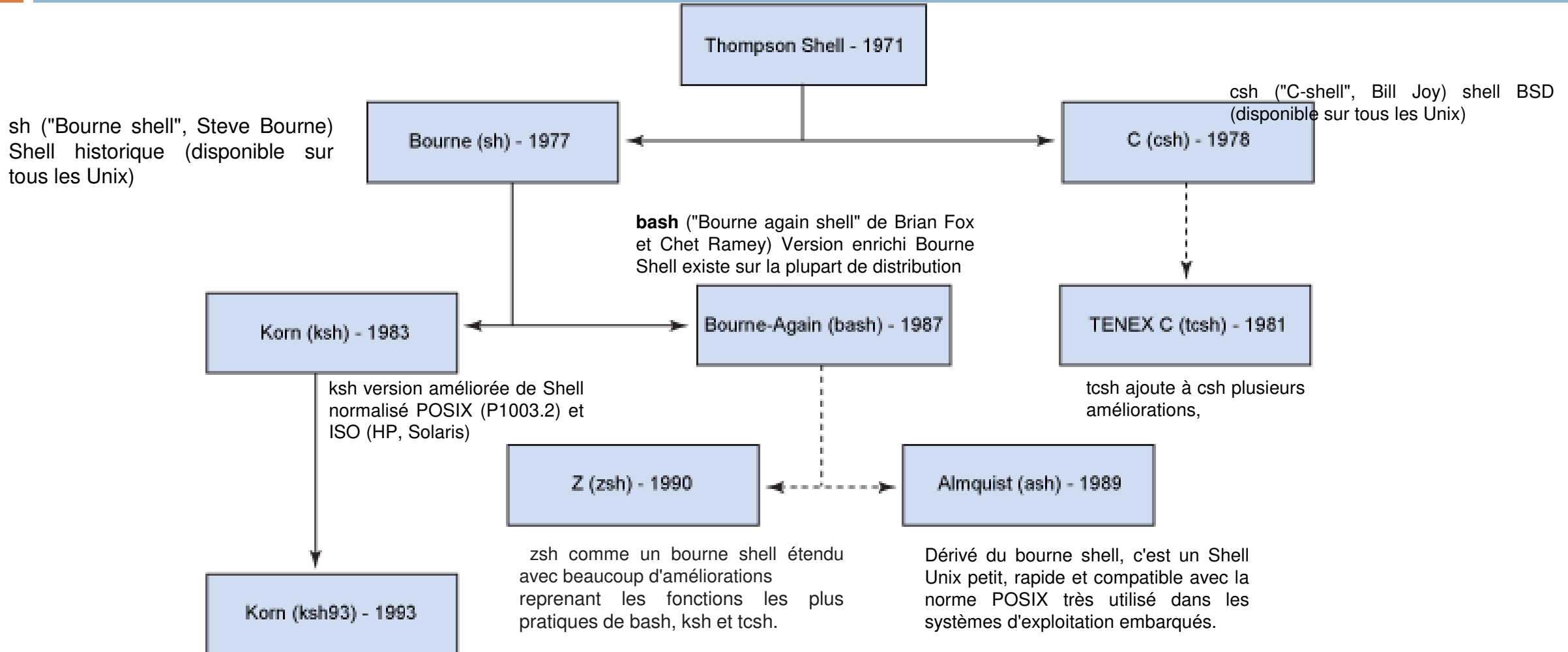
45

- ❑ **Shell** est une métaphore qui signifie enveloppe ou **Coquille**, qui entoure le noyau du système d'exploitation.
 - ❑ C'est un interpréteur de commandes, originellement utilisé sous Unix
 - ❑ C'est également un puissant langage de programmation: **Shell scripts**
 - ❑ Répandu avec différentes versions, la forme la plus simple est sh.
 - ❑ Remplaçable et personnalisable
 - ❑ Le Shell était la seule interface utilisateur disponible
-
- ❑ De nos jours, tous les systèmes, en plus des interfaces : ligne de commande (CLI, shell), les systèmes utilisent des interfaces utilisateur graphiques (GUI graphical user interface) comme macOS ou Windows 10



Shell Historique et variantes

46



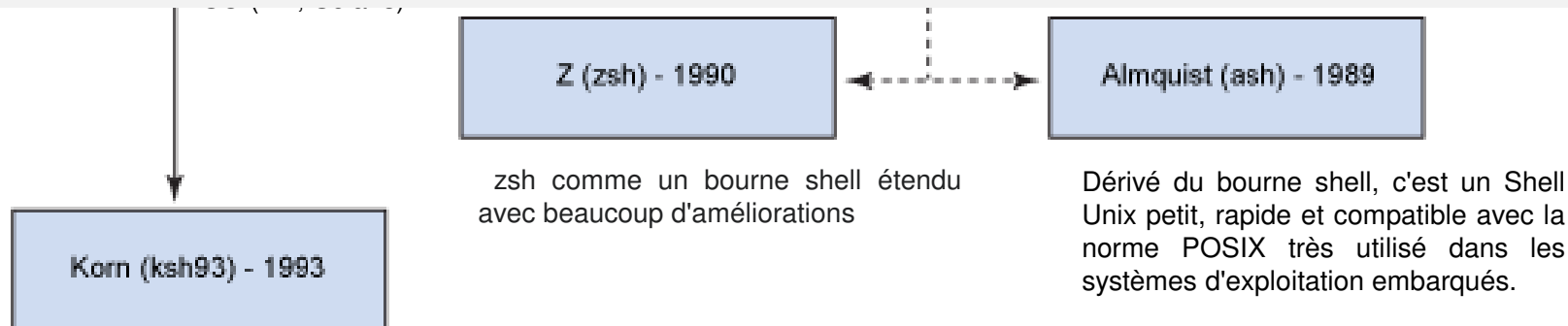
Shell: Historique des variantes

47

Thompson Shell - 1971

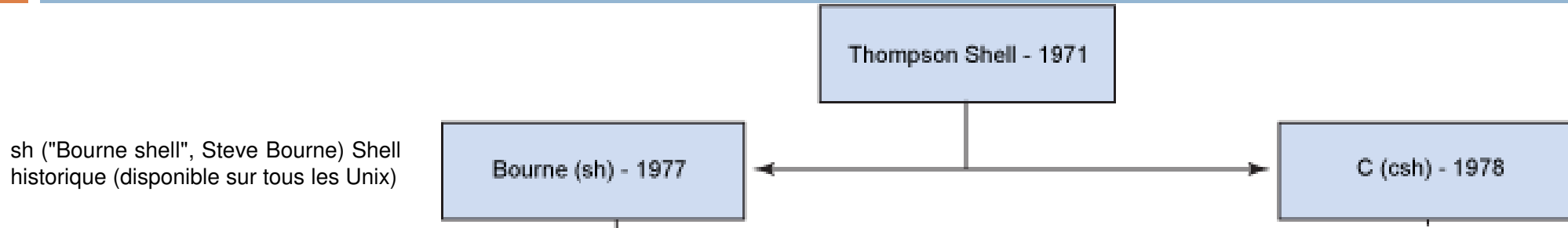
□ Thompson shell (invoqué sh)

- Par Ken Thompson est le premier shell de la première version de Unix (1971).
- Interpréteur de commandes simple et minimaliste (pas un langage de script)
- Propriétés innovantes qui ont servi pour les shells ultérieurs.



Shell: Historique des variantes

48



□ Bourne shell (sh)

- Par Stephen Bourne disponible sur tous les système Unix (1977).
- Le shell Bourne utilise une syntaxe similaire à celle du Shell original des tout premiers systèmes UNIX, comme System III.
- Avec un système de tubes innovant connectant la sortie d'une commande à l'entrée d'une autre, il permet d'écrire des scripts complexes à partir de commandes simples.
- C'est le Shell par défaut du compte root sur de nombreuses implémentations d'Unix.
- Le nom du shell Bourne sur la plupart des systèmes UNIX est /bin/sh

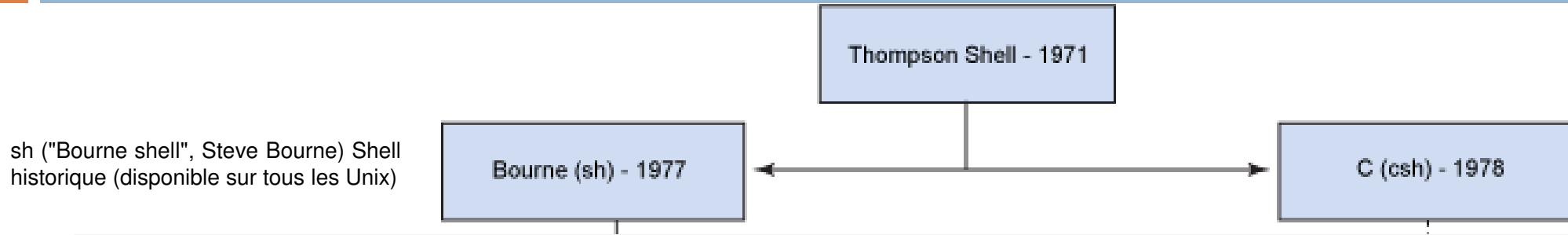


zsh comme un bourne shell étendu avec beaucoup d'améliorations

Dérivé du bourne shell, c'est un Shell Unix petit, rapide et compatible avec la norme POSIX très utilisé dans les systèmes d'exploitation embarqués.

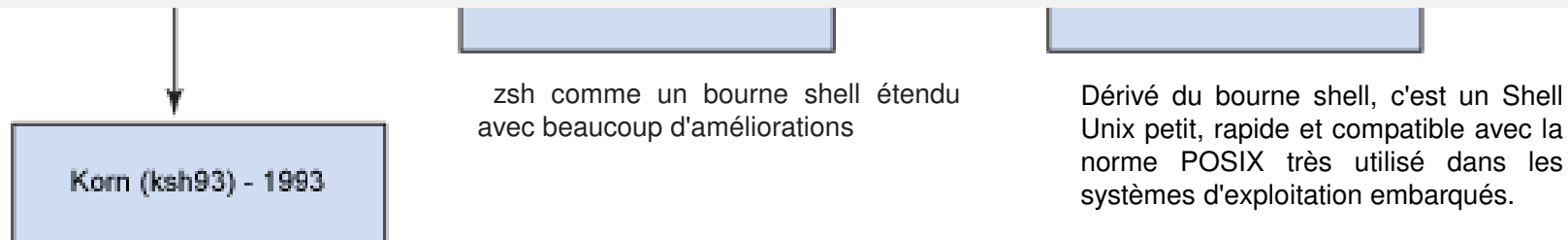
Shell: Historique des variantes

49



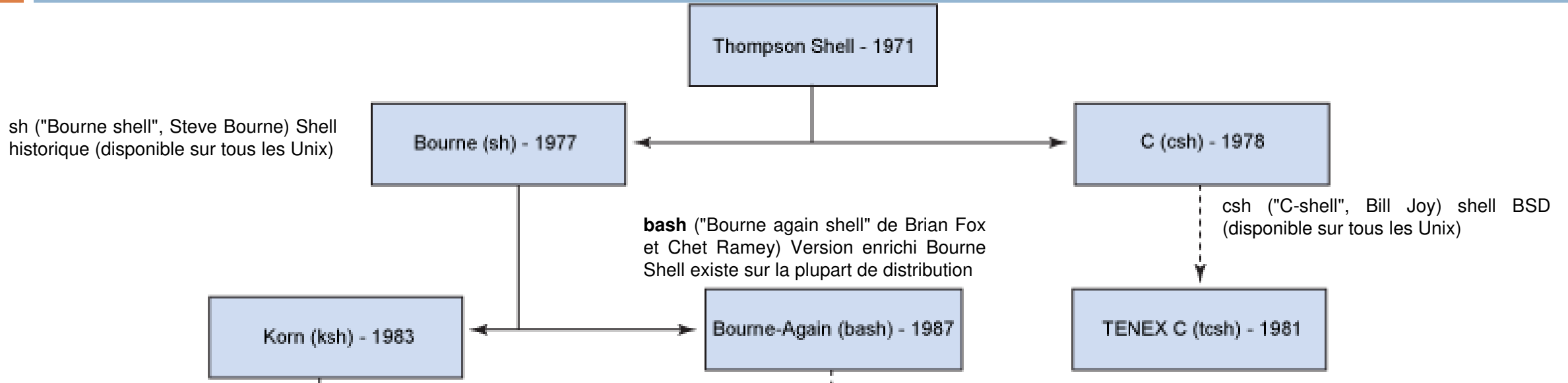
□ C shell (csh) :

- Shell Unix créée par Bill Joy étudiant à l'Université de Californie, Berkeley, 1978.
- Il a été largement distribué, à commencer par la version 2BSD de la Berkeley Software Distribution (BSD)
- Le C Shell utilise une syntaxe très différente, ressemblant beaucoup plus au langage C,
- S'appelle en général /bin/csh sous UNIX.



Shell: Historique des variantes

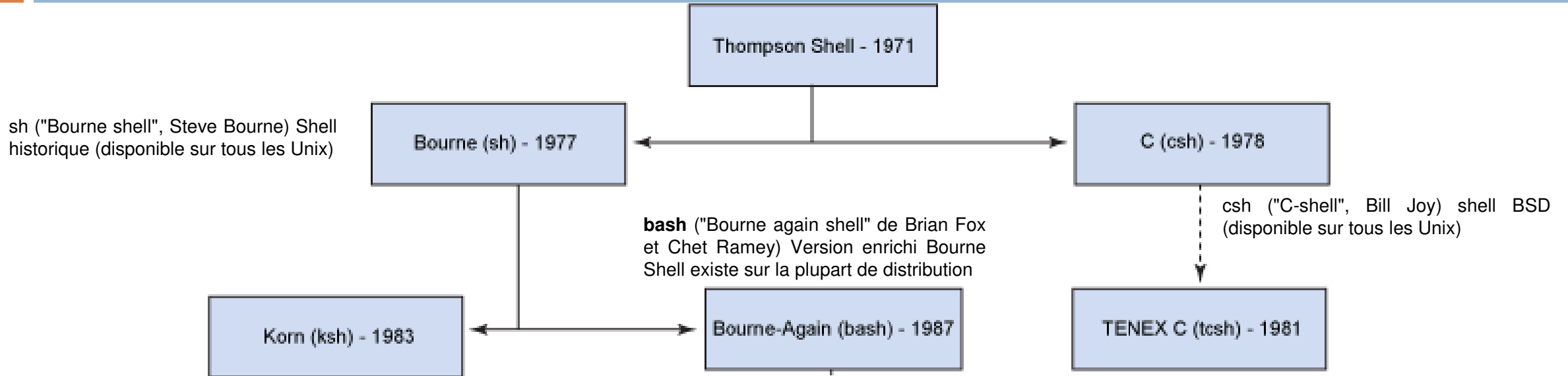
50



- **Bourne again shell (bash)**
 - ▣ Bourne-Again shell (bash) : c'est un Bourne Shell incluant des caractéristiques supplémentaires écrit dans le cadre du projet GNU.
 - ▣ Ce shell est souvent pré-installé et est le shell interactif par défaut pour les utilisateurs sur la plupart des systèmes Linux et macOS.
 - ▣ Comme Bash supporte se base sur la Bourne shell, tous les scripts écrits pour le shell Bourne standard fonctionneront avec bash.

Shell: Historique des variantes

51



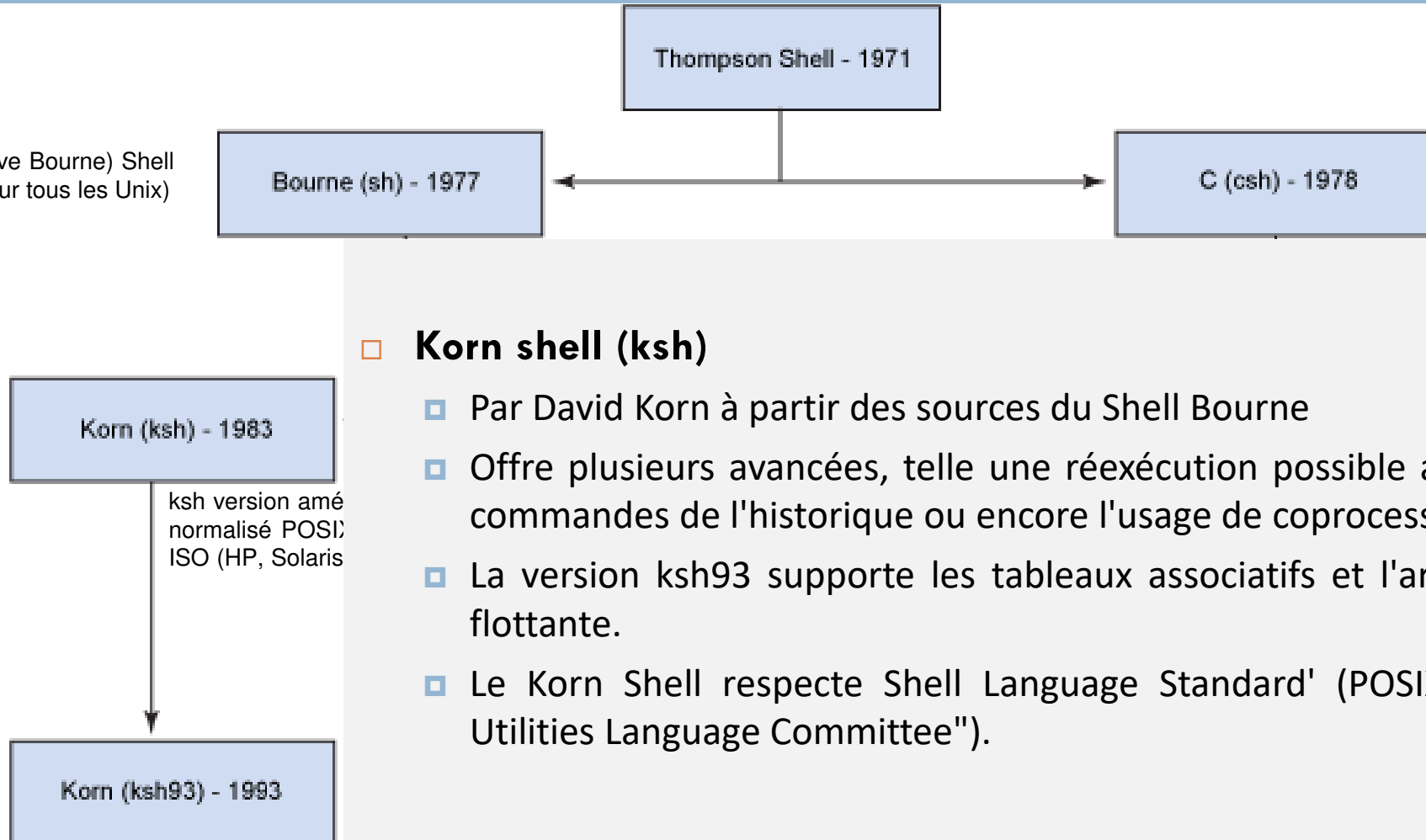
□ **TENEX Shell (tcsh):**

- Shell Unix basé sur et compatible avec csh.
- Il s'agit essentiellement d'un interpréteur de commandes C avec une complétion en ligne de commande programmable, une édition en ligne de commande et quelques autres fonctionnalités.
- Tcsh, qui est une version très améliorée du C Shell original.

Shell: Historique des variantes

52

sh ("Bourne shell", Steve Bourne) Shell historique (disponible sur tous les Unix)

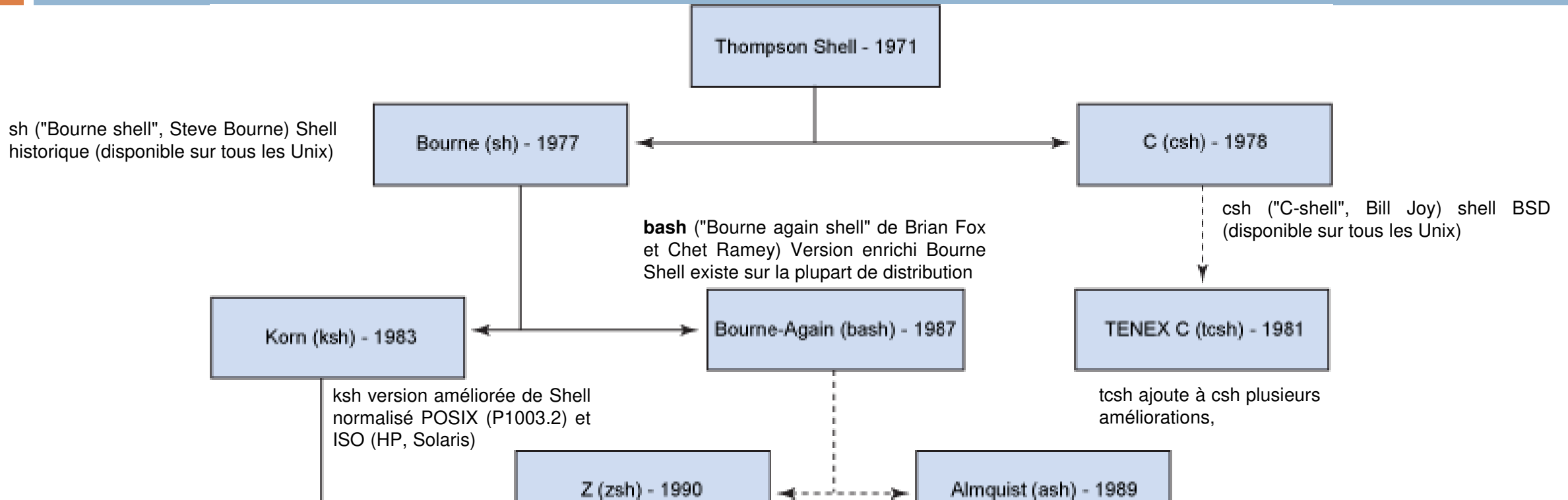


□ Korn shell (ksh)

- Par David Korn à partir des sources du Shell Bourne
- Offre plusieurs avancées, telle une réexécution possible avec substitution des commandes de l'historique ou encore l'usage de coprocessus.
- La version ksh93 supporte les tableaux associatifs et l'arithmétique à virgule flottante.
- Le Korn Shell respecte Shell Language Standard' (POSIX 1003.2 "Shell and Utilities Language Committee").

Shell: variantes

53



□ Z shell (zsh):

- Shell Unix qui peut être utilisé de façon interactive peut être vu comme un « Bourne shell » étendu avec beaucoup d'améliorations.
- Il reprend en plus la plupart des fonctions les plus pratiques de bash, ksh et tcsh
- C'est désormais le shell par défaut dans macOS depuis la version 10.15 de Catalina.

Almquist shell (ash), Debian Almquist shell (dash)

- Almquist shell (ash) : écrit en remplacement du Bourne Shell, sous licence BSD ; souvent utilisé dans des environnements aux ressources limitées. Les sh de FreeBSD, NetBSD (et leurs dérivés) sont basés sur des cendres qui ont été améliorées pour être conformes à POSIX.
- Debian Almquist shell (dash) : un remplacement moderne du shell ash dans Debian et Ubuntu
- Busybox: même si ce n'est pas un Shell à proprement dit mais plutôt un ensemble d'utilitaires Unix pour les petits systèmes et les systèmes embarqués, qui comprend 2 shells :
 - ▣ ash, un dérivé du shell Almquist ;
 - ▣ hush, une implémentation indépendante d'un shell Bourne.

Shell accès

- Accéder au Shell
 - ▣ Démarrage en mode texte
 - ▣ Shell accessible directement après authentification
- Dans un environnement de bureau
 - ▣ Basculer sur une autre console (Ctrl + Alt + Fx)
 - ▣ Ou utiliser un émulateur de terminal par exemple :
Konsole, Gnome Terminal, xterm
- Chaque compte utilisateur possède aussi un Shell par défaut, lancé à l'ouverture d'une invite de commande.
- Le Shell par défaut est précisé dans le fichier de configuration `/etc/passwd` dans le dernier champ de la ligne correspondant à l'utilisateur

Shell: prompt

56

- Lorsqu'un Shell est lancé (ouverture d'une Console ou d'un Terminal), il affiche une invite de commande (**prompt**)
- Prompt / invite de commande
 - ▣ Indique que l'interpréteur de commandes est prêt à exécuter des commandes.
 - ▣ Forme générale : `utilisateur@nom-de-poste:repertoire-courant$`
Exemple : `[user1@machine~]$`
 - ▣ Par défaut, ce prompt se termine par le caractère `$` pour un utilisateur normal et par un `#` pour le super-utilisateur root `[root@machine bin]#`

Shells

57

- Plusieurs Shell peuvent co-exister sur un même système. Consultez la liste des shells déjà installés sur le système d'exploitation avec la commande :

```
$cat /etc/shells
```

- Possibilité de changer de Shell dans une session Exemple: pour passer au Bourne shell, on tape :

```
$/bin/sh ou $sh
```

- Le Shell utilisé peut être connu par la commande ps qui liste les processus

```
$ps ou $ps -p $$
```

Shell: fichiers de démarrage

58

- Lors de l'ouverture d'une session, le Shell exécute des fichiers de configuration, qui peuvent contenir des commandes quelconques et sont généralement utilisées pour définir des variables d'environnement et des alias.
 - ▣ **csh** exécute le fichier `~/ .cshrc`
 - ▣ **tcsh** exécute le fichier `~/ .cshrc`
 - ▣ **sh** exécute le fichier `~/ .profile`
 - ▣ **bash** exécute le fichier `~/ .bash_profile` ou à défaut le fichier `~/ .profile`
 - On remarque bien que ces fichiers de configuration sont des fichiers cachés.
- Remarque : Chaque utilisateur peut ajouter des commandes shell au profil personnel `~/ .bash_profile`.

Fichiers d'initialisation du bash

59

- Shell de connexion (**login Shell**) : Lancé après une connexion d'un utilisateur au système (fournir un login et un mot de passe valides)
- Le Shell **/bin/bash** utilise une collection de fichiers de démarrage pour aider à la création d'un environnement de travail.
- Chaque fichier a une utilisation spécifique et peut affecter différemment la connexion et les environnements interactifs.
- Plusieurs fichiers sont nécessaires à la configuration d'un Shell de connexion Bash :
 - ▣ Certains affectent les environnements de tous les utilisateurs,
 - ▣ d'autres affectent uniquement l'environnement de l'utilisateur connecté

Fichiers d'initialisation du bash

60

- **/etc/profile** et **/etc/bashrc (/etc/bash.bashrc):**
 - ▣ pour une configuration globale qui affecte les environnements de tous les utilisateurs.

- **~/ .bash_profile (.profile ou .bash_login) et ~/ .bashrc :**
 - ▣ situés dans le répertoire home de l'utilisateur qui se connecte au système.
 - ▣ ces fichiers affectent uniquement l'environnement de cet utilisateur .
 - ▣ sont lus que par les login Shells juste après la connexion

- **~/ .bash_logout :**
 - ▣ situés dans le répertoire personnel de l'utilisateur connecté,
 - ▣ il est lu et exécuté par le Shell Bash quand cet utilisateur se déconnecte du système

Fichiers d'initialisation du bash

- Shell sans connexion (`non login shell`) : n'a pas besoin de login et de mot de passe) Par exemple celui qui est lancé via la ligne de commande (c'est-à-dire `[prompt]$ /bin/bash`)
- Les fichiers `~/ .bashrc` et `/etc/bashrc` sont, entre autres, lus et exécutés
- Les fichiers `~/ .bashrc` et `/etc/bashrc` sont donc lus et exécutés par les logins Shells et par les non-login Shells

Variables d'environnement

62

- ❑ Variables dynamiques permettant de paramétrer l'environnement d'exécution des processus.
- ❑ Elles affectent ainsi les comportements des processus et servent à communiquer des informations entre processus qui ne se trouvent pas sur la même ligne hiérarchique.
- ❑ Valeur de type chaîne de caractères, dont la signification lui est propre
- ❑ Les noms complètement en majuscules, mais ce n'est pas une obligation
- ❑ Chaque programme est susceptible de reconnaître un certain nombre de variables d'environnement qui lui sont propres, mais il existe également des variables d'environnement standards que tous les programmes utilisent.

Variables d'environnement

63

- Exemple: la variable d'environnement **PATH**
 - ▣ Contient la liste des répertoires dans lesquels le système doit rechercher les programmes à exécuter
 - ▣ Exemple de contenu de la variable PATH :
`/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin`
- Cette variable PATH permet donc de lancer les programmes en tapant simplement leur nom, et de laisser le système rechercher le fichier de ce programme dans chacun des répertoires indiqués dans cette variable

Variables d'environnement

64

Variables d'environnement **prédéfinies**

- ❑ **HOME** chemin d'accès au répertoire initial (personnel) de l'utilisateur
- ❑ **PATH** suite de chemins d'accès aux répertoires des exécutables
- ❑ **PS1** invite principale du Shell en mode interpréteur, Variable contenant une chaîne de texte réalisant la mise en forme du message d'attente (prompt) en ligne de commandes. Par exemple `[\u@\h \W]\$`
- ❑ **PS2** invite secondaire du Shell en mode programmation
- ❑ **IFS** séparateurs de champ des arguments
- ❑ **MAIL** chemin d'accès à la boîte aux lettres utilisateur
- ❑ **MAILCHECK** intervalle en sec au bout duquel le mail est contrôlé

Variables d'environnement

65

Variables d'environnement prédéfinies

- ❑ **CDPATH** liste de chemins d'accès pour la commande **cd**
- ❑ **ENV** nom du fichier des variables d'environnement
- ❑ **TERM** nom du type de terminal
- ❑ **USER** Nom de login de l'utilisateur. Cette information est également disponible au travers de la variable d'environnement **LOGNAME**
- ❑ **SHELL** Chemin de l'exécutable du Shell actuellement utilisé. Sous Linux, il s'agit souvent du Shell **BASH** (le contenu sera donc **/bin/bash**)
- ❑ **PWD** Cette variable contient le répertoire de travail actuel

Variables d'environnement

- Par défaut, les programmes sont lancés avec l'environnement du programme qui les lance, c'est-à-dire dans la plupart des cas **l'environnement d'exécution du Shell**.
- Les programmes peuvent également définir de nouvelles variables d'environnement, qui seront ainsi accessibles par les programmes qu'ils lanceront eux-mêmes.
- Comme tout programme, le Shell dispose d'un environnement qu'il utilise pour stocker ses propres variables
- le Shell utilise donc deux environnements différents :
 - ▣ Son propre environnement, qui contient les variables locales à la session du Shell en cours.
 - ▣ L'environnement d'exécution, dont les variables d'environnement sont transmises aux programmes que le Shell lance.

Commandes Shell

Notion de commande

Rappels sur les principales commandes

Localisation

Echappement

Notion de commande

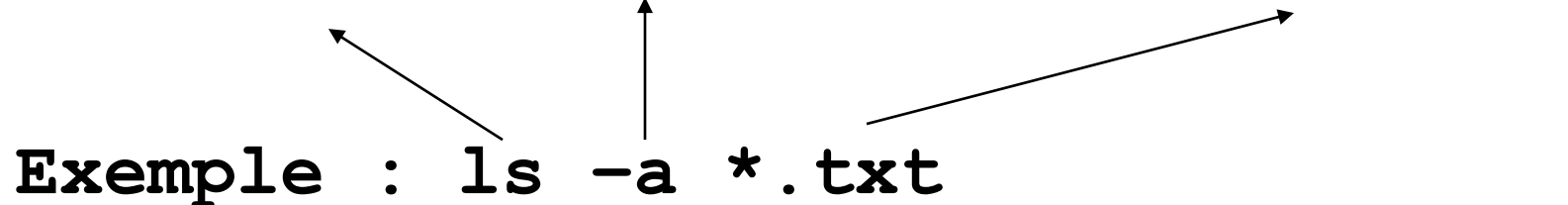
Un programme ou une partie d'un programme

- Fichier spécifique: les .exe sous Windows, fichier ordinaire Linux
- Un élément du shell : commandes internes

Format général des commandes:

`commande [-option(s)] [argument(s)]`

Exemple : `ls -a *.txt`



Commandes Shell

- Toute commande est instruction textuelle composée de trois parties

Commande <options> <attributs>

- <options> : commence obligatoirement par « - » en format court (-a) ou « -- » en format long (--all).
Chaque commande dispose de ses propres options
 - <attributs> : représente l'argument de la commande : fichier, répertoire, utilisateur, etc.
-
- Exemples de commandes
 - echo : permet d'afficher un message sur l'écran
 - clear : effacer l'écran
 - whoami : affiche l'utilisateur courant
 - date : afficher la date courante
 - cal : afficher la date courante

Commandes Shell

Raccourcis Bash

CTRL + r : recherche dans l'historique

CTRL + a : début de ligne

CTRL + e : fin de ligne

CTRL + ⬅ : mot précédent

CTRL + ➡ : mot suivant

CTRL + k : coupe de la position courante jusqu'à la fin de la ligne

CTRL + y : colle ce qui a été précédemment coupé

CTRL + d : efface le caractère à droite du curseur

CTRL + t : transpose le caractère sous le curseur avec le caractère suivant

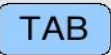
ESC + d : efface le mot à droite du curseur

ESC + t : transpose le mot précédent avec le mot suivant

Commandes Shell

71

□ Complétion

	:	complète la saisie en cours
 + 	:	affiche une liste des complétion possibles

□ Historique

 +  <i>texte</i>	:	recherche <i>texte</i> dans l'historique
	:	commande précédente
	:	commande suivante
 <i>nombre</i>	:	exécute la commande numéro <i>nombre</i>
 <i>texte</i>	:	exécute la dernière commande débutant par <i>texte</i>

□ La commande `history`

Commandes Shell : Utiliser l'aide

72

Il existe trois possibilités pour avoir de l'aide sur une commande

- L'aide en ligne par la commande **man** : qui permet de consulter les pages de manuel de linux
- Aide interne des commandes
 - ▣ Aucun standard
 - ▣ Très sommaire
 - ▣ Généralement, l'aide interne d'une commande est utilisée à travers les options `-h` et `--help`
- L'aide hiérarchisée info
 - ▣ Ensemble de pages hiérarchisées
 - ▣ Réparties en plusieurs niveaux ou « nœuds » Rubriques
 - ▣ Du plus général au plus particulier
 - ▣ Liens Marqués d'un astérisque (*)

Types de commandes

□ Commandes **externes** au Shell

- ▣ Dans un fichier programme ;
- ▣ Créent un nouveau processus (fils du processus Shell).
- ▣ Exemples: `ls`, `mkdir`, `vi`, `sleep`

□ Commandes **internes** au Shell

- ▣ Programmées dans le Shell ;
- ▣ Sont exécutées par le processus Shell lui-même
- ▣ Exemples de commandes internes : `cd` , `echo` , `for` , `pwd`

Pour connaître le type d'une commande

\$ type commande

-a : pour toutes les correspondances

-t : sortie raccourcie

□ Commandes **alias** définies via le Shell par Renommage et combinaisons de précédentes

Localisation des commandes

- **\$whereis [-bms] filename.**

Retrouver des exécutable et fichiers liés (documentation, configuration)

- ▣ **-b** : seulement les fichiers binaires

- ▣ **-m** : seulement les pages man

- ▣ **-s** : seulement les sources

- **\$\$ which [-a] filename**

recherche le(s) fichier(s) exécuté pour une commande **dans \$PATH**

- ▣ **-a** : afficher toutes les correspondances

Commandes Shell

75

- Gestion des fichiers
 - `ls ls -l`
 - `cd/pwd`
 - `mkdir/ rmdir`
 - `touch`
 - `rm`
 - `mv`
 - `cp`
- Gestion des droits d'accès
 - `chmod`
 - `Umask`
- Afficher le contenu d'un fichier
 - `cat`
 - `vi, emacs, gedit, vim`
 - `more, less`

Commandes Shell

76

□ Rappel sur les principales commandes

cat, head, tail, join, paste, cut, grep, look,
lpr, more, pg, sort, tr, uniq, wc

Grep et expressions régulières

find, sed et awk

Changer de répertoire

\$ cd [répertoire]

- "Change directory « Argument »[répertoire]
 - ▣ chemin absolu exemple : **/home/monrep**
 - ▣ chemin relatif, exemple : **monrep**
 - ▣ Utiliser un caractère spécial (raccourcis) : **~**, **.** .
- Changer de répertoire : par les raccourcis
 - ▣ **~** répertoire personnel
 - ▣ **.** répertoire courant
 - ▣ **..** répertoire parent du répertoire courant
 - ▣ **-** dernier répertoire (courant)

Pwd / echo

\$ pwd [-LP]

□ "Print working directory"

▣ -L suivre les liens symboliques

▣ -P ne pas suivre les liens symboliques

□ **\$ echo [-neE] [arguments . . .]**

▣ -n Supprime le retour chariot (retour à la ligne) final

▣ -e affiche les séquences d'échappement

▣ -E Désactive l'interprétation des séquences d'échappement

Le caractère ~ (tilde)

- "~" représente le répertoire **HOME** de l'utilisateur. Par exemple, pour l'utilisateur **toto**

```
$cd ~  
$pwd  
/home/toto
```

- Copier le fichier "**fichier1**" du répertoire **/tmp** dans **/home/toto**

```
$cd /tmp  
$cp fichier1 ~
```

- Copier le fichier "**fichier2**" du répertoire **/tmp** dans **/home/toto/mesfichiers**

```
$cd /tmp  
$cp fichier2 ~/mesfichiers
```

Le caractère ~ (tilde)

- Si le caractère "~" est immédiatement suivi d'un mot, ce dernier est considéré comme un nom d'utilisateur. Copier le fichier "fichier3" du répertoire /home/toto vers le répertoire /home/titi

```
$cd ~  
$pwd  
/home/toto  
$cp fichier3 ~titi
```

Copier le fichier "fichier4" du répertoire /home/toto vers le répertoire /home/titi/tmp

```
$ cd ~  
$ pwd  
/home/toto  
$ cp fichier4 ~titi/tmp
```

Echappement

- Échapper est une méthode pour mettre entre guillemets un caractère seul.
- L'échappement (\) précédant un caractère dit au shell d'interpréter le caractère littéralement.
- Avec certaines commandes et utilitaires, tels que **echo** et **sed**, échapper un caractère peut avoir l'effet inverse - cela peut activer un comportement particulier pour ce caractère.

Echappement :

Significations spéciales des caractères échappés

<code>\a</code>	« alerte » (sonore ou visuelle)
<code>\b</code>	retour arrière (backspace)
<code>\c</code>	suppression du retour-chariot final
<code>\f</code>	saut de page
<code>\n</code>	nouvelle ligne
<code>\r</code>	retour-chariot
<code>\t</code>	tabulation horizontale
<code>\v</code>	tabulation verticale
<code>\\</code>	backslash
<code>\nnn</code>	le caractère dont le code ASCII octal vaut nnn (un à trois chiffres)
<code>\xnnn</code>	le caractère dont le code ASCII hexadécimal vaut nnn (un à trois chiffres)

Expansion d'accolades

- L'expansion d'accolades est une technique utile pour générer automatiquement des listes de chaînes qui peuvent être utilisées dans des scripts et des alias et sur la ligne de commande Linux.
- Une expansion d'accolade est contenue { }.
 - ▣ liste d'éléments séparés par des (,) ou d'un spécificateur de plage.
 - ▣ Les espaces ne sont pas autorisés à l'intérieur des accolades, sauf si la chaîne entre guillemets « " »
 - ▣ Pour une liste séparée par des virgules, le processus d'expansion prend chaque élément à son tour et le transmet à la commande appelante.

Extensions simples

- Une expansion d'accolade est contenue { }.
- Il peut s'agir d'une liste d'éléments séparés par des (,) ou d'un spécificateur de plage.
- Les espaces ne sont pas autorisés à l'intérieur des accolades, sauf si la chaîne entre guillemets « " »
- Pour une liste séparée par des virgules, le processus d'expansion prend chaque élément à son tour et le transmet à la commande appelante.

Exemple avec echo

- Les virgules sont ignorées. Une liste peut être composée de mots ou de chiffres.
- L'ordre des éléments de la liste est complètement arbitraire.

```
$echo {one,two,three,four}  
one  two  three  four
```

```
$echo {1,2,3,4}  
1  2  3  4
```

```
$echo {4,2,3,1}  
4  2  3  1
```

```
// intervalles
```

```
$echo {3..12}  
3  4  5  6  7  8  9  10  11  12
```

```
$echo {5..1}  
5 4 3 2 1
```

```
$echo {4..-4}  
4 3 2 1 0 -1 -2 -3 -4
```

```
$echo {q..v}  
q r s t u v
```

```
$echo {f..a}  
f e d c b a
```

Exemples

#Concaténation et imbrication

```
$echo {q..t}{1..3}
```

```
q1 q2 q3 r1 r2 r3 s1 s2 s3 t1 t2 t3
```

```
$echo {part-1,part-2{a,b,c,d},part-3}
```

```
part-1 part-2a part-2b part-2c part-2d,part-3
```

```
$echo {1..6..2}
```

```
1 3 5
```

```
$echo {{5..0},{1..5}}
```

```
5 3 2 1 0 1 2 3 4 5
```

#Préambule et Postscript

```
$echo chapter{1..3} // prembule
```

```
chapter1 chapter2 chapter3
```

```
echo {contents,paper,bilbiography}.tex // Postscript  
contents.tex paper.tex bilbiography.tex
```

```
$ echo chapter{1..3}.txt // les deux  
chapter1.txt chapter2.txt chapter3.txt
```

Commandes Exemples

```
$touch file-{1..4}.txt
```

```
$zip source-code program{.c,.h,.css}
```

```
$cp brace/new/prog-1.c{,.bak}
```

```
$diff brace/{new,old}/prog-1.c
```

```
$mkdir -p {source,build,man,help{/pages,/yelp,/images}}
```

```
wget https://www.URL-of-your-  
choice.com/test{1,2}/picture{001,002}.jpg
```

Variables bash et expansion des accolades

- Un exemple d'expansion '*'

```
$ls -ld /home/*  
/home/riri  
/home/fifi  
/home/loulou
```

- Si dans le cadre d'un script définir d'un côté ma liste de dossiers, et de l'autre côté d'utiliser cette liste dans une boucle par exemple, comme cela :

```
$MaListe=/home/*  
$ for Dossier in $MaListe; do echo $Dossier; done  
/home/riri  
/home/fifi  
/home/loulou
```

Variables bash et expansion des accolades

- ❑ Cela fonctionne avec `*` mais ne fonctionne pas avec les accolades car l'expansion de `{ }` ne se fait pas dans le cadre d'une assignation de variable :

```
$ MaListe=/home/{riri,fifi}  
$ for Dossier in $MaListe; do echo $Dossier; done  
/home/{riri,fifi}
```

Variables bash et expansion des accolades

- Il est possible de contourner cette limitation. On utilise alors les parenthèses, qui permettent de forcer l'expansion des accolades, et qui transforment MaListe en tableau

```
$MaListe=( /home/{riri,fifi} )  
$for Dossier in ${MaListe[@]}; do echo $Dossier; done  
/home/riri  
/home/fifi
```

Ce qu'on a couvert

- Rappel sur les systèmes Linux et le shell
 - ▣ Les principaux éléments du système
 - SGF
 - User
 - Processus
 - ▣ Les principaux Shell existants
- Rappel sur les commandes
 - ▣ L'invite de commande
 - ▣ Commandes Linux
 - ▣ Mécanismes: échappement, expansion