

tavenel:~/epsi/i1\$ echo -e "Environnement Linux\nInstallation et configuration poste de travail"

ENVIRONNEMENT LINUX

INSTALLATION ET CONFIGURATION

POSTE DE TRAVAIL



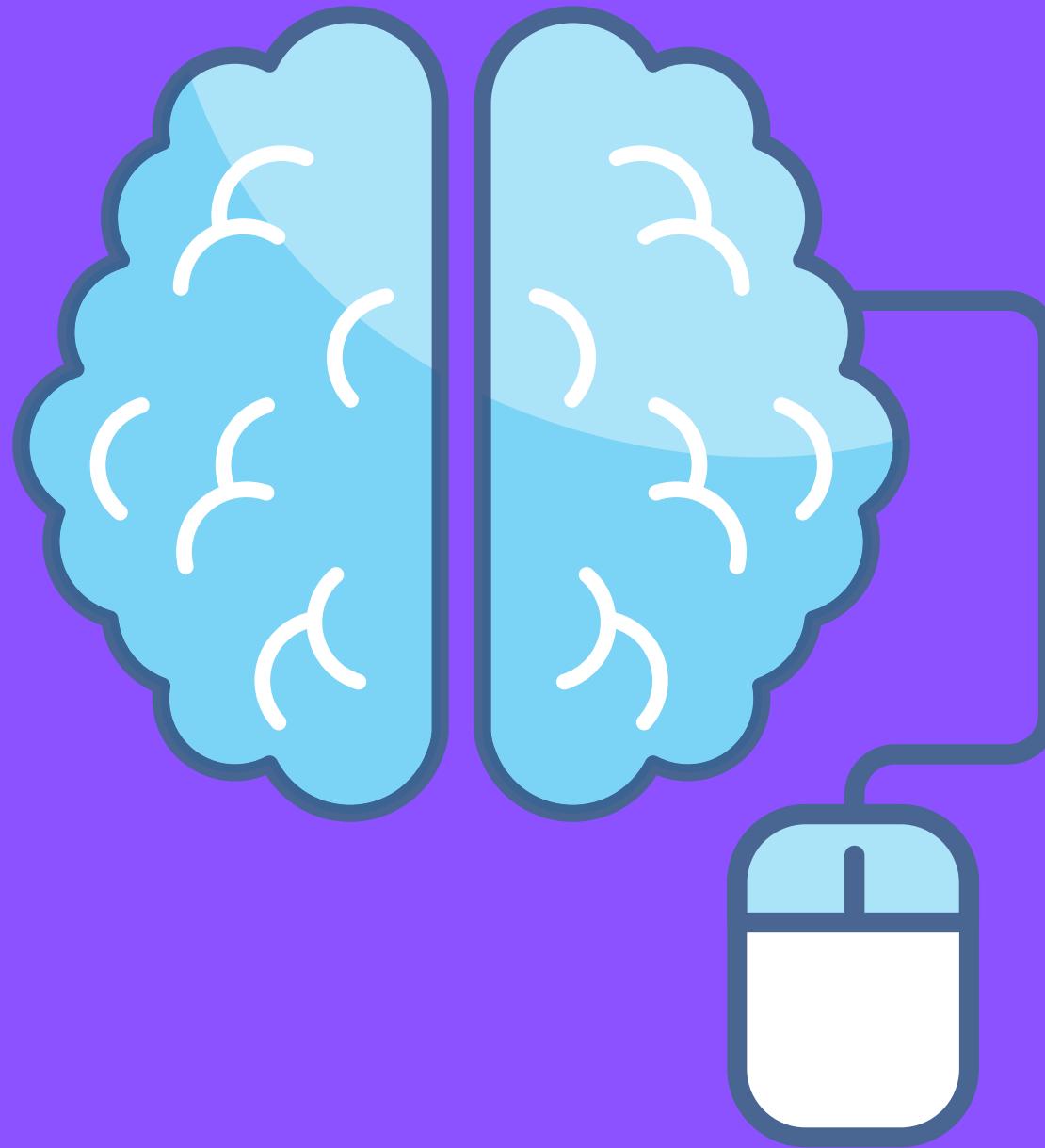
*Tom Avenel
tomavenel@gmail.com*



OBJECTIFS DU MODULE

- Acquérir les notions nécessaires à la compréhension d'un environnement linux

COMPÉTENCES À ACQUÉRIR



- 01 Maitriser les commandes de base de Linux
 - 02 Installer un serveur LAMP afin de faire fonctionner une application PHP/MySQL
 - 03 Administrer un système Linux afin de garantir son bon fonctionnement
 - 04 Gérer les ressources d'un système Linux



PARTIE I

APERÇU D'UN

SYSTÈME LINUX

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

Qu'est-ce que Linux ?

Quelles sont ses caractéristiques principales ?

LE SYSTÈME D'EXPLOITATION LINUX

LE SYSTÈME D'EXPLOITATION



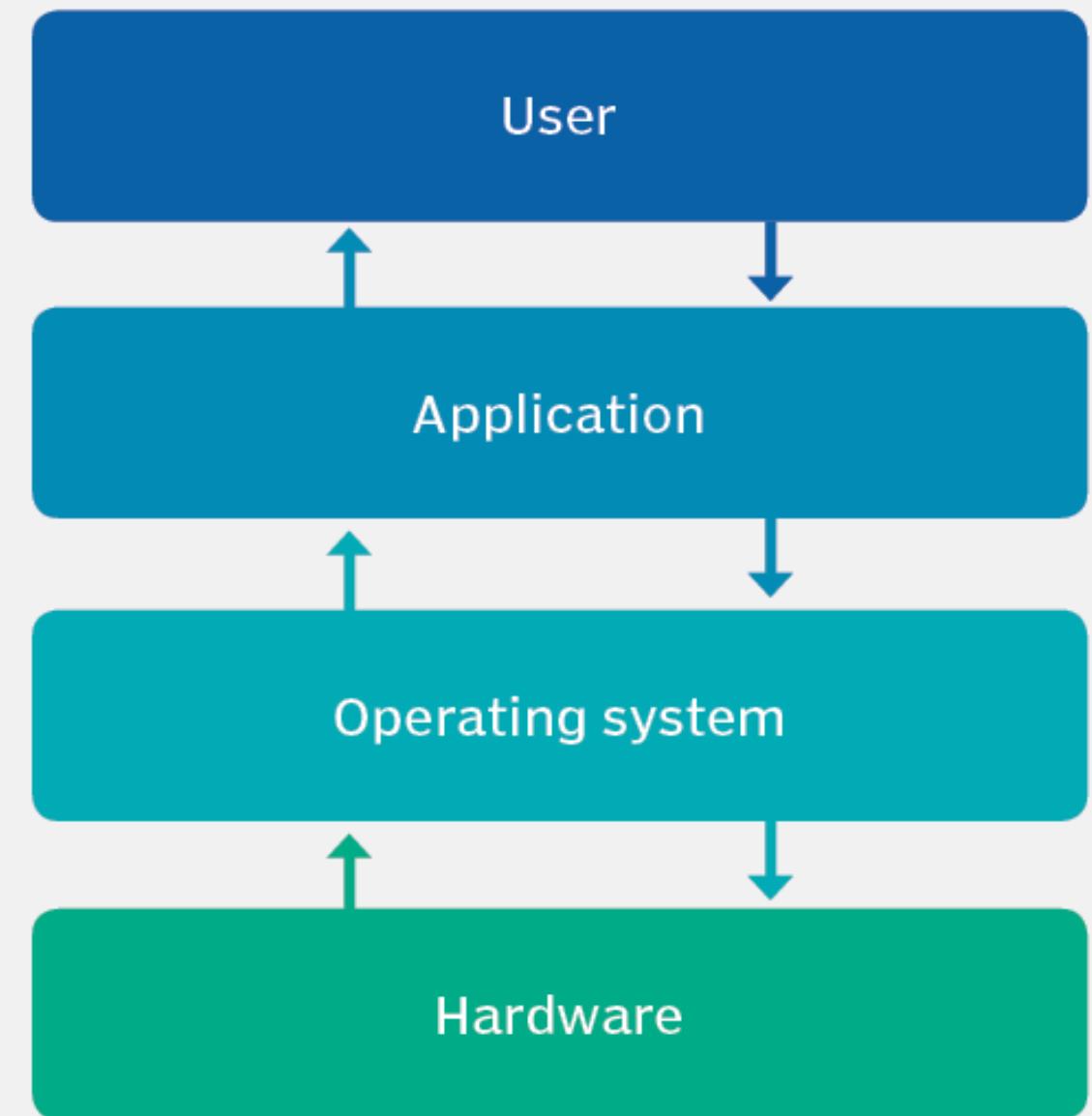
Un système d'exploitation (Operating System - OS) est un logiciel gérant **l'exécution des applications et leurs interactions avec le matériel**.

Les principaux systèmes d'exploitation sur PC sont Windows, Linux et MacOS : ce sont eux qui sont lancés au démarrage d'un ordinateur personnel ou d'un serveur avant toute autre application.



Il existe de nombreux autres OS : Android, iOS, ...

Operating system placement



POURQUOI LINUX ?

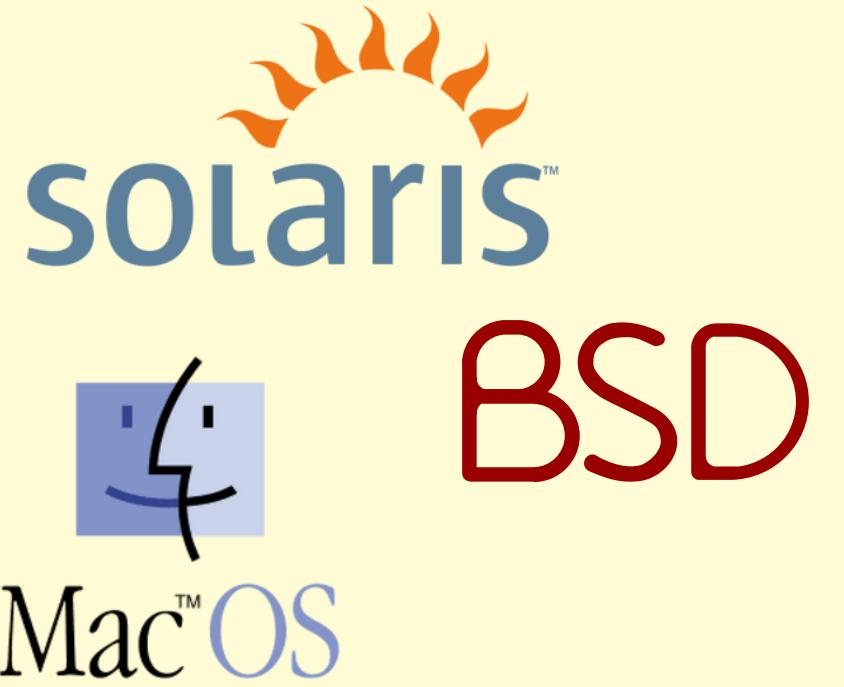
A la fin des années 1960, AT&T Bell Labs développe Unix : un système d'exploitation accessible et sécurisé pour **utilisateurs multiples**.

Dans les années 1980, certaines entreprises commencent à vendre leurs propres OS de type "Unix" : BSD, Solaris, Mac OS X, ...



Ces systèmes connaissent un certain succès mais sont coûteux et peu évolutifs : il manque dans cet écosystème un OS **gratuit** et **libre** (open-source).

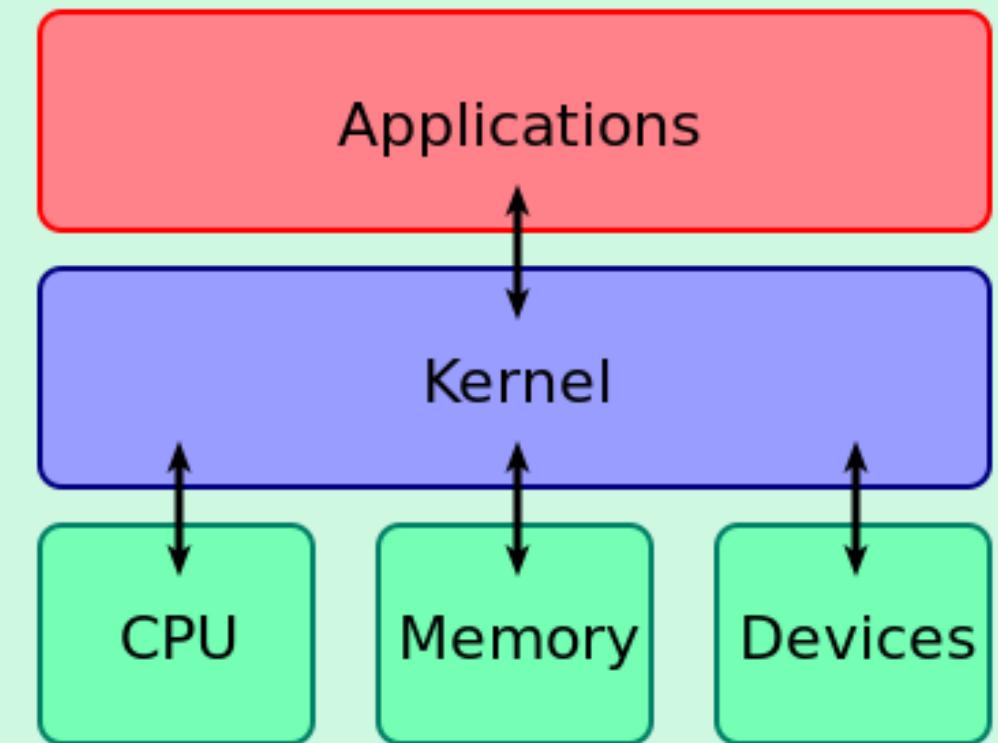
En 1991, Linux Torvalds libère la première version du noyau Linux écrit en langage C et copiant les APIs Unix.



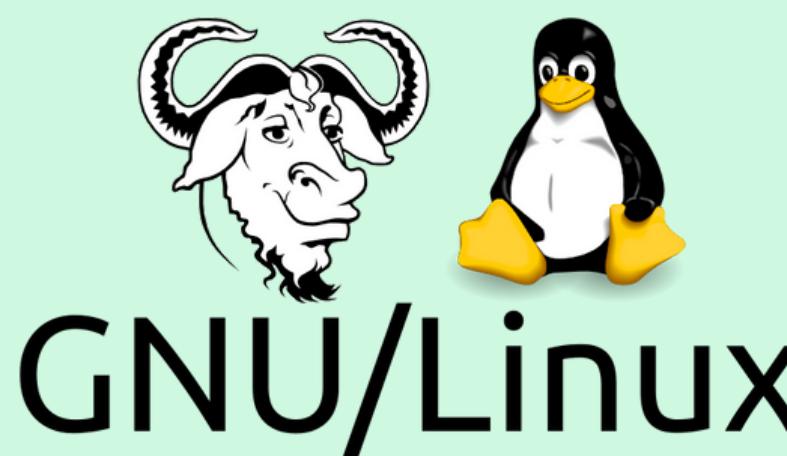
QU'EST-CE QUE LINUX ?

Linux est un **noyau** de système d'exploitation :

- Gratuit tant pour une utilisation personnelle que commerciale
- Libre : le code source est disponible
- Comme tout noyau, il ne gère que les entrées/sorties matérielles (clavier, souris, écran, ...) et l'orchestration des applications (exécution du programme sur des cycles CPU, gestion de la RAM, ...)



Un noyau n'est pas suffisant pour tourner des applications : il faut un système d'exploitation complet avec des librairies, des logiciels, ...



- Au-dessus du noyau Linux, les **distributions** ajoutent des outils, logiciels et librairies open-source (partagés sous le terme **GNU/Linux**) et des outils dédiés (libres ou non-libres)
- Ces distributions patchent souvent le noyau Linux standard pour fournir leur propre version légèrement modifiée

QU'EST-CE QUE LINUX ?



Il existe beaucoup de distributions GNU/Linux : généralistes ou dédiées à un usage particulier (montage vidéo, bureautique, ...), multiplateforme ou dédiées à un environnement particulier, généralement embarqué (dongle TV, raspberry pi, ...), gratuites ou payantes.

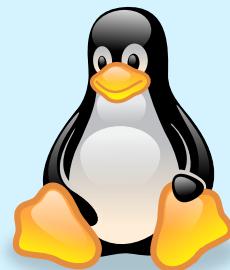
Certaines distributions ont tellement dévié du noyau standard qu'on ne les considère plus vraiment comme des distributions Linux mais elles en gardent les concepts principaux (Android, ...)

 Ces distributions partagent des standards communs les rendant grandement interconnectables et interchangeables : *POSIX*, *System-V*, ... mais ne les respectent que partiellement, chacune ayant ses spécificités. Par exemple, la commande grep possède des options particulières sur presque chaque distribution.

Il existe de nombreux autres OS : Android, iOS, ...

SYSTÈME GNU/LINUX

Un système GNU/Linux est donc un ensemble de plusieurs composants :



Le noyau Linux :

- Responsable des activités principales de l'OS
- Composé de plusieurs modules qui interagissent avec le matériel
- Gère la sécurité (droits d'accès, ...)



Les librairies système :

- Elles fournissent aux applications les APIs des opérations courantes de l'OS : opérations d'entrée/sortie (I/O), droits d'accès, création de processus, ...



Les outils système :

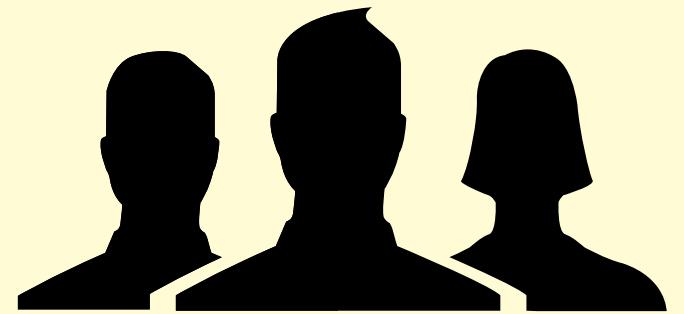
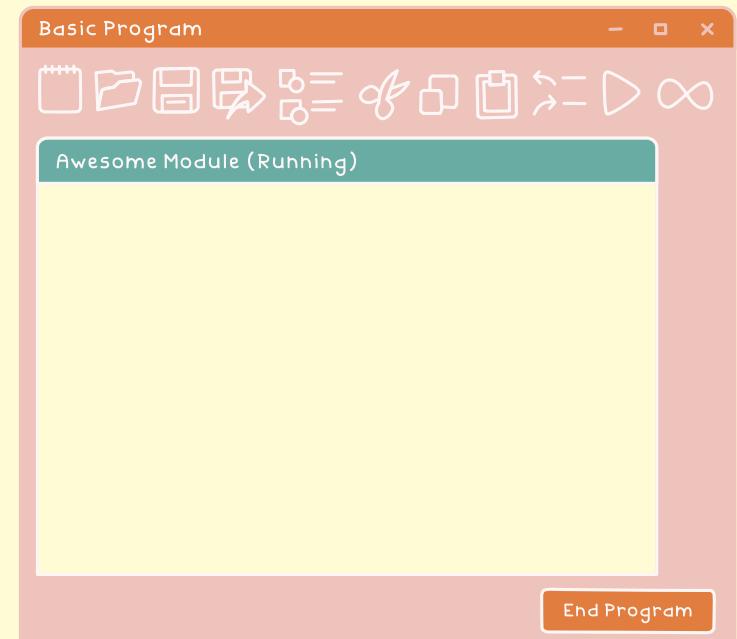
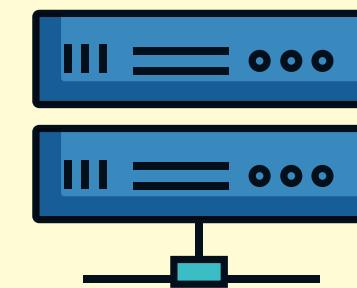
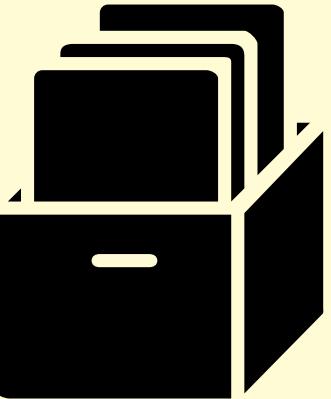
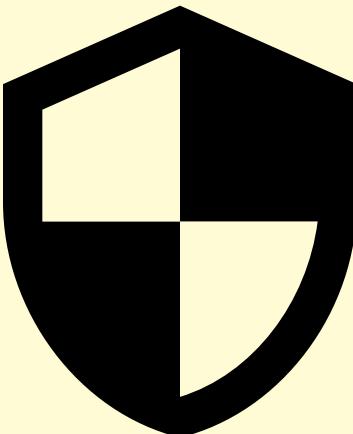
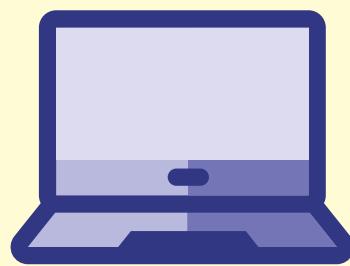
- Ce sont des applications dédiées qui fournissent une gestion de haut-niveau de l'OS en créant une abstraction sur des tâches complexes : gestion du réseau, ...

FONCTIONNALITÉS

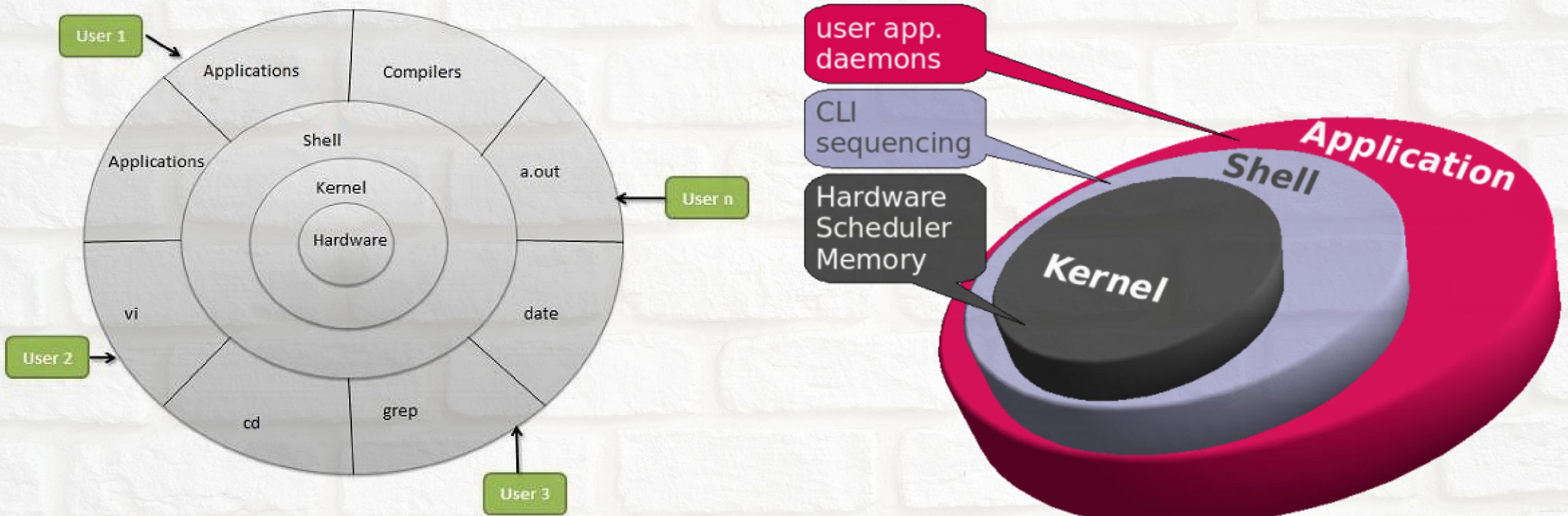


Quelques fonctions principales d'un système Linux :

- Portable
- Open-source
- Multi-utilisateur
- Multi-programmes
- Système de fichiers hiérarchique
- Invité de commandes (shell)
- Sécurité



ARCHITECTURE D'UN SYSTÈME LINUX



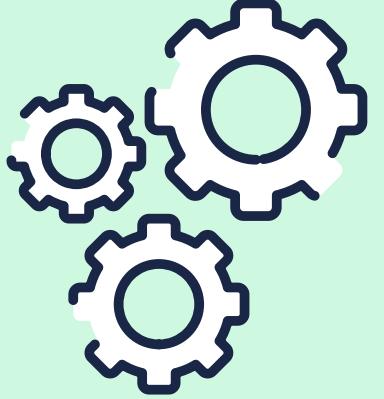
LE NOYAU

Quelques caractéristiques principales du noyau Linux :

- C'est un noyau monolithique (avec des modules chargeables dynamiquement : LKM)
- Supporte le "live patching"
- Le noyau Linux sépare l'environnement d'exécution en deux espaces :
l'espace noyau et l'espace utilisateur
- Focus important sur la sécurité : droits d'accès des utilisateurs, module noyau SELinux pour une gestion très poussée des autorisations

GESTION DES PROCESSUS

LES PROCESSUS



A chaque fois qu'une commande est exécutée ou qu'un programme est lancé, ceux-ci créent un nouveau processus comprenant :

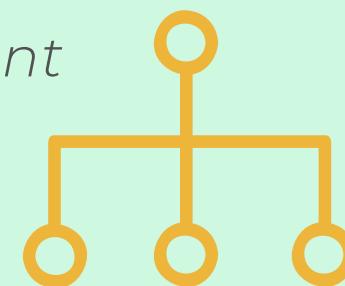
- Un **identifiant unique** à 5 chiffres : PID
- **Tous les services et/ou ressources nécessaires** au processus pendant son exécution (mémoire, accès disque, ...)
- Un **répertoire de travail**

Un processus peut créer des sous-processus (processus fils) :

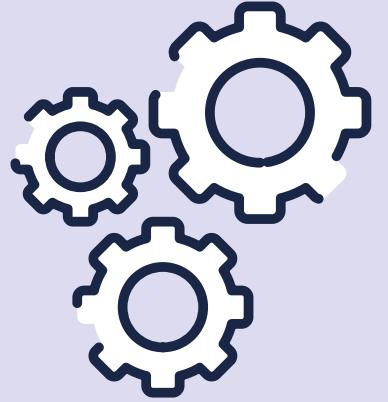
- *L'identifiant de processus parent (**ppid**) identifie le processus ayant créé ce nouveau sous-processus*
- *Tous les processus ont un parent, à l'exception du processus racine (init, pid=1) qui initialise l'espace utilisateur.*



Les processus sont donc hiérarchisés et peuvent être représentés par un arbre dont la racine est le processus "init"



LES PROCESSUS



Dans Linux, (sauf spécificité) un processus est **mono-threadé** : il n'existe pas de threads parallèles d'exécution. Un "programme" (processus) possède donc un seul fil d'exécution.

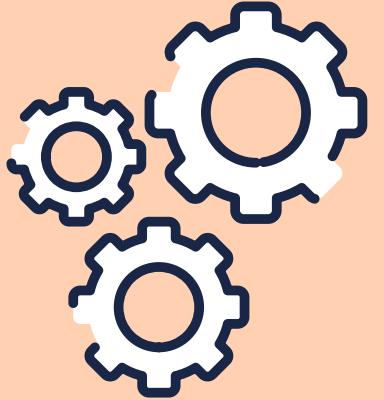
Le noyau offre des APIs pour créer des sous-process depuis le process courant : *fork()*, *vfork()*, *exec()*, *execve()*, ...



Il n'existe pas d'autre moyen de créer un processus !



Un processus appartient à l'utilisateur qui l'a créé



LES TYPES DE PROCESSUS

Il existe 2 types de processus :

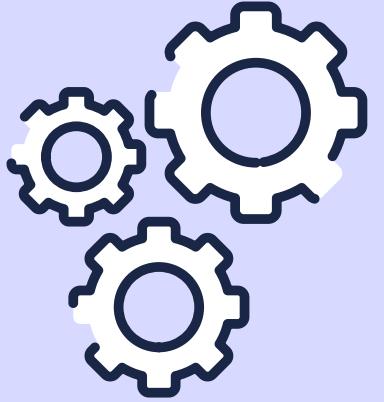
Les processus de **premier plan** (foreground) :

- *Ils sont initialisés et contrôlés à travers une session de terminal (shell)*
- *Ils nécessitent un utilisateur "vrai" pour les démarrer et interagir avec*

Les processus d'**arrière-plan** (background) :

- *Ces processus ne sont pas liés à un terminal*
- *Ils n'attendent aucune interaction utilisateur*

LES TYPES DE PROCESSUS



Le cas particulier des processus daemon

- Un processus *daemon* est un sous-type de processus d'arrière-plan qui fournit des services système
- En général, ils sont lancés au démarrage du système et ne sont jamais arrêtés



Exemples de daemons : serveur Web, service réseau, ...



La gestion des daemons est une des principales incohérences entre les différentes distributions GNU/Linux : SystemD sur Fedora, Upstart sur Ubuntu, SysVinit sur Debian, ...

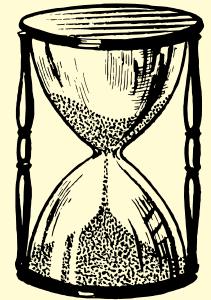
ÉTATS DES PROCESSUS



L'état d'un processus est défini par l'une des quatre valeurs suivantes :

Running

- Le processus est en cours d'exécution sur le CPU...
- ...ou en attente d'un cycle CPU libre pour s'exécuter



Waiting

- Le processus est en attente d'un événement : signal du noyau, condition matérielle (appui sur une touche du clavier, ...)

Stopped

- Le processus est en pause, généralement pour une opération de maintenance



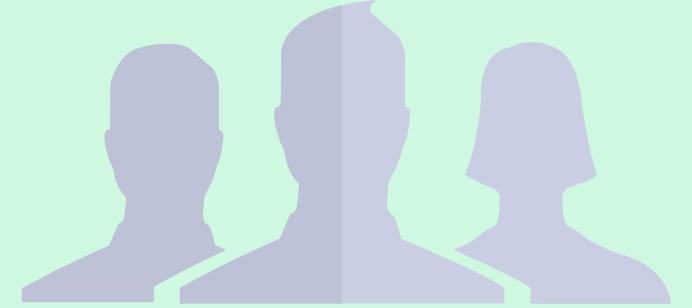
Zombie

- Le processus est mort mais toujours enregistré dans la table des processus (bug)



UTILISATEURS ET ACCÈS SYSTÈME

ADMINISTRATION DES UTILISATEURS



Linux identifie les utilisateurs par un identifiant unique : User ID (**UID**)

- L'**UID=0** est un utilisateur spécial appelé "**super-utilisateur**" (ou **root**). Cet utilisateur peut outrepasser toutes les vérifications de droits d'accès (dans l'espace utilisateur).
- Le processus *init*, les threads du noyau et la plupart des processus système appartiennent à l'utilisateur root



Pour faciliter les vérifications de sécurité, les **rôles** des utilisateurs sont réunis en groupes identifiés par un ID de groupe (**GID**). Un utilisateur peut faire partie de plusieurs groupes.



Linux est un système multi-utilisateur : **plusieurs utilisateurs** peuvent utiliser le système **en parallèle**.

LES FICHIERS

GESTION DES FICHIERS



En Linux,
TOUT EST FICHIER !



© 20TH CENTURY FOX AND MARVEL ENTERTAINMENT

GESTION DES FICHIERS



Linux utilise des fichiers pour décrire chaque partie du système : matériel, processus, documents, programmes, ...



Caractéristiques des noms de fichiers :

- Les fichiers portent souvent une extension en suffixe pour faciliter leur identification (*mon_document.pdf*, ...). Cette extension est purement descriptive (aucun impact sur le fichier lui-même).
- Linux supporte mal les espaces et accents dans les noms de fichiers
- En Linux, tous les noms de fichier et tous les chemins sont sensibles à la casse ! (majuscule / minuscule)
- Par convention, un nom de fichier commençant par un point est un fichier caché
 - *mon_programme_visible.bin* <-- fichier visible
 - *.ma_config_cachee.bin* <-- fichier caché



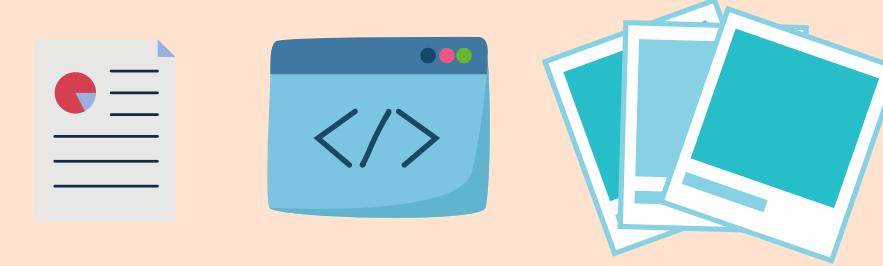
LES TYPES DE FICHIERS



Linux utilise plusieurs types de fichiers différents :

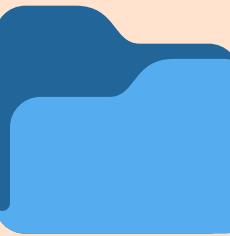
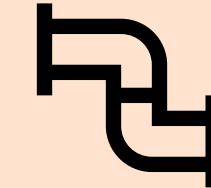
Fichier ordinaire

C'est le type de fichier le plus courant : fichier texte, image, binaire, ...



Lien symbolique

Un simple alias vers un autre fichier



Répertoire

Un répertoire est aussi un fichier !

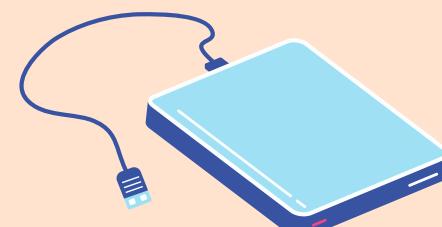
Fichier de socket locale / pipe nommé

Ces fichiers permettent la communication entre processus

Fichier de périphérique en mode caractère / fichier de périphérique en mode bloc

Ces fichiers permettent la communication avec les différents périphériques :

- *En mode caractère, ils permettent un accès matériel direct, caractère par caractère*
- *En mode bloc : sont les disques durs, la mémoire, ... qui utilisent des accès par tampons*
- *Un périphérique peut être virtuel : /dev/random, /dev/null, ...*



HIÉRARCHIE ET CHEMINS D'ACCÈS



Les fichiers sont stockés dans des **répertoires**, organisés selon un **système de fichiers hiérarchique** et accessibles en utilisant un chemin (**path**).

Le path décrit la progression dans des répertoires séparés par un / jusqu'au fichier final, de façon similaire à une URL dans un navigateur.

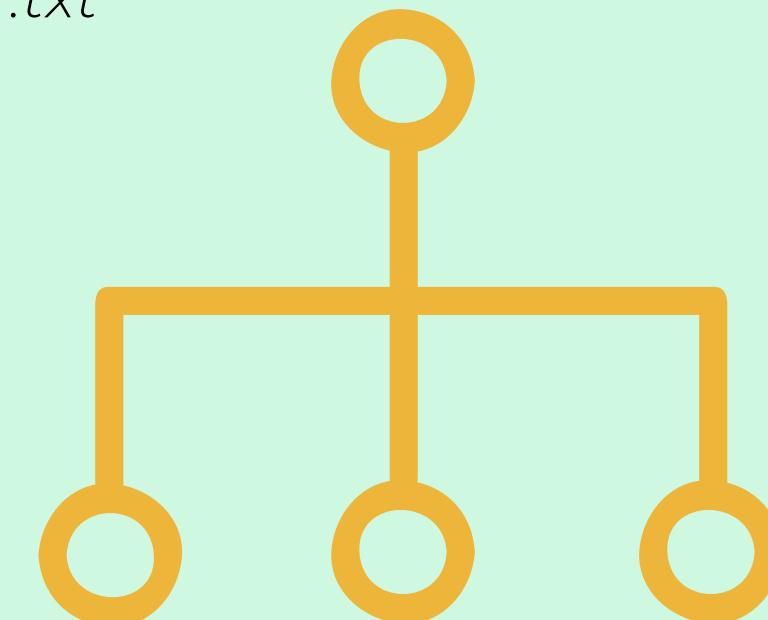
Un path peut décrire un chemin de deux façons, suivant qu'il commence ou non par un /

→ Soit **relatif** au répertoire courant

Exemple : *mon_sous_repertoire_dans_le_repertoire_courant/mon_fichier.txt*

→ Soit **absolu** en partant du répertoire à la racine, noté /

Exemple : */mon_repertoire_sous_la_racine/mon_fichier.txt*



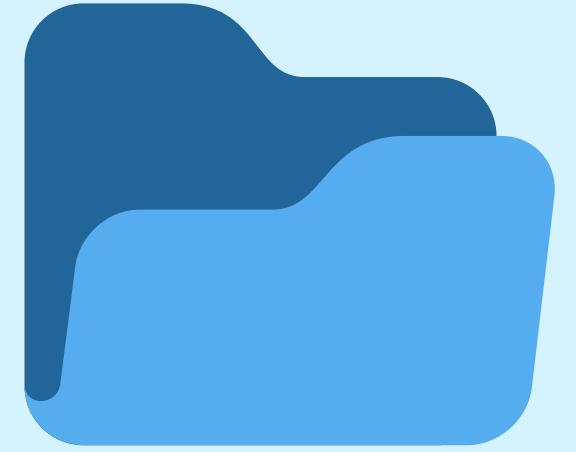
CHEMINS PARTICULIERS



Il existe des chemins particuliers :

- Le path nommé d'un simple slash **/** fait référence au répertoire à la racine du système de fichiers. C'est ce répertoire qui contient les répertoires de 1er niveaux, contenant eux-mêmes d'autres répertoires, et ainsi de suite
- Le path nommé d'un simple point **.** fait référence au répertoire courant. Les deux chemins ci-dessous sont donc identiques :
 - `mon_sous_repertoire/mon_fichier.txt`
 - `./mon_sous_repertoire/mon_fichier.txt`
- Le path nommé de deux simple points **..** fait référence au répertoire parent. Les deux chemins ci-dessous sont donc identiques :
 - `/mon_premier_repertoire/mon_sous_repertoire/mon_fichier.txt`
 - `/mon_premier_repertoire/mon_sous_repertoire/./mon_sous_repertoire/mon_fichier.txt`
- Le path nommé tilda **~** fait référence au répertoire de travail de l'utilisateur courant.

RÉPERTOIRE UTILISATEUR



 Comme dans la majorité des systèmes d'exploitation, chaque utilisateur possède **son propre répertoire de travail**, appelé **home directory**. C'est dans ce répertoire que sont stockés ses documents personnels, ses configurations propres, et c'est dans ce répertoire que l'utilisateur atterrit après s'être connecté au système.

- Le répertoire de travail du super-utilisateur (root) est généralement **/root**
- Les répertoires de travail (home) des utilisateurs standards sont en général situés dans **/home/NOM_DE_L'UTILISATEUR**
- Le path nommé tilda **~** fait référence au répertoire de travail de l'utilisateur courant. Si l'utilisateur `utilisateur1` est connecté au système, ces chemins sont donc identiques :
 - `/home/utilisateur1/mon_dossier_utilisateur`
 - `~/mon_dossier_utilisateur`

 Les répertoires de travail des utilisateurs (home) **sont accessibles uniquement à leurs utilisateurs respectifs** (à l'exception du super-utilisateur root ayant les pleins pouvoirs sur le système).

LES OUTILS

LINUX

LE SHELL



Les systèmes GNU/Linux sont des systèmes fortement orientés à l'**utilisation principale d'interfaces texte**.

Le shell (ou interpréteur de commandes) est le point d'entrée principal pour gérer un système Linux. Son utilisation se fait en utilisant des entrées (commandes) et des sorties de type texte.



*Il existe de nombreux interpréteurs de commandes différents et partiellement compatibles. La plupart d'entre eux suit cependant les standards POSIX partagés par tous les systèmes *NIX (BSD, Linux, Mac OS, Android, ...)*



Bash (ou sh) est de très loin l'interpréteur les plus utilisé et ce cours se concentrera sur son usage.



LE SHELL



Même si un shell utilise en général un langage de script complet (utilisation de fonctions, de variables, ...) son utilisation est principalement **interactive**. Les commandes sont entrées **l'une après l'autre** : l'utilisateur entre une commande complète (avec ses arguments) sur l'invité de commande (prompt), puis démarre l'exécution de cette commande en appuyant sur la touche entrée.

L'invité de commandes (prompt)

The screenshot shows a terminal window with a dark background and light-colored text. At the top, the prompt is "wiki@ubuntu: ~/Desktop/text". Below it, the user has entered the command "cd text" followed by the output of the "ls" command, which lists "sample.txt". The user then enters "cat > sample.txt" and the terminal displays the text "This is the sample text file created in Linux terminal. by wikihow.com".

Annotations:

- A green oval highlights the prompt "wiki@ubuntu: ~/Desktop/text". A black arrow points from the text "L'invité de commandes (prompt)" to this oval.
- A blue oval highlights the command "cd text" and its output "ls". A purple arrow points from the text "La commande avec ses arguments entrés par l'utilisateur" to this oval.
- A red oval highlights the entire bottom line of text, including the command "cat > sample.txt" and its output. A red arrow points from the text "Les informations renvoyées par la commande" to this oval.

```
wiki@ubuntu: ~/Desktop/text
wiki@ubuntu:~/Desktop$ cd text
wiki@ubuntu:~/Desktop/text$ ls
sample.txt
wiki@ubuntu:~/Desktop/text$ cat > sample.txt
This is the sample text file created in Linux terminal.
by wikihow.com
```

Les informations renvoyées par la commande

LE SHELL



*Linux étant un système fortement orienté fichiers, il est possible d'utiliser un fichier pour enregistrer la liste des commandes à lancer. Un tel fichier, contenant la liste des commandes qui seront exécutées dans le shell, est appelé un **script**.*



Les bonnes pratiques recommandent d'ajouter un "shebang" au début de chaque script pour forcer l'utilisation du bon interpréteur de commandes (bash dans ce cours) plutôt que celui par défaut défini dans le système.
On ajoutera donc la ligne : #!/bin/bash



Les postes de travail récents possèdent généralement un véritable écran plutôt qu'un terminal : l'exécution du shell est effectuée à travers des interfaces virtuelles (appelées TTY) ou grâce à une application appelée émulateur de terminal.

LES ARGUMENTS DES COMMANDES

Les commandes à exécuter nécessitent souvent des arguments à utiliser comme paramètres lors de leur exécution (nom de l'utilisateur, chemin vers le fichier, adresse IP, ...).

Les arguments sont fournis à la suite du nom de la commande, et séparés par un espace.

Par exemple : `$ nomDeMaCommande argument1 argument2 argument3`



Pour simplifier l'utilisation des nombreuses commandes, le shell bash fournit une grammaire stricte et commune pour la documentation des arguments des commandes décrivant :

- Un argument obligatoire : `$ nomDeMaCommande argumentObligatoire`
- Un argument optionnel : `$ nomDeMaCommande [argumentOptionnel]`
- Un choix entre 2 arguments incompatibles : `$ nomDeMaCommande argumentA / argumentB`
- Un argument pouvant être répété plusieurs fois : `$ nomDeMaCommande argumentMultiple...`



En shell bash, le caractère `#` indique d'ignorer le reste de la ligne.

Ce caractère est utile pour ajouter des commentaires dans un script, par exemple :

`maCommande monArgument # Ceci est un commentaire`

LE PROMPT DU SHELL

 Par convention, la documentation d'une commande à exécuter dans le shell est précédée du caractère `$` afin de la mettre en évidence. Ce caractère représente le **prompt** par défaut, c'est-à-dire l'endroit où l'utilisateur entre sa commande.

Par exemple, si un document indique d'entrer la commande `$ maCommande argument1`, on tapera sur le clavier seulement la partie `maCommande argument1` (sans le signe `$`).



 En Linux, il est courant d'exécuter des processus en tant que super-utilisateur (root) pour effectuer des opérations de maintenance sur le système (installation d'application, ...)

Par convention, les commandes à exécuter par un utilisateur standard sont préfixées par le symbole `$` alors que celles à exécuter par le super-utilisateur root sont préfixées par un `#`.

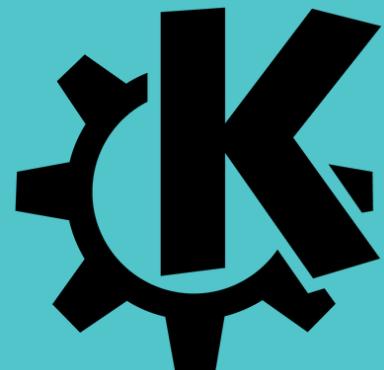
Par exemple, la commande suivante, exécutée par un utilisateur standard, liste les processus lui appartenant : `$ ps -ef`

La même commande, exécutée par le super-utilisateur (avec plus de droits) liste l'ensemble des processus lancés par tous les utilisateurs sur le système : `# ps -ef`

L'ENVIRONNEMENT GRAPHIQUE

 Si ils sont orientés interface texte en priorité, les systèmes Unix supportent aussi les interfaces graphiques :

- Le système X (ou X11) est un système de gestion de fenêtres très populaire
- Wayland sera le nouveau standard d'environnement graphique sur Linux. Certaines distributions le supportent déjà (Ubuntu, Fedora)
- Quartz sur Mac OS, SurfaceFlinger sur Android, ...
- Un gestionnaire de bureau (séparé) est généralement exécuté en plus, au-dessus du simple système d'affichage des fenêtres lorsque le système GNU/Linux est utilisé comme un ordinateur personnel (Gnome, KDE, ...)



LES ÉDITEURS DE TEXTE



La plupart des distributions GNU/Linux sont également livrées avec un ou plusieurs éditeurs de texte en mode console (pouvant être exécutés dans un terminal) :

- *vi* (ou sa version améliorée *vim*) est un éditeur extrêmement puissant mais plutôt complexe à prendre en main, disponible dans presque toutes les distributions
- *emacs*, *nano*, ... sont d'autres exemples souvent disponibles



Qu'est-ce qu'un fichier sous Linux?
Quelle est leur utilité?
Comment les administrer?

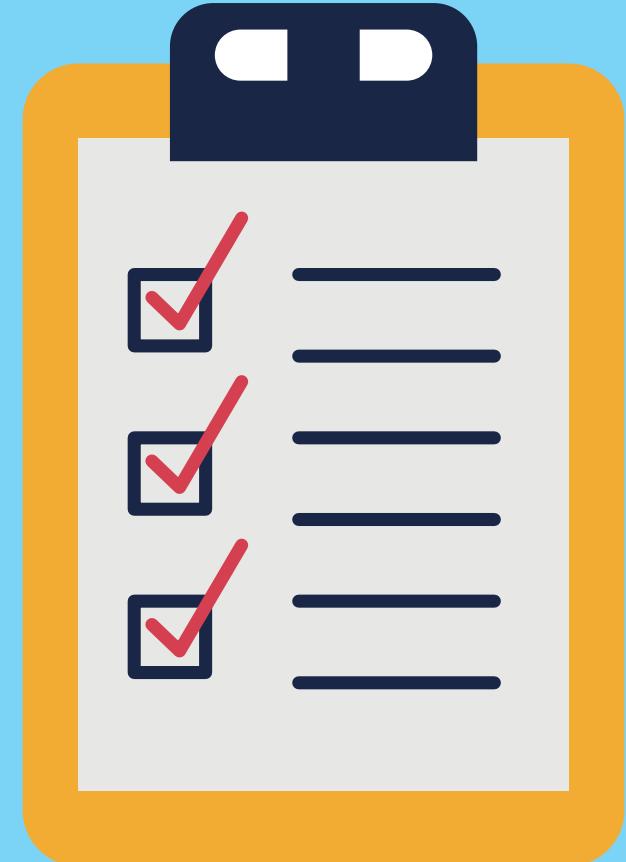
PARTIE II > -

COMMANDES PRINCIPALES

LES COMMANDES DE GESTION DE FICHIERS

LS : LISTER LES FICHIERS

La commande **ls** (list) affiche tous les répertoires et fichiers principaux dans le chemin fourni en paramètre (ou dans le répertoire courant sans paramètre).



Par exemple, la commande suivante affiche tous les fichiers et dossiers stockés dans le répertoire applications à la racine :

```
$ ls /applications
```

CD : CHANGER DE RÉPERTOIRE

La commande **cd** (change directory) permet de changer le répertoire courant vers un nouveau répertoire



Par exemple, la commande suivante permet de se déplacer dans le dossier *monDossierPerso* stocké dans le répertoire utilisateur. Les commandes commandes seront maintenant exécutées dans ce nouveau répertoires.

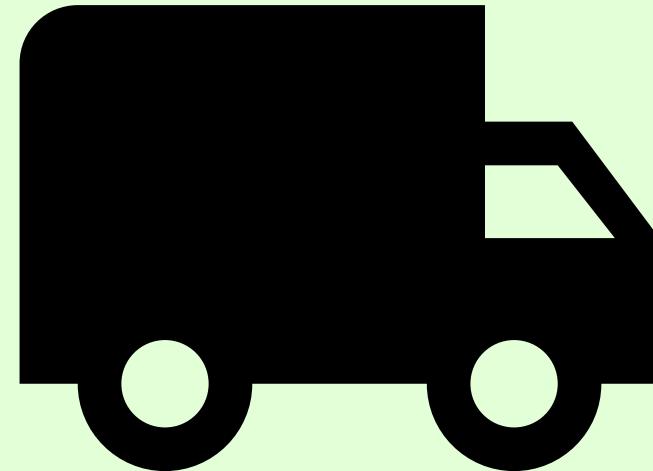
```
$ cd ~/monDossierPerso
```

MV : DÉPLACER UN FICHIER

La commande **mv (move) permet de déplacer un fichier de son emplacement courant vers un nouveau chemin.**

Si le nouveau chemin est celui d'un répertoire existant, le fichier sera déplacé dans celui-ci.

Sinon, le fichier sera déplacé et/ou renommé pour que son nouveau chemin soit celui fourni.

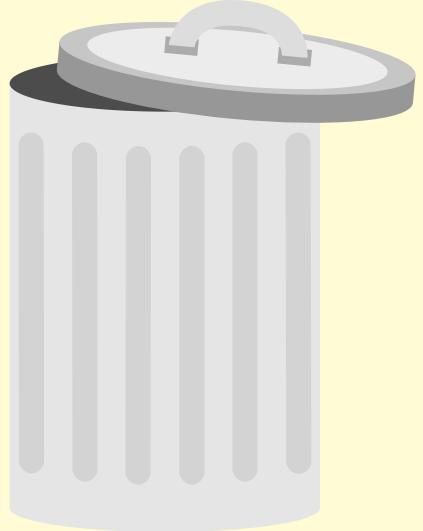


Par exemple, la commande suivante permet de déplacer le fichier *monFichier* depuis le répertoire courant vers un répertoire voisin et de renommer le fichier en *monNouveauFichier* :

```
$ mv ./monFichier ../monReperoireVoisin/monNouveauFichier
```

RM : SUPPRIMER DES FICHIERS

La commande **rm** (remove) permet de supprimer les fichiers dont le chemin est donné en paramètre.



Par exemple, la commande suivante supprime un fichier *monFichier* dans le répertoire courant :

```
$ rm monFichier
```

TOUCH : ATTEINDRE UN FICHIER

La commande **touch** permet d'atteindre le fichier fourni en paramètre. Cela met à jour sa date d'accès, et c'est également un moyen de créer un nouveau fichier vide.

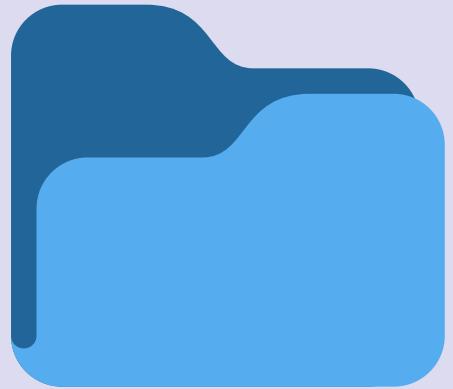


Par exemple, la commande suivante crée un nouveau fichier dans le répertoire utilisateur :

```
$ touch ~/monNouveauFichier
```

MKDIR / RMDIR : CRÉER / SUPPRIMER UN RÉPERTOIRE

Les commandes **mkdir** (make directory) et **rmdir** (remove directory) permettent créer un dossier vide et de supprimer un dossier vide.



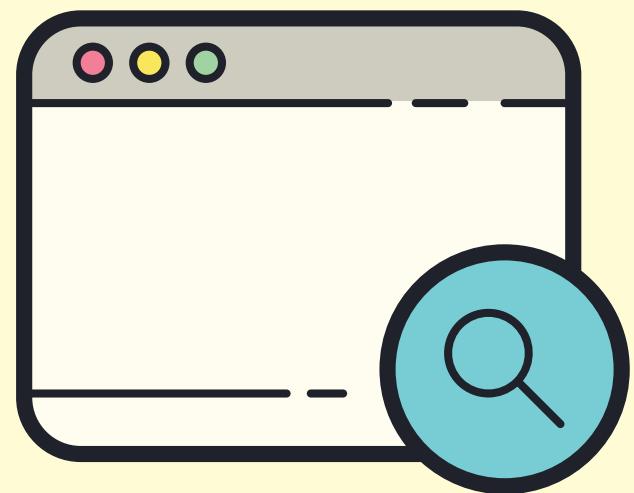
Par exemple, la commande suivante crée un nouveau dossier vide *monDossier* à la racine. Comme ce dossier est vide, on peut ensuite le supprimer :

```
$ mkdir /monDossier  
$ rmdir /monDossier
```

Un dossier est un fichier - il est donc possible d'utiliser la commande *rm* à la place de *rmdir*. On préfèrera cependant la commande *rmdir* qui vérifie que le dossier est vide avant sa suppression.

FIND : CHERCHER DES FICHIERS

La commande **find** permet de chercher le chemin vers des fichiers dont le nom suit un pattern donné dans un chemin donné (récursevement).



Par exemple, la commande suivante cherche le chemin du fichier *monFichier*, situé quelque part dans l'arborescence commençant au répertoire *monRépertoire* :

```
$ find /home/monUtilisateur/monRépertoire -name monFichier
```

 La commande **find** est très puissante et possède de nombreux paramètres, il est intéressant de pratiquer son utilisation qui s'avère vite très pratique.

CAT : AFFICHER LE CONTENU D'UN FICHIER

La commande **cat** (concatenate) permet de concaténer des fichiers vers la sortie standard.

Dans la pratique, cette commande est souvent utilisée pour afficher à l'écran le contenu d'un fichier.

Par exemple, la commande suivante affiche à l'écran le contenu du fichier monFichier stocké dans le répertoire courant :

```
$ cat ./monFichier
```

MORE, LESS, HEAD, TAIL : AFFICHAGES PARTIELS

Les commandes **more**, **less**, **head** et **tail** permettent de couper ou de paginer un contenu qui leur est fourni en entrée avant d'afficher ce contenu à l'écran.

Dans la pratique, ces commandes sont très utiles lorsque l'on enchaîne des commandes en utilisant des pipes (nous verrons cette notion plus tard dans le cours).



Par exemple, les commandes suivantes affichent à l'écran le début et la fin du contenu du fichier monFichier stocké dans le répertoire courant :

```
$ head ./monFichier
```

```
$ tail ./monFichier
```

**AUTRES
COMMANDES
UTILES**

MAN : LA PAGE DE MANUEL

La commande **man** (**manual**) permet d'afficher la documentation d'une commande ou d'un service fourni en paramètre.

C'est une commande fondamentale d'un système Linux : toutes les commandes du système sont rigoureusement documentées.

Les pages de manuel sont vérifiées en profondeur, elles sont à jour avec la version du programme installé sur la machine et suivent une grammaire de documentation commune stricte. Elles peuvent être délicates à appréhender au début mais il est important de comprendre comment les utiliser et de s'entraîner à les lire.



Par exemple, la commande suivante affiche la page de manuel de la commande `find` :

```
$ man find
```

AUTRES COMMANDES UTILES

echo : affiche un message sur la sortie standard (par défaut la console)

```
$ echo "Hello, World"
```



clear : efface la sortie du terminal

```
$ clear
```

alias : crée une nouvelle commande qui, lorsqu'elle est appelée, exécutera l'instruction enregistrée

```
$ alias maCommandeLs='ls -a'
```

```
$ maCommandeLs
```

AUTRES COMMANDES UTILES

history : affiche l'historique des commandes entrées par l'utilisateur

```
$ history
```



grep : cherche une ligne suivant un pattern donné dans l'entrée de la commande

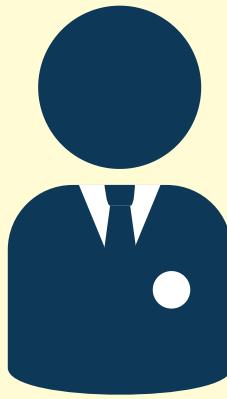
```
$ grep monPattern monDossier/monFichier
```

sudo : exécute une commande en utilisant le rôle de super-utilisateur (root).

```
$ sudo rm /monDossierProtege
```

```
# rm /monDossierProtege
```

Les deux commandes précédentes font la même chose mais on utilisera toujours *sudo* lorsque c'est possible, car cette commande enregistre le changement de contexte (le passage en super-utilisateur), et limite ce contexte à une commande. La 2e commande nécessite de se logger en utilisateur root sur le système pour une session entière.



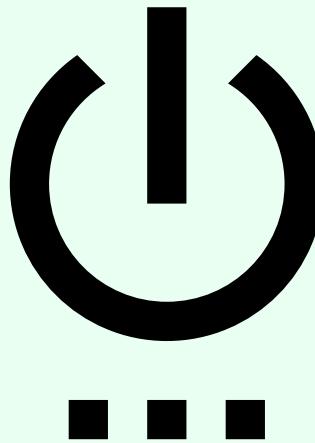
AUTRES COMMANDES UTILES

shutdown, halt, reboot : gèrent le cycle de vie du système

\$ *halt*

\$ *reboot*

\$ *shutdown -h now*



ping : appelle (ping) un système pour vérifier si cet hôte peut être joint

\$ *ping www.google.fr*



traceroute : affiche la route (avec la liste des points d'accès) à traverser avant d'atteindre une destination

\$ *traceroute www.google.fr*

netstat : affiche différentes informations réseau, comme les ports ouverts et les tables de routage :

\$ *netstat -rn*



HACKER LE SHELL

Enchaîner les commandes
Rediriger les entrées/sorties
Filtres shell
Variables

LES FILTRES DU SHELL



HACKER LE SHELL

Le shell fournit des filtres permettant de réaliser des opérations avancées sur les commandes. Ces filtres s'utilisent principalement pour omettre de préciser un ou plusieurs caractères dans un nom de fichier.

Le caractère **?** représente **n'importe quel caractère 1 fois**

```
$ ls monFichier?.txt
```

```
monFichier1.txt monFichier2.txt
```

Le caractère ***** représente **n'importe quel caractère 0 ou plusieurs fois**

```
$ ls *.txt
```

```
monFichier1.txt monFichier2.txt
```

LINUX

LES VARIABLES DU SHELL

Il est possible de définir ou modifier une variable dans le shell courant (et uniquement dans celui-ci) en utilisant la syntaxe :

variable=valeur

La nouvelle variable sera accessible en utilisant l'expression : `$variable` (en ajoutant bien le signe \$ avant la variable cette fois).

```
$ maVariable=2  
$ echo "Ma variable est : $maVariable"  
Ma variable est : 2
```

LES VARIABLES DU SHELL

- 💡 Il est possible d'exporter une variable en dehors du shell courant pour l'utiliser dans l'environnement global en utilisant la commande **export** et le nom de la variable (sans signe \$ devant)

\$ export maVariable



Certaines variables spéciales prédéfinies gèrent l'exécution des commandes :

\$# indique le nombre d'arguments fournis à une commande

\$1, \$2, ... sont chacun des arguments pris séparément

\$ ou \$@ est la liste des arguments d'une commande*

\$? est le code de sortie d'une commande (0 en absence d'échec)

CARACTÈRES D'ÉCHAPPEMENT

Pour éviter d'évaluer un filtre, il est possible de l'entourer de caractères d'échappement. Cela permet d'utiliser un caractère spécial (*, ?, \$) comme un caractère standard dans un nom.

Le guillemet simple '' permet une protection forte d'une chaîne de caractères : aucun caractère à l'intérieur ne sera évalué : la chaîne de caractères est utilisée telle qu'elle

```
$ ls '*'
```

*: no such file or directory

CARACTÈRES D'ÉCHAPPEMENT

 Cet échappement est très utile par exemple avec la commande `find` : cela permet de passer des patterns de noms de fichiers avec des caractères spéciaux sans les évaluer.

```
$ find ~ -name *.txt
```

Cette commande va remplacer `*.txt` par les noms de tous les fichiers `.txt` dans le répertoire courant (par exemple `fichier1.txt` et `fichier2.txt`), avant d'appeler la commande `find`. La commande réellement exécutée sera :

```
$ find . -name fichier1.txt fichier2.txt
```

```
$ find ~ -name '*.*'
```

Cette commande ne va pas remplacer `*.txt`. La commande réellement exécutée sera

```
$ find . -name *.txt
```

Les échappements sont souvent utilisés dans les scripts, par exemple pour stocker des commandes avec des caractères spéciaux dans des variables et les évaluer plus tard.

CARACTÈRES D'ÉCHAPPEMENT

Le guillemet double " " permet une protection faible d'une chaîne de caractères. Son intérêt principal est d'utiliser plusieurs mots (avec des espaces) comme un seul argument. Les caractères spéciaux sont cependant évalués.

```
$ echo Le contenu de argument1 est $1
```

6 arguments => [Le, contenu, de, argument1, est, maValeur]

```
$ echo "Le contenu de argument1 est $1"
```

1 arguments => Le contenu de argument1 est maValeur

```
$ echo 'Le contenu de argument1 est $1'
```

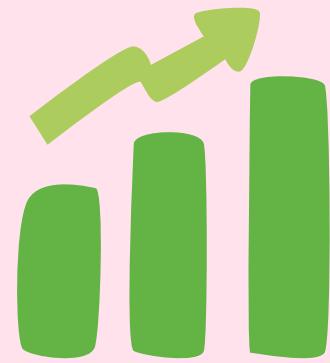
1 arguments => Le contenu de argument1 est \$1

ENCHAÎNEMENT DE COMMANDES

Il est possible d'enchaîner 2 commandes à la suite sur une même ligne en utilisant un ;

```
$ commande1; commande2
```

- 1
- 2

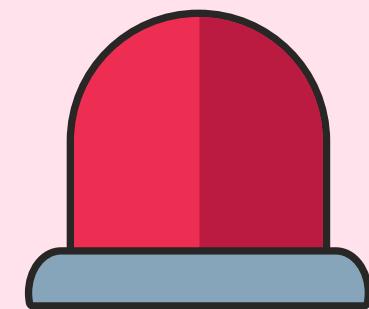


Il est possible d'exécuter une commande, puis d'enchaîner avec la seconde seulement si la première n'a pas échoué (utile si la 2e commande nécessite un état particulier après la 1ère) :

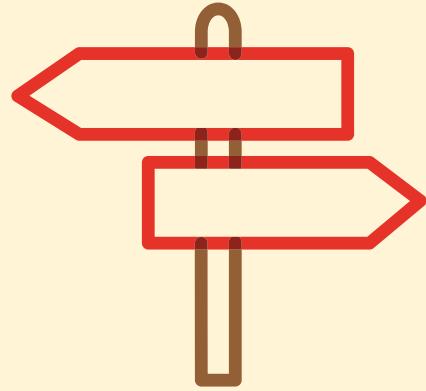
```
$ commande1 && commande2
```

Il est aussi possible d'exécuter une commande, puis d'enchaîner avec la seconde seulement en cas d'échec dans la première (utile pour faire de la gestion d'erreur après la 1ère commande) :

```
$ commande1 || commande2
```



REDIRECTIONS



Tout processus Linux possède des streams qu'il utilise pour dialoguer avec le reste du système. Ces streams possèdent tous une implémentation par défaut :

- **L'entrée standard (STDIN)**

- C'est la stream fournissant les données à traiter à la commande
- Par défaut, il s'agit des entrées de l'utilisateur (le clavier)

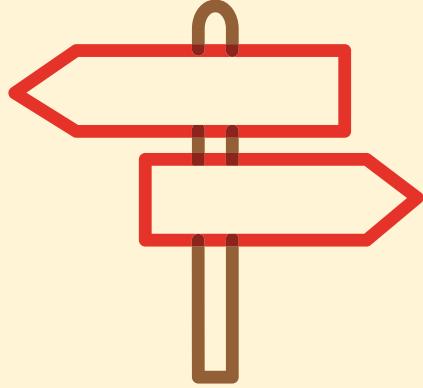
- **La sortie standard (STDOUT)**

- C'est la stream sur laquelle la commande va renvoyer les données après traitement
- Par défaut, il s'agit de la console, permettant d'afficher ces données à l'utilisateur dans le terminal

- **La sortie d'erreur (STDERR)**

- C'est une stream utilisée pour afficher les éventuelles erreurs rencontrées à l'exécution
- Par défaut, cette sortie est fusionnée avec la sortie standard sur la console

REDIRECTIONS

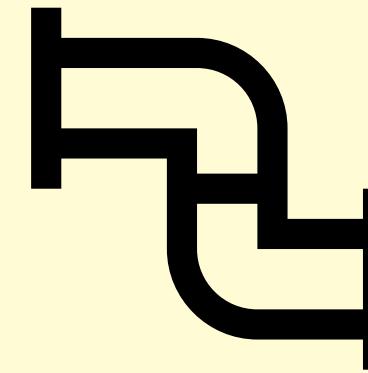


 Ces streams peuvent être redirigées : c'est l'une des fonctionnalités les plus puissantes du shell !

Il est possible de rediriger :

- La sortie standard vers un nouveau fichier grâce à l'expression 1> ou directement >
 - \$ maCommande > monFichierDesResultats
 - Attention : le contenu du fichier est écrasé si ce fichier existait déjà !
- La sortie d'erreur vers un nouveau fichier grâce à l'expression 2>
 - \$ maCommande 2> monFichierDesErreurs
- Il est possible de fusionner deux streams grâce à l'expression &
 - \$ maCommande 2>&1 monFichierDesResultatsEtErreurs
- Il est possible de remplacer le caractère > par un caractère >> pour ne pas écraser le fichier
 - \$ maCommande 2>> monFichierDesErreursCombineesSurPlusieursExecutions

REDIRECTIONS



Une autre fonction de redirection très utile est le pipe, noté | : il permet de rediriger directement la sortie standard d'une première commande dans l'entrée de la seconde. C'est une fonctionnalité très utile pour enchaîner des commandes... comme si l'on utilisait réellement un pipe !

- *\$ maPremiereCommande | maDeuxiemeCommandeUtilisantLesResultats*



Par exemple, la commande `find` permet de chercher des chemins de fichiers, et la commande `grep` permet de chercher du texte dans un seul fichier. On peut combiner ces commandes : chercher tous les noms de fichiers .txt dans le répertoire utilisateur, puis chercher une chaîne de caractères dans ces fichiers

- *\$ find ~ -name '*.txt' | grep -i "ma recherche"*

Exemple de redirection complexe

```
$ wc -l *.pdb
```

wc -l *.pdb

OUT



Output in Shell

```
$ wc -l *.pdb > lengths
```

wc -l *.pdb

OUT



lengths

Output in File

```
$ wc -l *.pdb | sort -n | head -n 1
```

wc -l *.pdb

OUT

sort -n

OUT

head -n 1

OUT



Output in Shell

PARTIE III

GESTION DES

PROCESSUS



PS : LISTER LES PROCESSUS

La commande ps (process status) permet d'afficher la table des processus.

Dans la pratique, cette commande est utile pour récupérer un ID de processus, afin de le fournir à des commandes de gestion de ce processus.



Par exemple, la commande suivante affiche toute l'information disponible sur tous les processus du système :

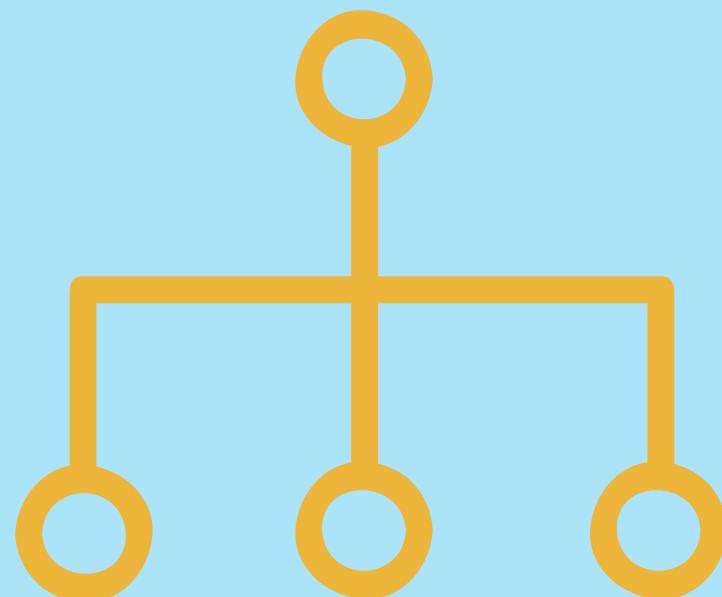
```
# ps -ef
```

- 💡 *La liste des processus affichés dépend des droits de l'utilisateur : pour avoir accès à tous les processus du système, on devra parfois utiliser le compte root.*

PSTREE : AFFICHER LA HIÉRARCHIE DES PROCESSUS

La commande **pstree** (arbre des processus) permet d'afficher la hiérarchie de la table des processus sous la forme d'un arbre.

Dans la pratique, cette commande est utile pour récupérer les processus ayant été créés par une commande ou un programme.



Cette commande permet d'afficher la relation parent / enfant des processus :

\$ *pstree*

KILL : CHANGER L'ÉTAT D'UN PROCESSUS

La commande `kill` permet d'envoyer un signal à un processus.

Sous Linux, un processus a la capacité de recevoir un signal du système ou d'un utilisateur.

Dans la pratique, comme son nom l'indique cette commande est souvent utilisée pour mettre fin à un processus, de manière conventionnelle ou non.



Il est possible de demander au processus de terminer son action (action conventionnelle, similaire à un bouton Quitter, ...):

```
$ kill -TERM MonIDdeProcessus
```

Il est aussi possible, par exemple lorsque le processus ne répond plus, de le tuer. Attention, cette action est instantanée et le processus n'effectuera aucune opération de fin d'exécution (sauvegarde, fermeture de connexion, ...) !

```
$ kill -9 MonIDdeProcessus
```

KILL : CHANGER L'ÉTAT D'UN PROCESSUS

La commande kill est à utiliser avec prudence : Linux est un système qui suit une philosophie de simplicité et d'efficacité et demande rarement confirmation des actions.

Par exemple, la commande suivant va tuer tous les processus dont l'utilisateur a le contrôle, sans limitation et sans confirmation, ce qui peut laisser le système dans un état inutilisable :

```
$ kill -9 -1
```



Il n'est bien sûr pas possible de terminer des processus sur lesquels l'utilisateur n'a pas les droits de gestion.



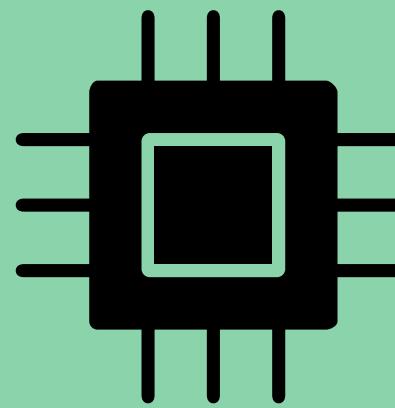
La plupart des distributions Linux fournissent une commande xkill qui permet de tuer une application simplement à travers l'interface graphique

TOP : INFORMATIONS SYSTÈME

La commande **top** permet de surveiller des informations sur le système.

Cette commande fournit des *informations mises à jour automatiquement* sur :

- La plupart des ressources système (CPU, mémoire, charge, ...)
- Tous les processus avec des *informations sur leur consommation mémoire et CPU*
- C'est une commande utile pour analyser des problèmes de performances



Cette commande ne prend pas d'argument :

```
$ top
```

W : ACTIVITÉ UTILISATEUR

La commande w permet de surveiller l'activité des utilisateurs.

Dans la pratique, cette commande est surtout utilisée pour vérifier qui est connecté sur le système.



Sans argument, cette commande affiche la liste des utilisateurs connectés :

```
$ w
```

(RE)NICE : PRIORITISER LES ACTIVITÉS

Les commandes **nice** et **renice** permettent d'appliquer et de changer une priorité à une commande.

La commande nice exécute une commande avec une priorité :

```
$ nice -n MaPriorite MaCommande
```

La commande renice modifie la priorité d'un processus :

```
$ renice MaPriorite MonIDdeProcessus
```

PARTIE IV

SYSTÈME DE

FICHIERS

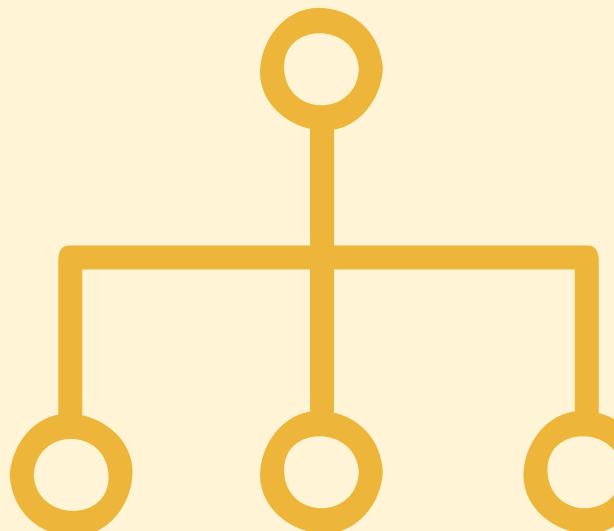
Qu'est-ce qu'un fichier sous Linux ?
Quelle est leur utilité ?
Comment les administrer ?



HIÉRARCHIE DE FICHIERS

Les fichiers et dossiers d'un système Linux sont ordonnés dans un unique système de fichiers hiérarchisé :

- La partition principale est montée en premier pour créer la racine de ce système de fichiers. Cette racine est notée /
- Le système et les utilisateurs peuvent monter d'autres partitions (par exemple : partition d'une clé USB) dans n'importe quel répertoire vide, n'importe où dans le système
- La liste des partitions connues du système est stockée dans une table des partitions.



Cette gestion des fichiers est très différente des systèmes DOS/Windows où chaque partition crée un disque associé à une lettre de montage (C; D; ...). Dans un système GNU/Linux, on utilisera des répertoires, généralement /mnt/maPartition ou /media/cdrom, ...

MOUNT : MONTER UNE PARTITION

La commande **mount** permet de monter une partition, c'est-à-dire de lui assigner un point de montage (répertoire où le système de fichiers de la partition sera accessible), ainsi qu'un type et/ou une technologie pour accéder à ces fichiers.



Utilisée sans argument, cette commande affiche la liste des partitions actuellement montées sur le système :

```
$ mount
```

 Le fichier de configuration `/etc/fstab` contient la liste des partitions à monter au démarrage du système

PWD : RÉPERTOIRE COURANT

La commande **pwd** (print working directory) permet d'afficher le répertoire de travail courant, c'est-à-dire le dossier dans lequel est actuellement le shell.



Utilisée sans argument, cette commande affiche le répertoire courant :

```
$ pwd
```

LA VARIABLE \$PATH

Jusqu'ici, nous avons toujours fait référence à des commandes en utilisant uniquement leur nom : ls, top, ...

En réalité, absolument tout en Linux est fichier... y compris une commande système !

Ces noms de commandes renvoient en fait à des fichiers du même nom dans des dossiers bien connus du système : /bin/ls, /usr/bin/top, ...



La variable PATH est une variable réservée du SHELL qui fournit une liste de chemins à scanner pour y trouver des fichiers exécutables. Chaque fichier dans ces dossiers pourra être utilisé en utilisant directement son nom de fichier plutôt que le chemin d'accès complet.

Il est possible de modifier le PATH pour ajouter son propre dossier de commandes :

```
$ export PATH=$PATH:MonDossierAjouter
```

WHICH : LOCALISATION DE COMMANDE

La commande **which** permet de remonter le path, pour retrouver le chemin d'accès complet d'une commande depuis son nom.



Dans la pratique, cette commande est utile lorsqu'il y a plusieurs versions d'une commande installée sur le système : par exemple, on pourra chercher le chemin utilisé pour la commande `java`.

```
$ which java
```

DU : UTILISATION DISQUE

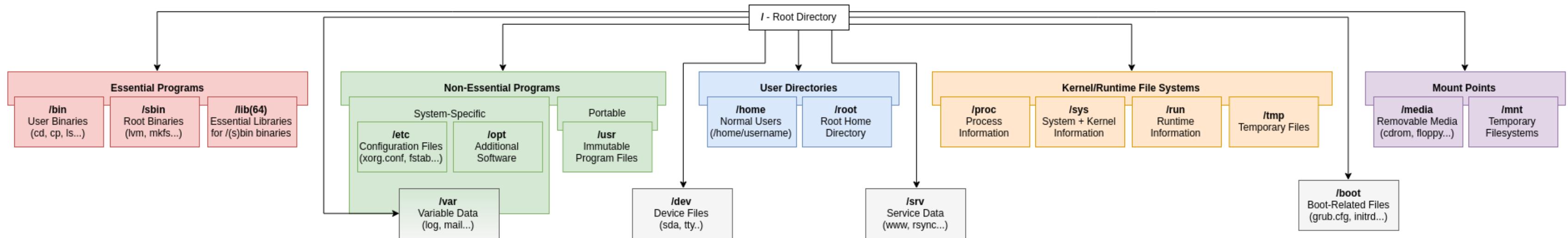
La commande **du** (disk usage) permet de calculer l'utilisation disque.



Par exemple, la commande suivante affiche l'utilisation disque d'un répertoire, dans un format compréhensible par un humain :

```
$ du -sh MonRépertoire
```

The Filesystem Hierarchy Standard (FHS)



Essential Programs:

Directories containing files needed to run essential programs

- **/bin** - Essential binaries such as 'cp' or 'ls' that all users have access to
- **/sbin** - Essential binaries only available to the root user
- **/lib(64)** - Libraries needed for essential binaries in /sbin

Non-Essential Programs (Secondary Hierarchy):

Directories containing files needed to run non-essential programs

- **/etc** - System-specific configuration files for programs in /usr and /opt
- **/opt** - Additional programs not found in distribution repositories
- **/usr** - Portable, read-only, non-essential programs and program files
- **/var** - Used for storing dynamic program data that may change

Mount Points:

Directories used for mounting devices and file systems

- **/media** - Removable media such as CD-ROMs and floppy drives
- **/mnt** - Temporary file systems such as USB drives

User Directories:

Directories containing user-specific files

- **/home/(username)** - User files, configuration, and programs
- **/root** - Home directory for the root user

Kernel/Runtime File Systems:

Directories populated by the kernel to provide information to programs and the user

- **/proc** - Information about processes, the kernel and system hardware
- **/sys** - Information about system hardware and the kernel
- **/run** - Information about the system since the last boot
- **/tmp** - Directory for temporary files. Usually a tmpfs that is cleared on boot

Other directories:

Version 1.1 - Created by Max Hösel and licensed under the Creative Commons CC-BY 4.0 license. Last edit: 2018-05-20

A close-up shot of a chain-link fence, with the background being a soft-focus sunset or sunrise. The sky is filled with warm orange and yellow hues, with a few bright white circles of light visible through the fence's mesh.

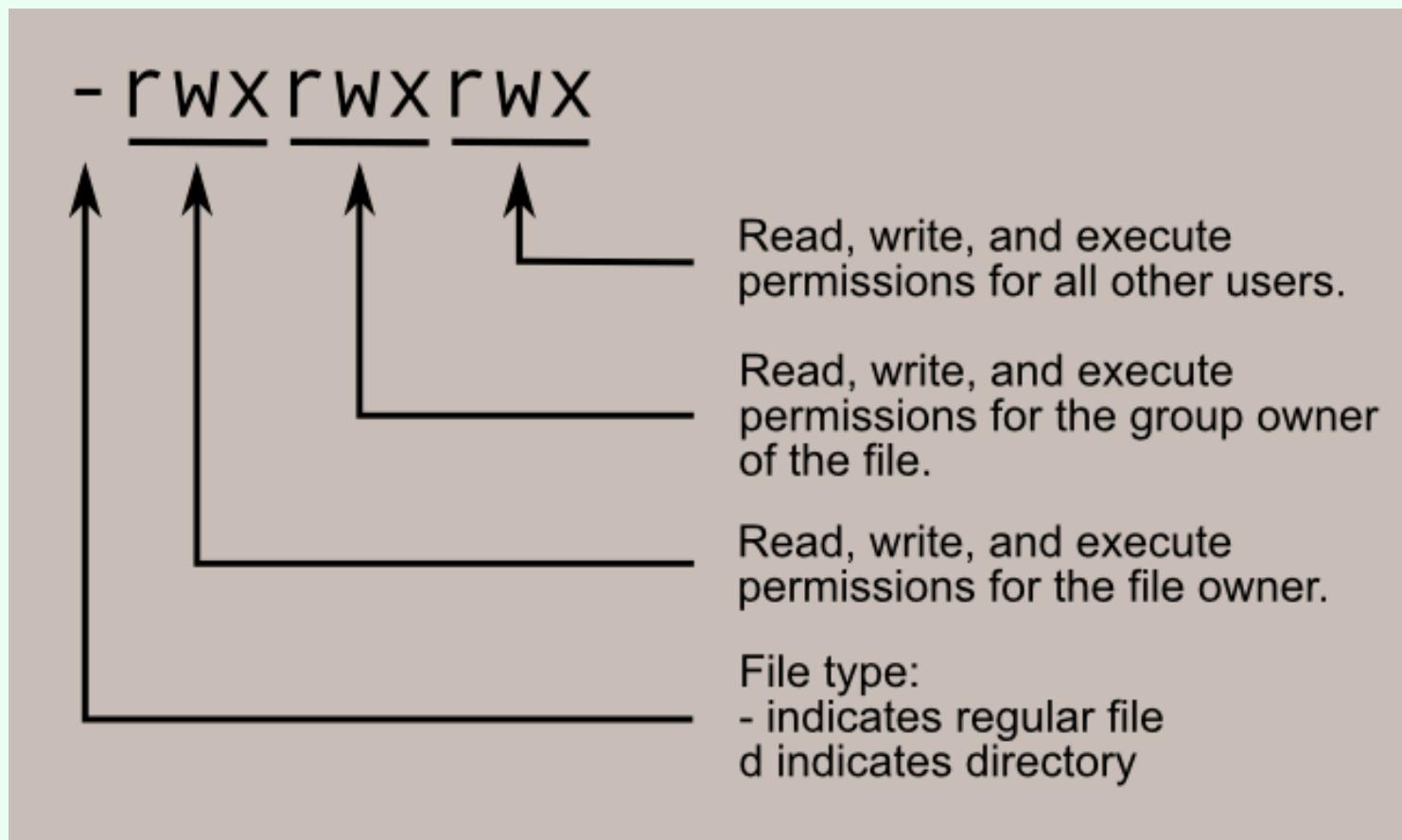
**GESTION DES
DROITS D'ACCÈS**

DROITS D'ACCÈS



Linux sécurise tous les fichiers avec 3 permissions : **lecture(read)**, **écriture(write)**, **exécution**.

Ces permissions sont distinguées pour : **le propriétaire**, **le groupe du propriétaire**, **les autres utilisateurs**.



```
$ ls -l test.txt
```

```
-rw-rx-r--. 1 tavenel vboxusers 0 Nov 16 14:39 test.txt
```

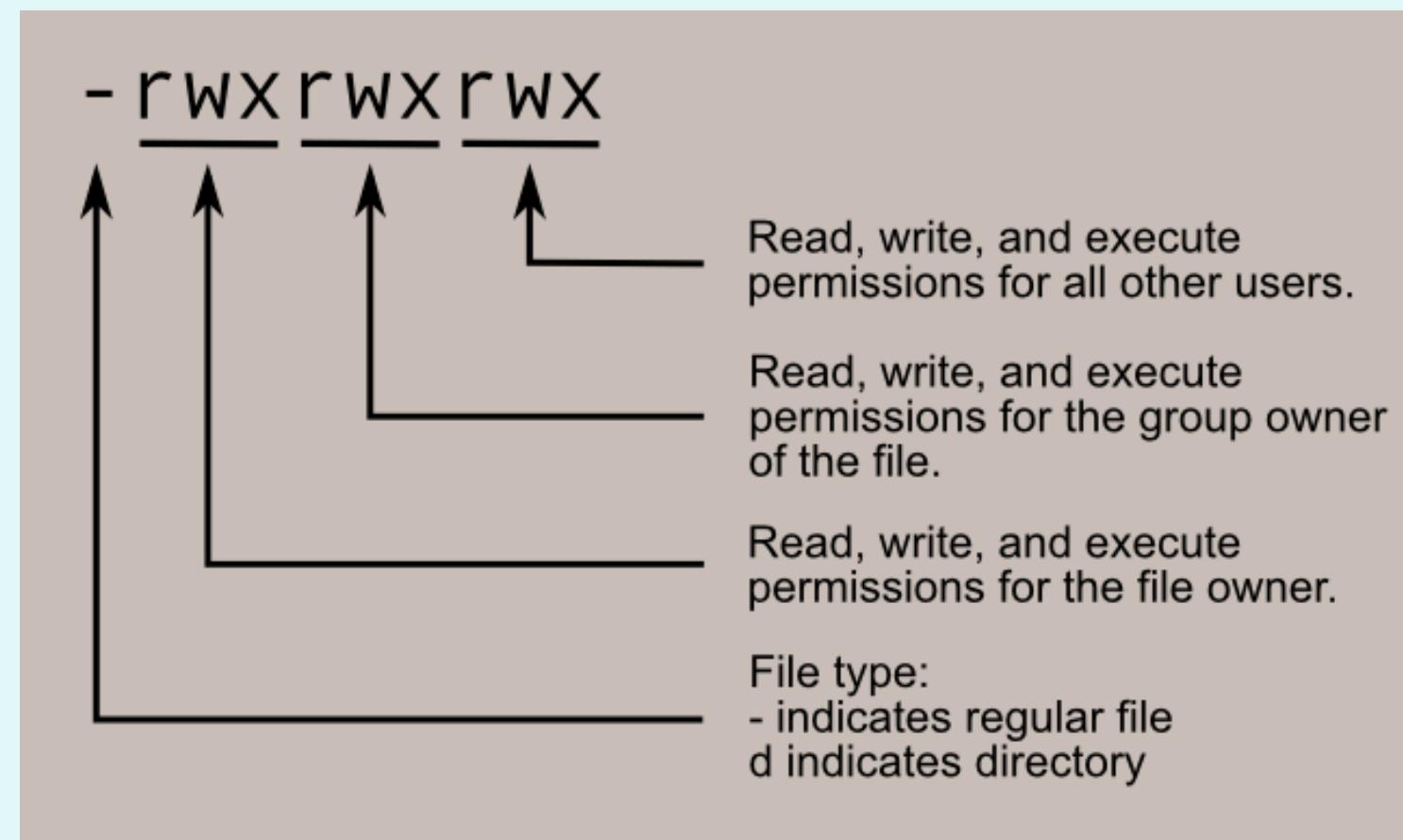
Dans cet exemple, le propriétaire (tavenel) peut lire et écrire dans ce fichier, les utilisateurs du groupe propriétaire (vboxusers) peuvent lire et exécuter ce fichier, et tous les autres utilisateurs peuvent lire ce fichier.



CHMOD : CHANGER LES PERMISSIONS

La commande **chmod** permet de changer les permissions d'un fichier.

Cette commande applique un bitmask de permissions sur le fichier. Un bitmask est un masque bit à bit : chaque fois que la permission est présente, le bit vaut 1, sinon le bit vaut 0.



Par exemple, pour donner les droits Lire/Ecrire/Exécuter à l'utilisateur et Lire/Ecrire à son groupe :

- on applique le masque `rwx rw---`
- on applique le bitmask correspondant : `111 110 000`
- on convertit le format binaire en décimal : `760`

```
$ chmod 760 test.txt
```

```
$ ls -l test.txt
```

```
-rwxrw----. 1 tavenel vboxusers 0 Nov 16 14:39 test.txt
```



PERMISSIONS DES RÉPERTOIRES

Les permissions sur un répertoire ont un sens légèrement différent :



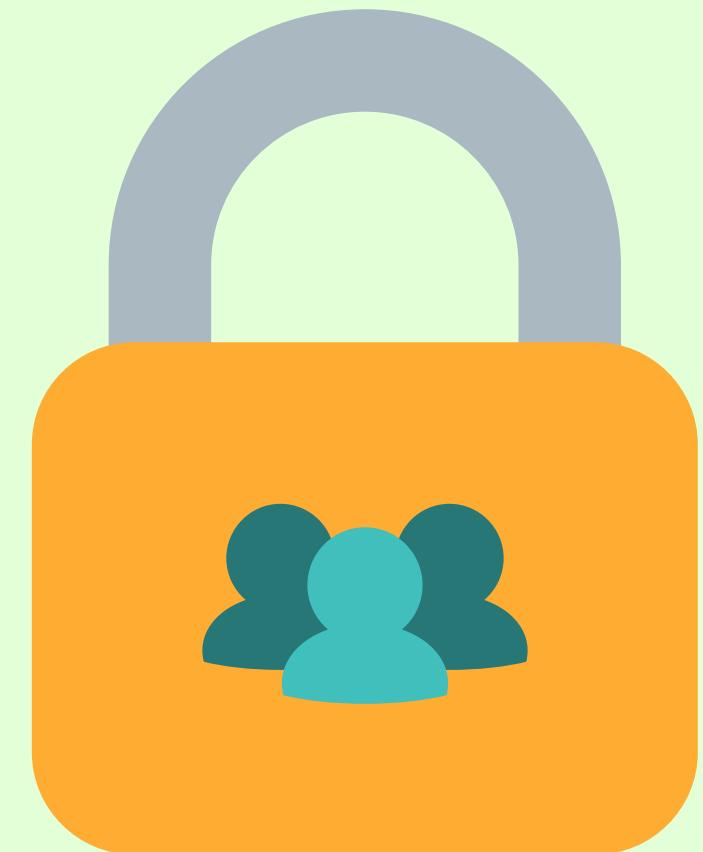
- *r* : Autorise à *lister le contenu du répertoire* (si le droit *x* est également présent)
- *w* : Autorise à *ajouter, supprimer ou renommer des fichiers dans le dossier* (si le droit *x* est également présent)
- *x* : Autorise à *se déplacer dans un répertoire* (commande *cd*)

CHOWN / CHGRP : CHANGER LE PROPRIÉTAIRE

Les commandes **chown** (change owner) et **chgrp** (change group) permettent de changer le propriétaire d'un fichier et le groupe auquel il appartient.

Exemple pour changer l'utilisateur et/ou le groupe d'un fichier :

```
$ chown NouvelUtilisateur:NouveauGroupe MonFichier1  
$ chgrp NouveauGroupe MonFichier2
```



GESTION DES UTILISATEURS

GESTION DES UTILISATEURS

Les systèmes GNU/Linux sont des systèmes multi-utilisateurs : plusieurs utilisateurs peuvent se connecter sur le système en même temps.

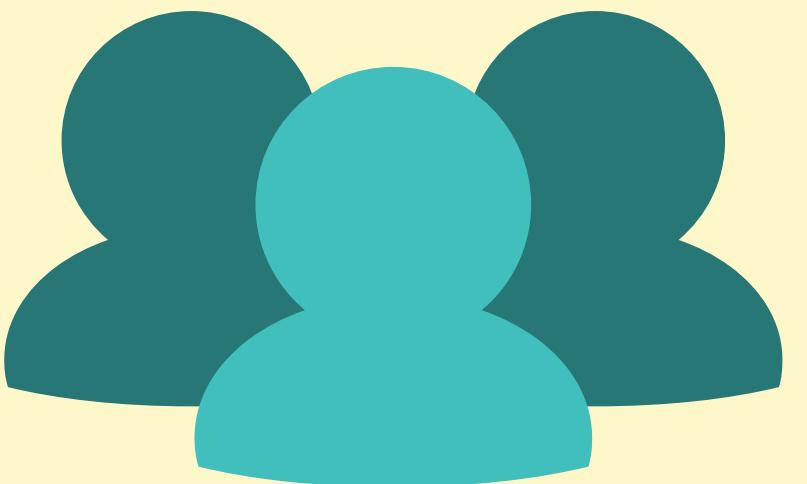
On peut s'attendre à ce que plusieurs utilisateurs aient besoin d'accéder au même fichier ou dossier au même moment.

Partager des mots de passe pour permettre ces accès serait une mauvaise pratique de sécurité : comment faire alors pour partager des droits sans partager de secret ?

Linux utilise la notion d'utilisateur et de groupe pour partager les mêmes droits d'accès à différents utilisateurs ayant des comptes et mots de passe différents.



Certains patchs du noyau et certains modules ajoutent une gestion beaucoup plus fine de la sécurité, en utilisant par exemple un vrai système de rôles et de règles. Le plus connu et utilisé d'entre eux est SELinux.



ADDUSER / USERADD : AJOUTER UN UTILISATEUR

Les commandes **adduser** et **useradd** permettent d'ajouter un nouvel utilisateur sur le système.

Adduser est *interactif* alors que useradd est prévu pour être utilisé dans des scripts.

```
$ adduser NouvelUtilisateur
```



La commande **userdel** permet de supprimer un utilisateur et ses fichiers.

```
$ userdel UtilisateurExistant
```

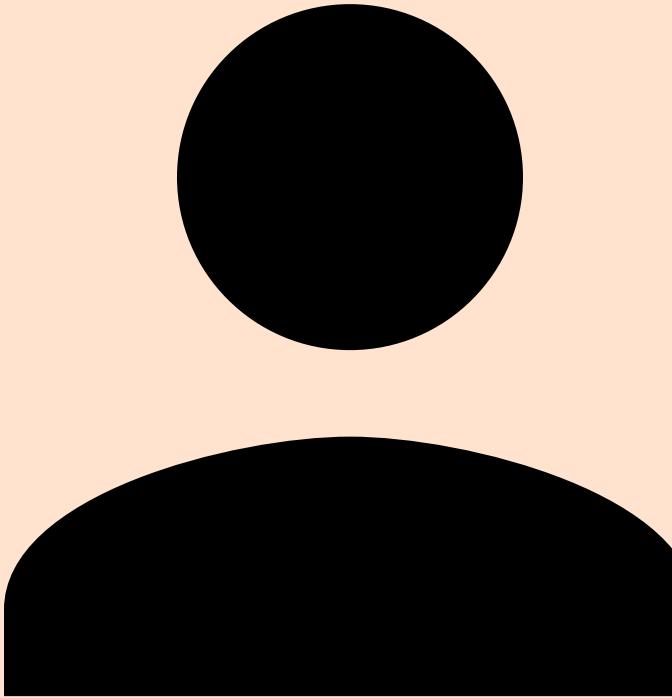
Les commandes **addgroup** et **delgroup** permettent d'ajouter / supprimer un groupe dans le système.

```
$ addgroup NouveauGroupe
```

ID, USERMOD, PASSWD : GESTION UTILISATEUR

La commande **id** permet d'obtenir des informations sur un utilisateur et son groupe

```
$ id UtilisateurExistant
```



La commande **usermod** permet de modifier un compte utilisateur.

Par exemple, on peut ajouter un utilisateur à un groupe existant :

```
$ usermod -a -G GroupeID1,GroupeID2 MonUtilisateur
```

La commande **passwd** permet de modifier le mot de passe d'un utilisateur

```
$ passwd UtilisateurExistant
```

SU : CONNEXION EN TANT QU'AUTRE UTILISATEUR

La commande **su** (à l'origine : super-user) permet de lancer une nouvelle session de shell en se connectant avec un autre compte utilisateur (par défaut le super-utilisateur root).

Attention, il s'agit d'une véritable session du nouvel utilisateur : les commandes sont utilisées et loggées chez le nouvel utilisateur !

Par défaut cependant, su conserve les variables de l'environnement courant avant de changer de contexte (sauf si l'option **-** ou **-l** est passée en paramètre)

- *root login (garder l'environnement courant) :*

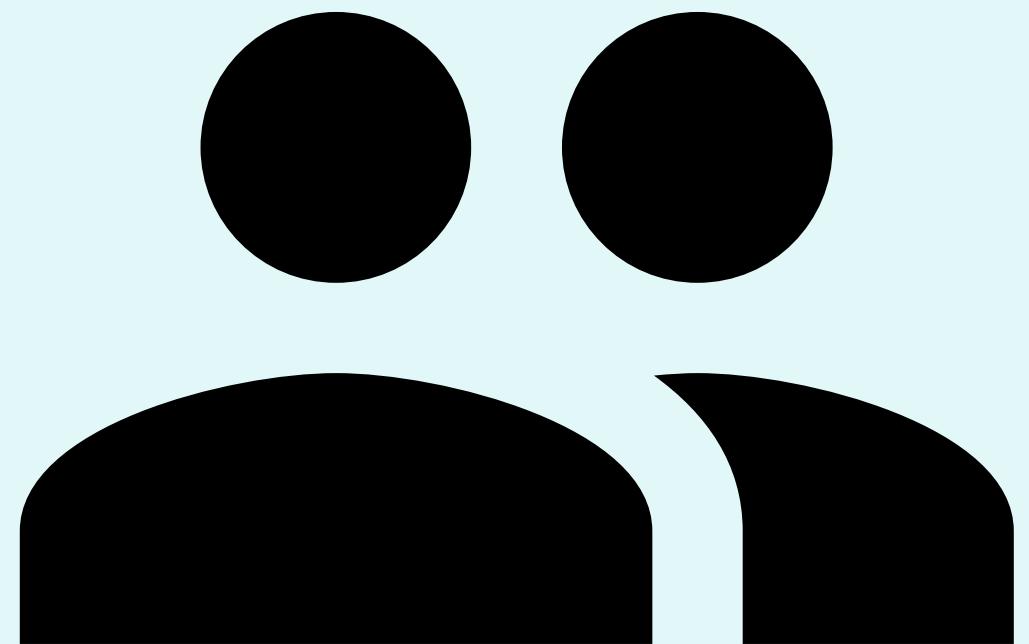
```
$ su
```

- *root login (nettoyer l'environnement) :*

```
$ su -
```

- *connexion comme AutreUtilisateur (nettoyer l'environnement) :*

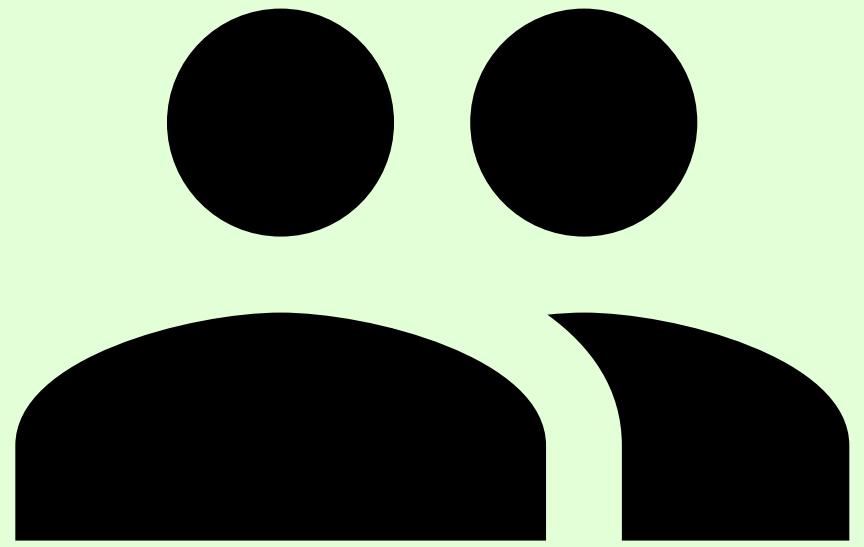
```
$ su - AutreUtilisateur
```



SUDO : DÉLÉGATION À UN AUTRE UTILISATEUR

La commande **sudo** (à l'origine : super-user do) permet de lancer une nouvelle commande (et uniquement une commande) en déléguant son exécution à un autre utilisateur (par défaut le super-utilisateur root).

```
$ sudo -u AutreUtilisateur ls /UnRepertoirePrive  
$ sudo ls /root
```



La connexion dans un shell complet en tant qu'utilisateur root posant de nombreux problèmes de sécurité, certaines distributions récentes (Ubuntu, ...) ont choisi de la désactiver par défaut. Si besoin, il est toujours possible de se connecter en tant que root, il suffit pour cela... d'avoir les droits root !

```
$ sudo su
```

Cette méthode est bien meilleure car l'utilisateur ayant augmenté ses droits au moment du login root est enregistré.

FICHIERS DE CONFIGURATION

Rappel : en Linux, tout est fichier... y compris les configurations !

Les utilisateurs et leur configuration (groupes, home, shell par défaut, ...) sont stockés dans le fichier **/etc/passwd**

Les informations sur les groupes sont stockées dans le fichier **/etc/groups**

La configuration des utilisateurs pouvant passer root en exécutant la commande sudo est stockée dans le fichier **/etc/sudoers**

