



# ADMINISTRATION AVANCÉE DES SYSTÈMES LINUX

Pr.: B. CHERKAOUI

Email: [b.cherkaoui@ucd.ac.ma](mailto:b.cherkaoui@ucd.ac.ma)

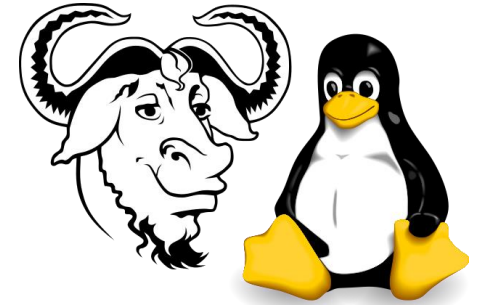
# PLAN DU COURS

- I. Introduction
- II. Rappels et aperçu rapide du système
- III. Utilisateurs
- IV. Les droits d'accès
- V. Processus
- VI. Scripting Shell
- VII. Planification des tâches (Cron)



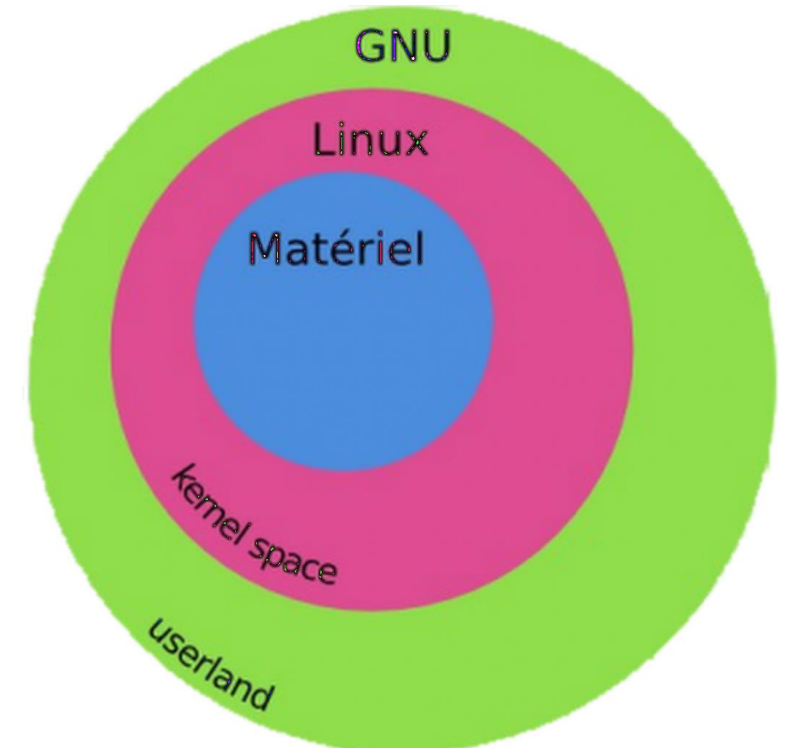
# INTRODUCTION

# INTRODUCTION



Qu'est-ce que GNU ? Linux ? GNU/Linux ?

- GNU( Sous une licence GPL: General Public License) est un système d'exploitation (comme Windows ou OS X), qui utilise le noyau Linux pour former le système GNU / Linux.
- GNU/Linux est la terminaison d'un système complet ... mais souvent, on dit "Linux" pour faire plus court



# INTRODUCTION

Historique...

*“There is no system but GNU, and Linux is one of its kernels.”*

*Richard M. Stallman*

Le système GNU:

- Le système GNU a été créé en 1983 par Richard M. Stallman.
- C'est une version gratuite et “libre” (voir plus loin) du système Unix
- GNU = “**G**NU’s **N**ot **U**nix” (acronyme récursif) ;



**Manque d’un noyau**

# INTRODUCTION

Historique...

***“I’m doing a (free) operating system (just a hobby, won’t be big and professional like gnu) for 386(486) AT clones. “***

*Linus B. Torvalds*

Le noyau Linux

- Le noyau Linux a été créé en 1992 par Linus B. Torvalds
- Torvalds a également créé git



**GNU et Linux sont du free software ; il s’agit de “logiciel libre”, et pas “logiciel gratuit” (free as in freedom) !**

# INTRODUCTION

Historique...

- GNU et Linux sont du free software ; il s'agit de “logiciel libre”, et pas “logiciel gratuit” (free as in freedom) !
- GNU définit quatre libertés essentielles (paraphrasées) :
  - le droit d'utiliser le logiciel sans restrictions
  - l'accès au code source et le droit d'étudier et modifier le logiciel
  - la redistribution sans restrictions du logiciel ainsi que vos modifications apportées



**Possibilité d'étudier et d'améliorer le système  
et même créer son propre système GNU / Linux**

# INTRODUCTION

## Statistiques sur Linux...

L'utilisation de Linux en chiffre:

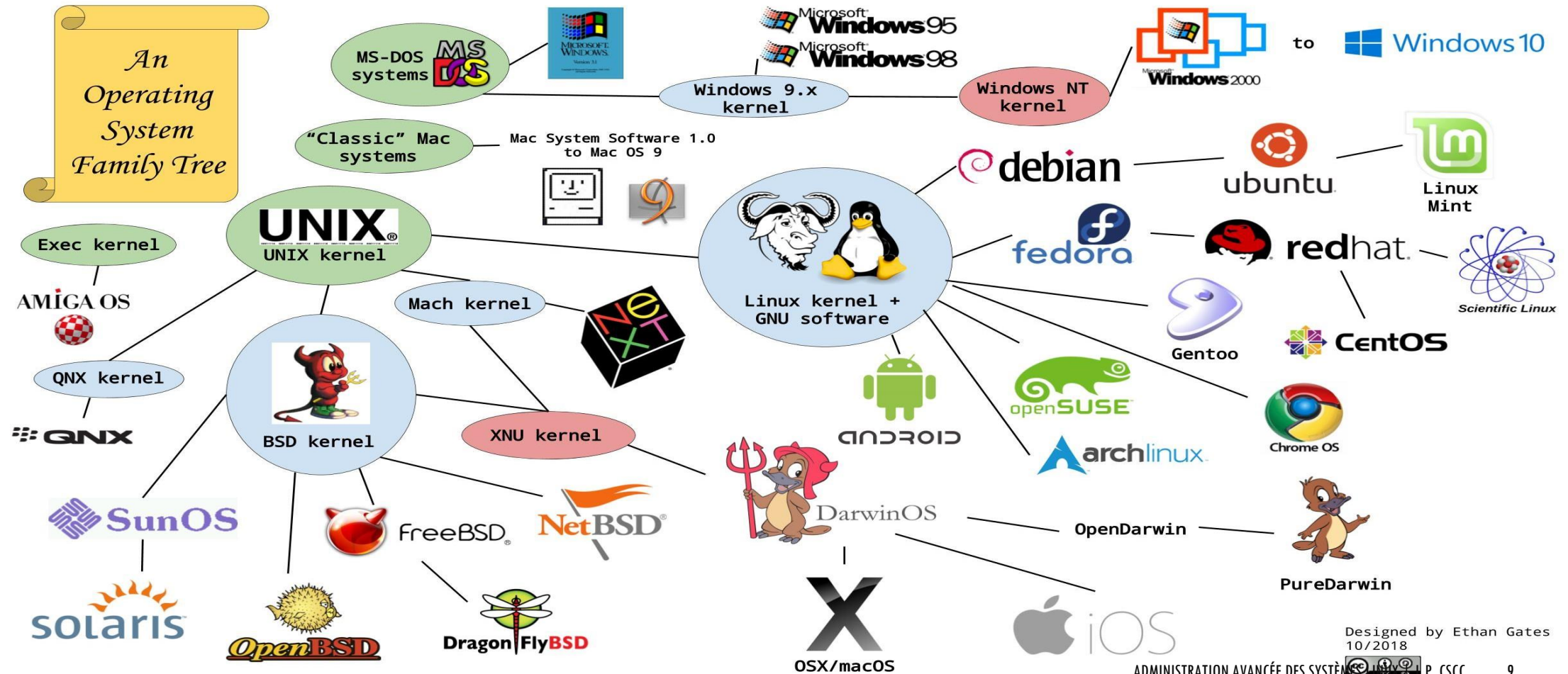
- DeUser1top / Laptop : environ 2.87%
- Mainframe : environ 28%
- Embarqué : environ 38.42%
- Smartphone / Tablette : environ 70.80%
- Serveurs : environ 77.4%
- Supercomputers : 100%

([source](https://en.wikipedia.org/wiki/Usage_share_of_operating_systems) : [https://en.wikipedia.org/wiki/Usage\\_share\\_of\\_operating\\_systems](https://en.wikipedia.org/wiki/Usage_share_of_operating_systems))



# INTRODUCTION

La famille OS...



Designed by Ethan Gates  
10/2018

# INTRODUCTION

La famille OS...

Quelle distribution choisir ?

- Il existe beaucoup de variantes de GNU / Linux, que l'on appelle des **distributions**
- Une **distribution** est OS complet directement installable, utilisable, et maintenus par une équipe de développeurs.
- On utilisera **Ubuntu**, plus accessible aux débutants



les distributions se ressemblent très fort  
ce qu'on apprend sous Ubuntu nous servira sous  
les autres distributions



# RAPPELS ET APERÇU RAPIDE SUR LE SYSTÈME

# RAPPEL ET APERÇU SU LE SYSTÈME

- L'administration d'un système GNU / Linux se fait principalement à l'aide de la ligne de commande(CLI).
- Les commandes sont écrites et exécutés via un **Terminal** afin de réaliser les tâches voulues

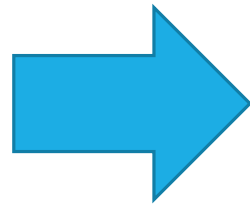
Avantages	Inconvénients
<ul style="list-style-type: none"><li>○ Administration à distance (via SSH par exemple)</li><li>○ Programmation de scripts de maintenance</li><li>○ Uniformité : les commandes fonctionnent sur la plupart des systèmes malgré leurs différences.</li><li>○ Flexibilité : Plusieurs manières/méthodes pour effectuer une tâche.</li></ul>	<ul style="list-style-type: none"><li>○ Il faut apprendre et maîtriser l'utilisation de toutes les commandes</li></ul>

# RAPPEL ET APERÇU SU LE SYSTÈME

TOUT EST UN FICHIER

Sous GNU / Linux, tout est un fichier : les fichiers bien sûr, mais aussi :

- Les répertoires
- Les liens
- les périphériques
- Les entrées/Sorties



**Traitement unifié de beaucoup d'objets  
différents**

# RAPPEL ET APERÇU SU LE SYSTÈME

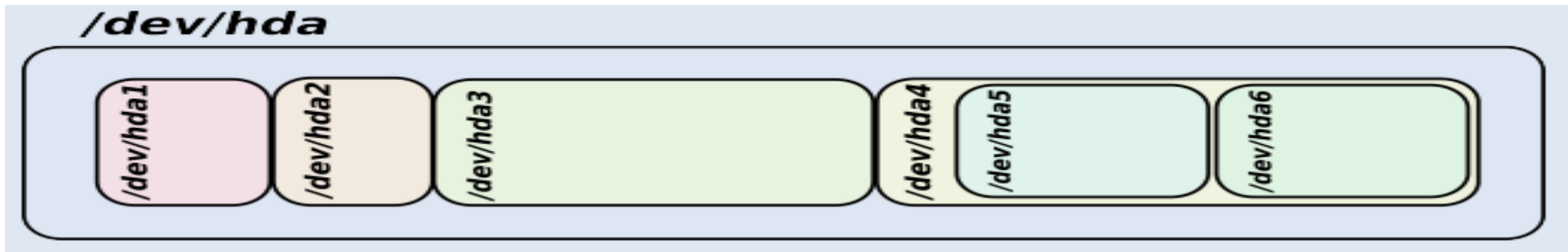
## TYPE DES FILESYSTEM

- Un système de fichiers (filesystems FS) désigne à la fois l'organisation hiérarchique des fichiers au sein d'un système d'exploitation, et aussi l'organisation des fichiers au sein d'un volume physique ou logique, qui peut être de différents types.
- Un FS est généralement journalisé, c-à-d note toutes les transactions à venir avant de les exécuter. En cas de crash, le système peut savoir ce qui a été fait et ce qui ne l'a pas été.

# RAPPEL ET APERÇU SU LE SYSTÈME

TYPE DES FILESYSTEM

- Un disque est découpé en partitions primaires (max : 4) et étendues.
- les partitions sont représentées par des numéros à la fin du device représentant le disque



- les devices dépendent de l'interface du disque et de sa position (e.g. `/dev/hda` pour le 1<sup>er</sup> disque IDE, `/dev/hdb` pour le 2<sup>ème</sup>, etc...)

# RAPPEL ET APERÇU SU LE SYSTÈME

## TYPE DES FILESYSTEM

Type FS	Description
<b>minix</b>	est le système de fichiers utilisé par le système d'exploitation <b>Minix</b> , le premier à avoir fonctionné sous Linux. Il a de nombreuses limitations (un maximum de 64 Mo par partition, des noms de fichiers courts, un seul horodatage, etc.). Néanmoins, il reste très appréciable pour les disquettes et les disques en mémoire vive.
<b>ext</b>	est une extension élaborée du système de fichiers <b>minix</b> . Il a été complètement remplacé par sa seconde version ( <b>ext2</b> ) et supprimé du noyau (depuis la version 2.1.21).
<b>ext2</b>	est un système de fichiers de hautes performances, utilisé par Linux pour les disques fixes tout autant que pour les supports amovibles. Le second système de fichiers étendu a été conçu comme une extension du système ( <b>ext</b> ). <b>ext2</b> offre les meilleures performances (en termes de vitesse et de consommation CPU) de tous les systèmes de fichiers gérés par Linux. La taille maximale du fichier peut atteindre 2To.



# RAPPEL ET APERÇU SU LE SYSTÈME

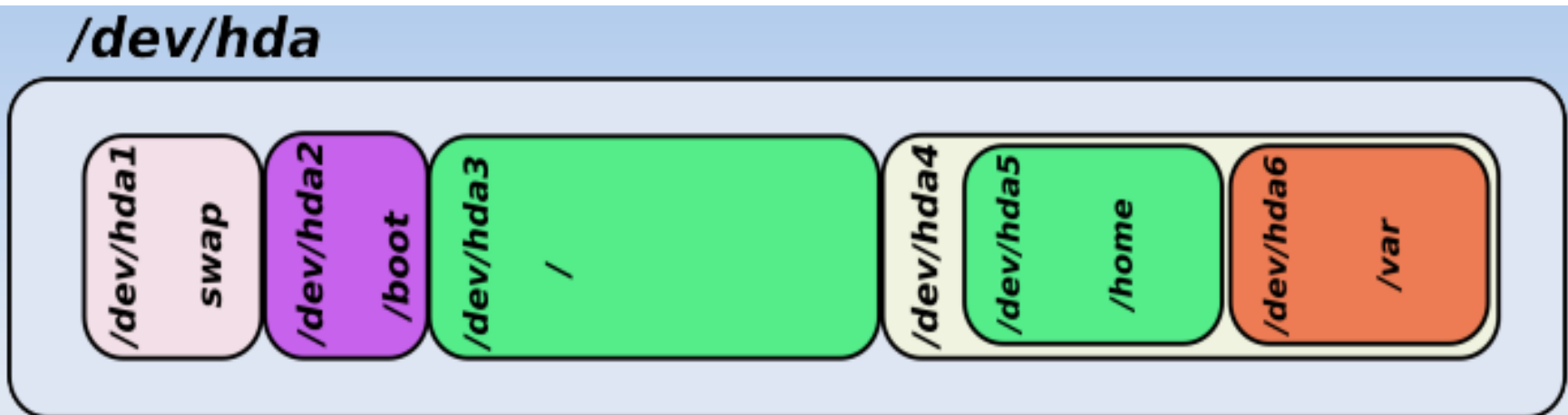
## TYPE DES FILESYSTEM

Type FS	Description
<b>ext3</b>	est une version d' <b>ext2</b> gérant la journalisation. On peut basculer facilement d' <b>ext2</b> à <b>ext3</b> , et inversement.
<b>ext4</b>	est un ensemble de mises à jour d' <b>ext3</b> qui apporte des améliorations notables en terme de performance et de stabilité, ainsi qu'une augmentation importante des limites des volumes, fichiers et tailles de répertoire.
<b>Reiserfs</b>	est un système de fichiers journalisé, conçu par Hans Reiser, qui a été intégré dans Linux avec la version 2.4.1 du noyau
<b>Swap</b>	est une zone d'un disque dur faisant partie de la mémoire virtuelle de l'ordinateur. Il est utilisé pour décharger la mémoire vive physique (RAM) de l'ordinateur lorsque celle-ci arrive à saturation. L'espace d'échange (swap), dans Ubuntu, se trouve généralement sous une forme de partition de disque dur – on parle alors de partition d'échange. Il peut aussi se présenter sous forme de fichier

# RAPPEL ET APERÇU SU LE SYSTÈME

## TYPE DES FILESYSTEM

- Chacune des partitions contient un FS qui sera monté dans l'arborescence du système.
- Pour avoir accès à leur contenu, il faut les monter, c-à-d les accrocher à un répertoire appelé point de montage.
- En réalité, un disque n'est pas monté directement mais bien ses partitions



 **Linux swap**

 **ext2**

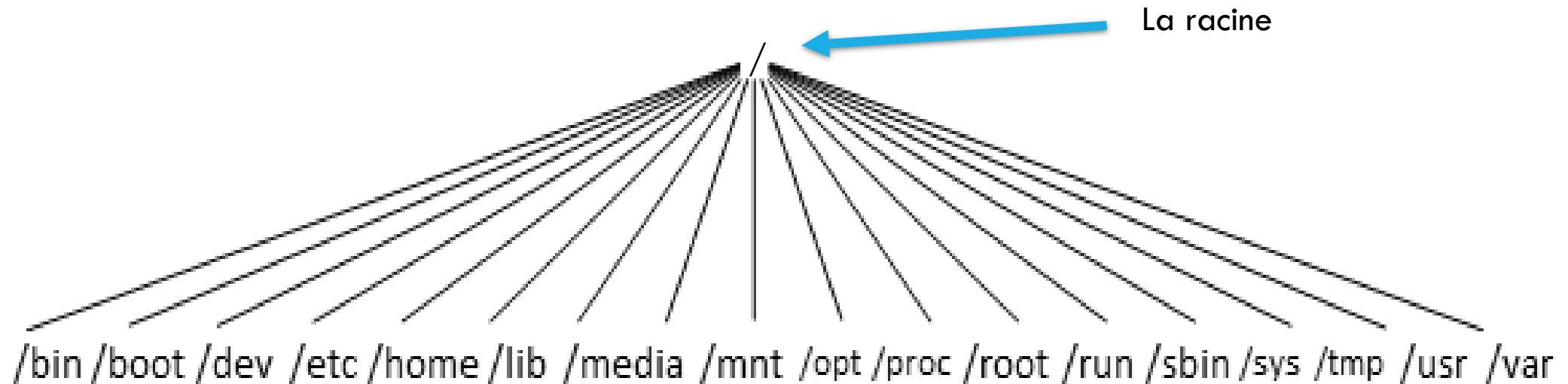
 **ext3**

 **reiserfs**

# RAPPEL ET APERÇU SU LE SYSTÈME

## STRUCTURE DU SYSTÈME DE FICHIERS

Le système de fichiers structure les données sur le(s) disque(s). C'est une arborescence qui suit les conventions Unix et se présente comme suit :



# RAPPEL ET APERÇU SU LE SYSTÈME

## STRUCTURE DU SYSTÈME DE FICHIERS

- “/” est la racine (à peu près comme “C:\” sous Windows)
- Le caractère “/” sépare les répertoires (“\” sous Windows)
- Les majuscules et minuscules importent (case sensitivity)
- Voici quelques commandes pour basculer entre les différents répertoires:
  - **cd** permet d’aller dans un répertoire
  - **ls** affiche le contenu d’un répertoire
  - **pwd** donne le chemin du répertoire actuel

# RAPPEL ET APERÇU SU LE SYSTÈME

## STRUCTURE DU SYSTÈME DE FICHIERS

Répertoire	Description
.	est le répertoire actuel
..	est le répertoire parent
<b>/dev</b>	(pour devices) contient le matériel (disques durs, processeurs, . . . )
<b>/etc</b>	contient les fichiers de configuration globaux
<b>/home</b>	contient les répertoires personnels des utilisateurs
<b>/mnt et /media</b>	contiennent les disques “montés”
<b>/tmp</b>	contient des fichiers temporaires : il est vidé à chaque redémarrage
<b>/var</b>	contient diverses données (en particulier des “logs” dans /var/logs)

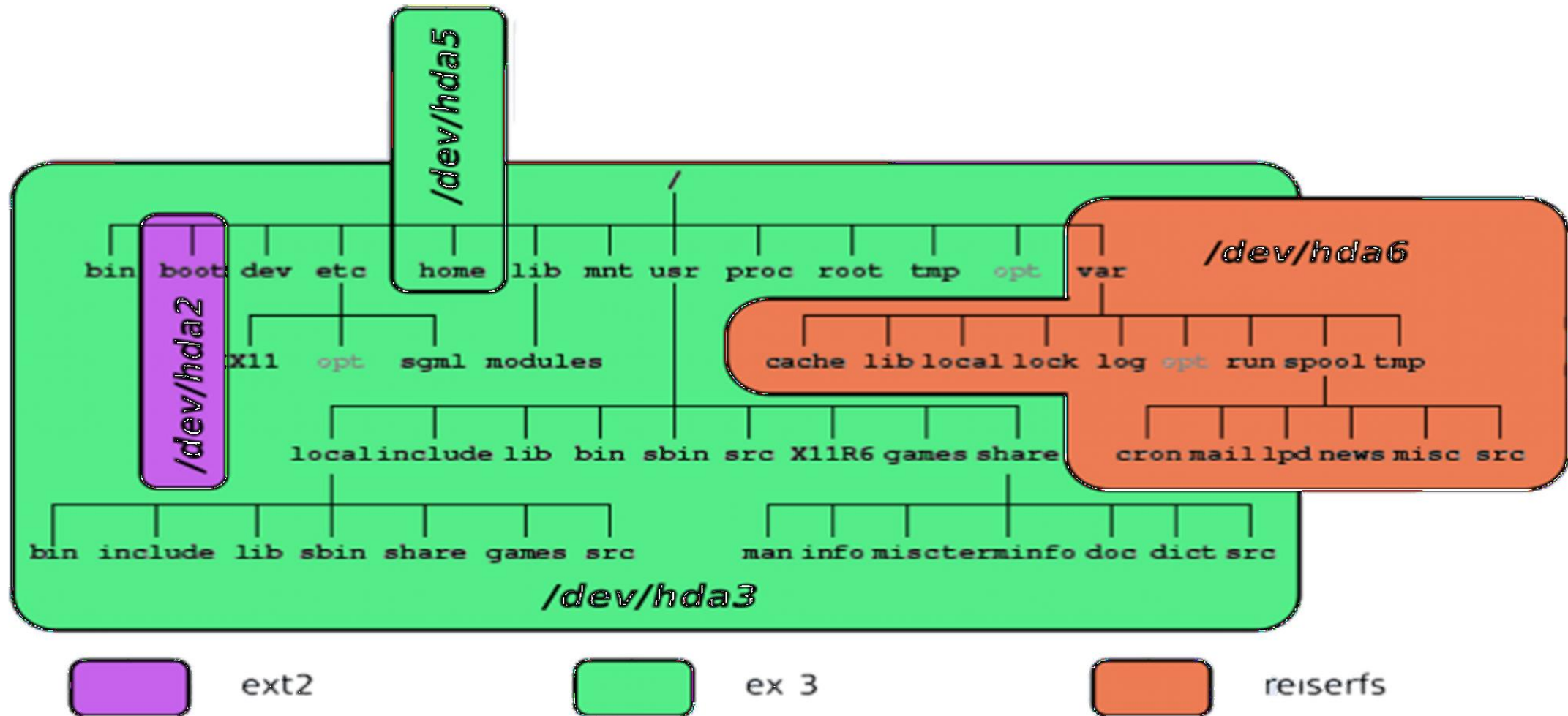
# RAPPEL ET APERÇU SU LE SYSTÈME

## STRUCTURE DU SYSTÈME DE FICHIERS

Répertoire	Description
<b>/bin</b>	contient les fichiers exécutables par l'utilisateur.
<b>/boot</b>	Contient le bootloader et le noyau exécutable ainsi que les fichiers de configuration nécessaires pour démarrer un OS Linux.
<b>/lib</b>	Contient les fichiers de la bibliothèque partagée qui sont nécessaires pour démarrer le système.
<b>/opt</b>	Les fichiers facultatifs, tels que les programmes d'application fournis par son éditeur, doivent être placés ici.
<b>/sbin</b>	System Binary Files, Il s'agit d'exécutables utilisés pour l'administration du système.
<b>/usr</b>	Il s'agit de fichiers partageables, en lecture seule, y compris les binaires et les bibliothèques exécutables, les fichiers man et d'autres types de documentation

# RAPPEL ET APERÇU SU LE SYSTÈME

## POINTS DE MONTAGE



# RAPPEL ET APERÇU SU LE SYSTÈME

## POINTS DE MONTAGE

- Les partitions à monter au boot sont décrites dans **/etc/fstab**

Partition à monter	Point de montage	Type FS	Options de montage	Boolean
<code>LABEL=ROOTFS</code>	<code>/</code>	<code>ext3</code>	<code>defaults</code>	<code>1 1</code>
<code>UUID=ed2d8d56-5c08-4bd4-977e-673f7a1966b2 /usr</code>		<code>ext3</code>	<code>defaults</code>	<code>0 0</code>
<code>/dev/hda1</code>	<code>/var</code>	<code>reiserfs</code>	<code>defaults</code>	<code>0 0</code>
<code>none</code>	<code>/dev/pts</code>	<code>devpts</code>	<code>gid=5,mode=620</code>	<code>0 0</code>
<code>none</code>	<code>/dev/shm</code>	<code>tmpfs</code>	<code>defaults</code>	<code>0 0</code>
<code>none</code>	<code>/proc</code>	<code>proc</code>	<code>defaults</code>	<code>0 0</code>
<code>none</code>	<code>/sys</code>	<code>sysfs</code>	<code>defaults</code>	<code>0 0</code>
<code>/dev/cciss/c0d0p2</code>	<code>swap</code>	<code>swap</code>	<code>defaults</code>	<code>0 0</code>
<code>/dev/cdrom</code>	<code>/mnt/cdrom</code>	<code>udf,iso9660</code>	<code>noauto,owner,ro</code>	<code>0 0</code>
<code>/dev/fd0</code>	<code>/mnt/floppy</code>	<code>auto</code>	<code>noauto,owner</code>	<code>0 0</code>
<code>/dev/ida/c0d0p1</code>	<code>/home</code>	<code>ext3</code>	<code>defaults</code>	<code>1 1</code>

- Un booléen à 1 si le FS doit être sauvegardé par dump
- Un numéro d'ordre pour la vérification de FS au boot



# RAPPEL ET APERÇU SU LE SYSTÈME

## POINTS DE MONTAGE

- Pour monter un périphérique, on utilise la commande **mount**
- La commande **mount** permet de manipuler tous les montages de systèmes de fichier de manière très précise.

```
sudo mount -t [type] /dev/sdc3 /media/stock
```

**Exemple:** monter l'unité de stockage /dev/sdc3 dans le dossier /media/stock

Sans spécification du type FS

```
sudo mount /dev/sdc3 /media/stock
```

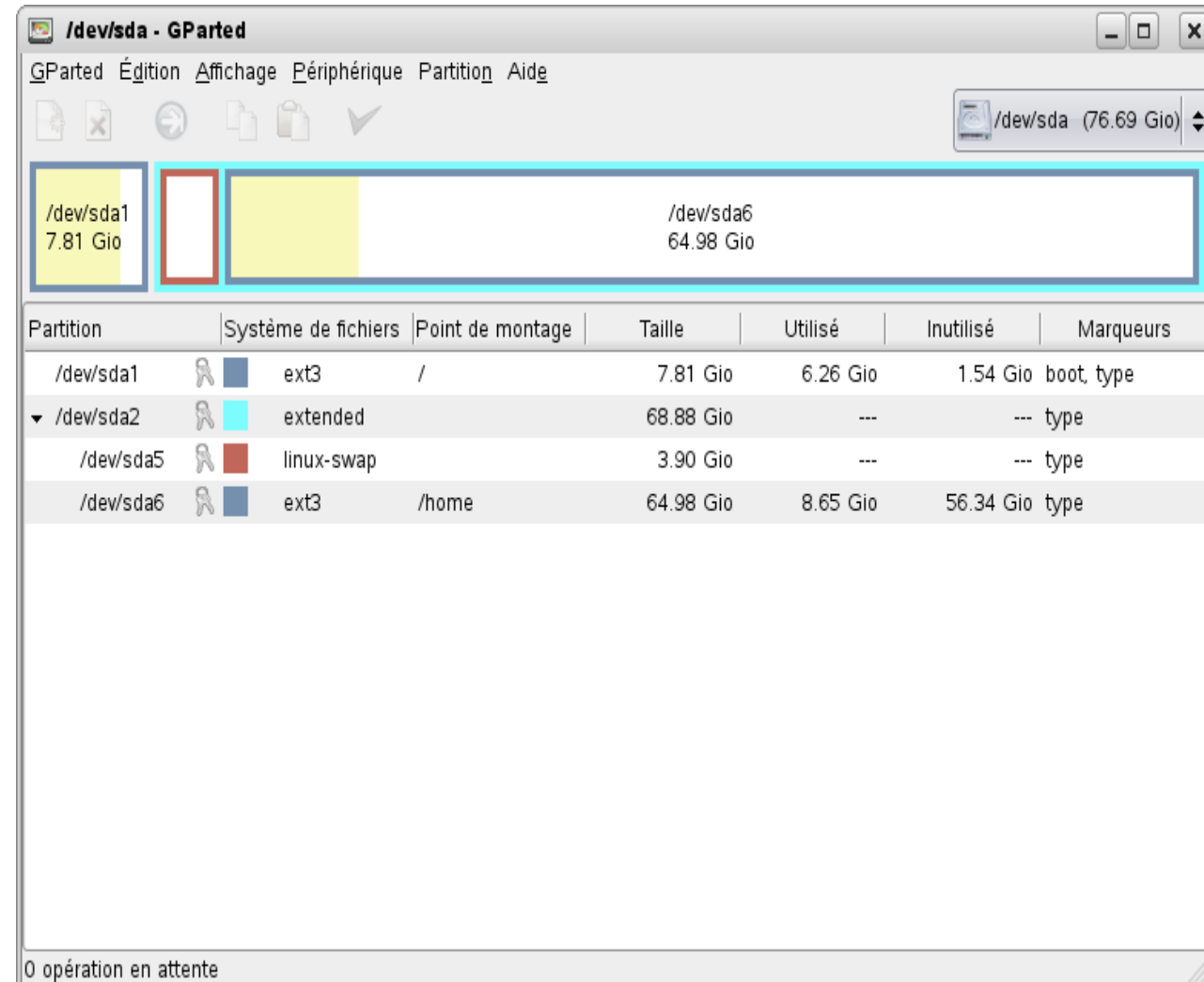
Avec spécification du type FS

```
mount -t ext4 /dev/sdc3 /media/stock
```

# RAPPEL ET APERÇU SU LE SYSTÈME

## PARTITIONNEMENT DU DISQUE

- Pour partitionner le disque, il existe 3 méthodes:
  1. sfdisk : ligne de commande (CLI)
  2. fdisk, cfdisk : menus texte (curses)
  3. gparted : interface graphique (gnome)



# RAPPEL ET APERÇU SU LE SYSTÈME

## COMMANDES DE GESTION DES FICHIERS

- La syntaxe générale des commandes sous linux est:

- La commande **cd** (change directory):

\$ *cd argument*  Accéder au répertoire *argument*



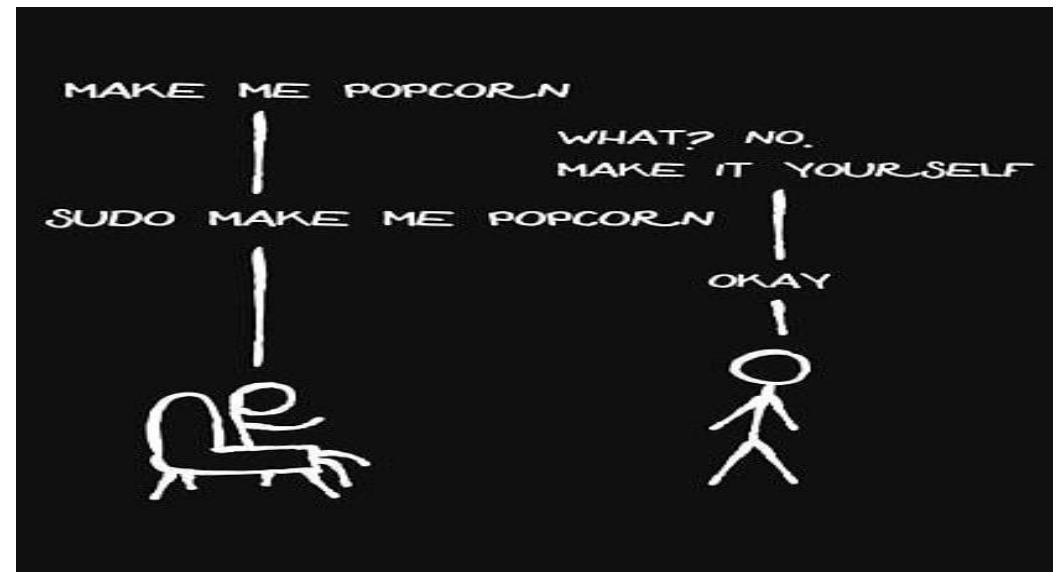
# LES UTILISATEURS

# LES UTILISATEURS

- GNU est un système **multi-utilisateurs** : plusieurs utilisateurs peuvent s'y connecter en même temps et partager des ressources
- Chaque utilisateur possède un nom (ex : cherkaoui) et un répertoire personnel dans /home (ex : /home/cherkaoui/)
- Le système peut théoriquement accueillir  $2^{32}$  utilisateurs
- On peut aussi mettre les utilisateurs dans des **groupes**

# LES UTILISATEURS

- Pour des raisons de sécurité, les utilisateurs ordinaires ne peuvent pas tout faire.
- Seuls les **super-utilisateurs** peuvent administrer le système
- La commande **sudo** permet d'exécuter une commande en tant que super-utilisateur



# LES UTILISATEURS

- Les utilisateurs sont identifiés par un nom et un UID (**U**ser **I**Dentifier)
  - Le super-utilisateur **root** a toujours l'UID 0
  - Utilisateurs « système » (UID < 1 000) : daemon, postfix, sshd, ...
  - Vrais utilisateurs (UID ≥ 1 000) : toto, marcel
- Le fichier **/etc/passwd** contient les utilisateurs
- Les groupes sont identifiés par un nom et un GID (**G**roup **I**Dentifier)
- Ils servent à définir des permissions (voir plus loin) de manière plus globale et simple.
- La commande **groups** affiche les groupes dont vous faites partie.
- Chaque utilisateur fait partie de son propre groupe

# LES UTILISATEURS

- Pour chaque utilisateur, le fichier `/etc/passwd` contient sept champs :
  - login
  - mot de passe ('x' pour les shadow passwords)
  - UID (User ID)
  - GID (Group ID, groupe principal)
  - Champ GECOS (nom complet, adresse, téléphones)
  - Répertoire personnel (« home dir »)
  - Le shell exécuté au login

```
root:x:0:0:Linux Torvalds,0,1 23,456:/root:/bin/bash
```



# LES UTILISATEURS

- Seul root peut lire/modifier le fichier `/etc/shadow`
- Pour chaque utilisateur, le fichier `/etc/shadow` contient le mot de passe de connexion et ses paramètres de validité :
  - login
  - mot de passe chiffré
  - Nombre de jours depuis le dernier changement de mot de passe: '0' l'utilisateur devra changer son mot de passe lors de la prochaine connexion
  - Âge minimum du mot de passe: combien de jours l'utilisateur doit-il garder son mot de passe avant de pouvoir le changer ? Si vous avez un "0", alors l'utilisateur peut le changer dès qu'il le souhaite
  - Âge maximum du mot de passe: combien de jours le mot de passe est-il valide ?
  - Avertissement: combien de jours avant que le mot de passe expire
  - Date d'expiration: quand le compte a-t-il été désactivé ? (date d'effet à partir de 01/01/1970)
  - Pas encore utilisé: s' il n'y a pas de valeur après le dernier ":"

```
mblanc:$1$QJ//btH...jL:13428:0:99999:7:::
```

# LES UTILISATEURS

○ \$ cat /etc/passwd

root:x:0:0:Linus Torvads:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/bin/sh

man:x:6:12:man:/var/cache/man:/bin/sh

lp:x:7:7:lp:/var/spool/lpd:/bin/sh

mail:x:8:8:mail:/var/mail:/bin/sh

gdm:x:106:111:Gnome Display

Manager:/var/lib/gdm:/bin/false

acox:x:1000:1000:Alan Cox,Kernel

St,0625081221,0474701221:/home/acox:/bin/bash

○ \$ cat /etc/shadow

root:!:13428:0:99999:7:::

daemon:!:13428:0:99999:7:::

bin:!:13428:0:99999:7::: sys:!:13428:0:99

sync:!:13428:0:99999:7:::

games:!:13428:0:99999:7:::

man:!:13428:0:99999:7::: lp:!:13428:0:99

mail:!:13428:0:99999:7::: gdm:!:13428:0:9

acox:\$1\$QN//abU4\$nhUser1ZjoAb3nx23

z.:13428:0:99999:7:::

# LES UTILISATEURS

- Les groupes des utilisateurs sont décrits au sein du fichier `/etc/group`
- Chaque ligne contient 4 champs :
  - Le nom du groupe
  - Le mot de passe du groupe (ou x si le fichier `gshadow` existe)
  - l'ID du groupe
  - La liste des membres du groupe, séparés avec des `,`
- Chaque utilisateur possède en général un groupe à son nom (Unique Private Group), c'est son groupe primaire
- Chaque utilisateur peut aussi appartenir à `n` groupes secondaires
- Les groupes systèmes permettent souvent de permettre aux utilisateurs de manipuler des devices (dialout, fax, audio, ...)

# LES UTILISATEURS

```
$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:acox,ttso,ltorvalds
dialout:x:20:cupsys,acox,ttso,ltorvalds
fax:x:21:hugo,corentin
cdrom:x:24:haldaemon,acox,ttso,ltorvalds
floppy:x:25:haldaemon,acox,ttso,ltorvalds
tape:x:26:acox,ttso,ltorvalds
sudo:x:27:
audio:x:29:ttso,ltorvalds
www-data:x:33:
backup:x:34:
shadow:x:42:
utmp:x:43:
video:x:44:acox,ttso,ltorvalds
sasl:x:45:
plugdev:x:46:haldaemon,acox,ttso,ltorvalds
acox:x:1000:
ttso:x:1001:
ltorvalds:x:1002:
$
```

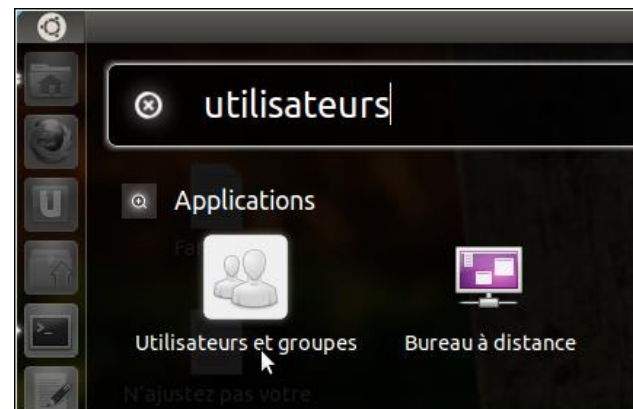
# LES UTILISATEURS

- Il existe généralement deux méthodes pour gérer les utilisateurs et les groupes:

- Via ligne de commande (CLI) en se servant des principales commandes suivantes:

useradd, usermod, userdel	gestion des comptes utilisateur
groupadd, groupmod, groupdel	gestion des groupes
pwck, grpck	vérification des fichiers
chfn, id, groups, finger	utilitaires divers

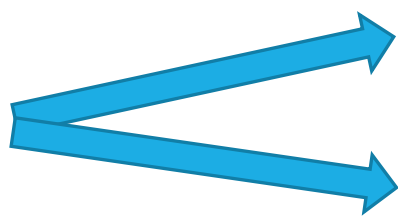
- Via un outil graphique (GUI)

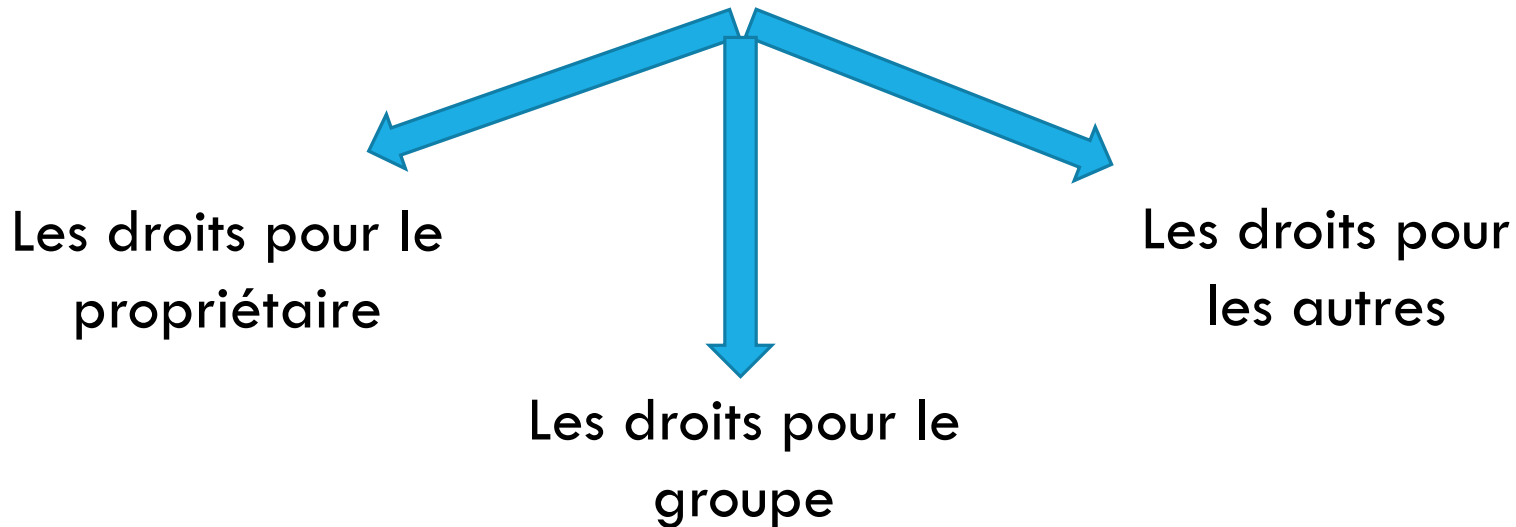




# LES DROITS D'ACCÈS

# LES DROITS D'ACCÈS

- Chaque fichier ou répertoire possède:
  - Un propriétaire
  - Un groupe propriétaire
- Chaque fichier ou répertoire possède 3 listes de droits :

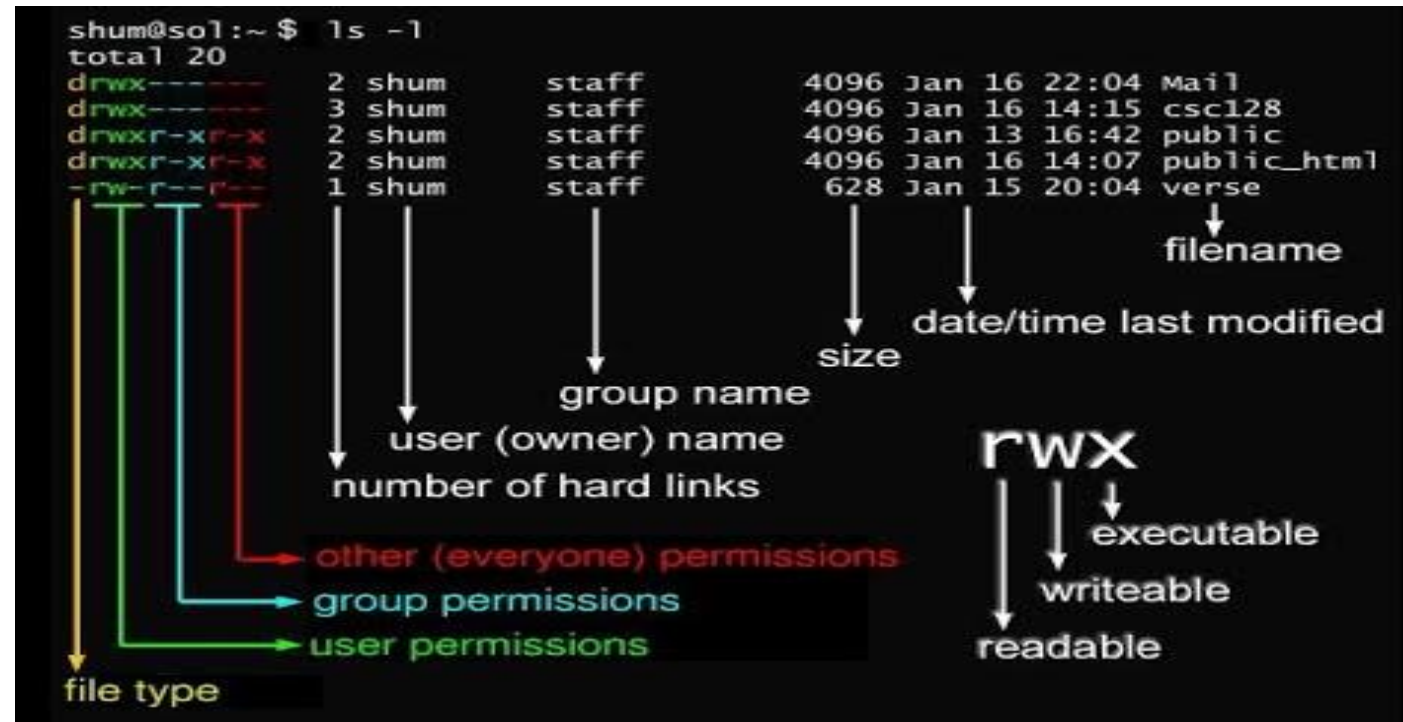
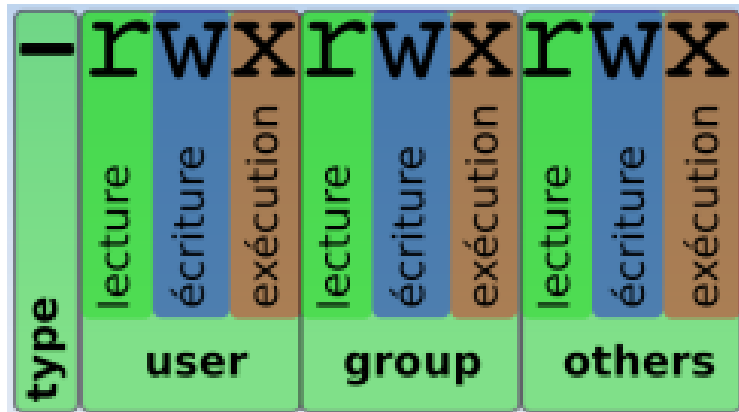


# LES DROITS D'ACCÈS NOTATIONS

- Chaque liste donne les opérations possibles pour cet utilisateur :
  - **Lecture (r) :**
    - Fichier : donne la possibilité de lire le contenu du fichier
    - Répertoire : donne la possibilité de lire le contenu d'un répertoire (donc la liste des fichiers qu'il contient)
  - **Écriture (w):**
    - Fichier : permet d'écrire dans ce fichier
    - Répertoire : permet d'y créer/renommer/supprimer des fichiers
  - **Exécution (x):**
    - Fichier : permet d'exécuter ce fichier
    - Répertoire : permet d'y 'rentre' (cd) et de voir son contenu



# LES DROITS D'ACCÈS



Type	Droits	Destinataire
b : Block device	r : Read	u : User
c : Character device file	w : Write	g : Group
d : Répertoire (Directory)	x : Execute	o : Others
l : Lien symbolique	X: Special execute	a : All
s : Socket	s : SUID bit	
- : Fichier normal	t : Sticky bit	

# LES DROITS D'ACCÈS chmod

- La commande **chmod** permet de gérer les droits d'accès à un fichier/répertoire selon deux modes:

- Mode littéral:

**chmod** destinataire(s) opération droits

destinataire: u(**u**ser), g(**g**roup), o(**o**thers)

opérations: + (ajouter), - (supprimer), = (mettre à la valeur)

droits: r (**r**ead), w (**w**rite), x (**x**ecute)

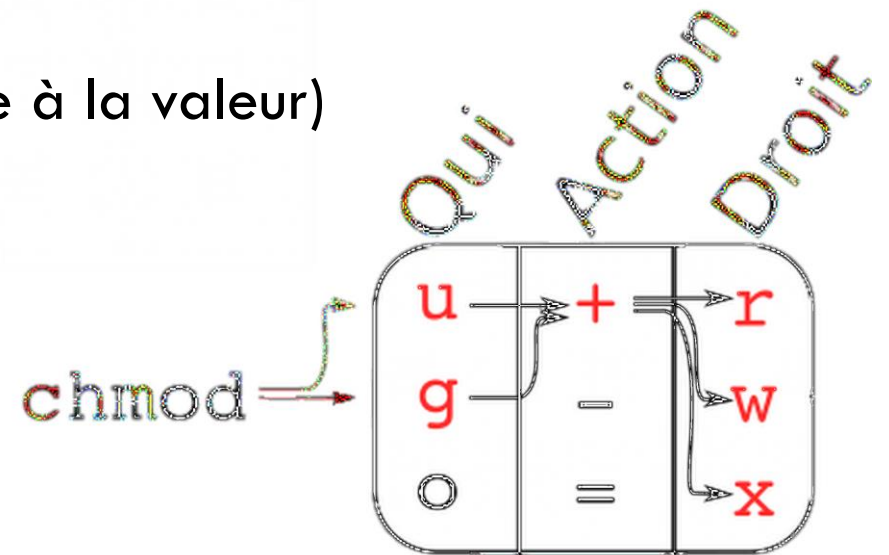
## Exemples:

```
$ chmod ugo+rwxfichier1 fichier2
```

```
$ chmod u+rw, g+r, orwxfichier3
```

```
$ chmod ug=rwx,o=rx fichier4
```

```
$ chmod a+rx,u+w repertoire
```



# LES DROITS D'ACCÈS chmod

- Mode octal:

**chmod** peut aussi fonctionner en mode octal en désignant trois bits par groupe correspondant à r, w et x

`$chmod 755`  $\Leftrightarrow$  `chmod u=rwx,g=rx,o=rx`

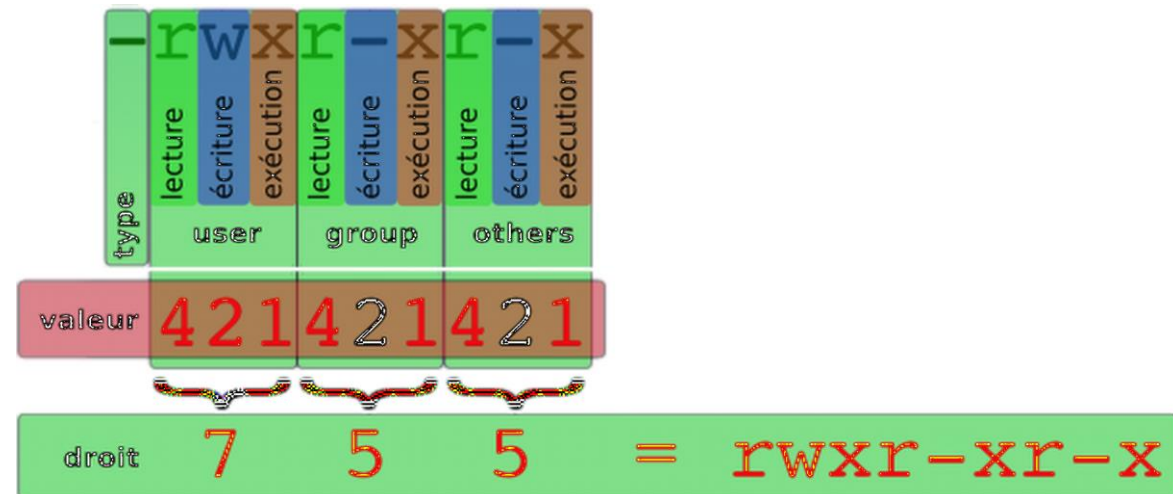
- Exemples:

`$ chmod 755 fichier1 fichier2`

`$ chmod 700 fichier3`

`$ chmod 750 fichier4`

`$ chmod 555 repertoire`



# LES DROITS D'ACCÈS chown

- La commande **chown** permet de changer le propriétaire ou le groupe propriétaire d'un fichier

\$ chown propriétaire:groupe fichier ...

- propriétaire : nouveau propriétaire du fichier ou répertoire
- groupe : nouveau groupe propriétaire du fichier ou répertoire
- fichier ... : fichiers ou répertoires dont il faut changer la propriété

- Si le groupe est omis, **chown** ne change que le propriétaire:

\$ chown root /etc/passwd

- Si propriétaire est omis, **chown** ne change que le groupe:

\$ chown :cdrom /dev/cdrom

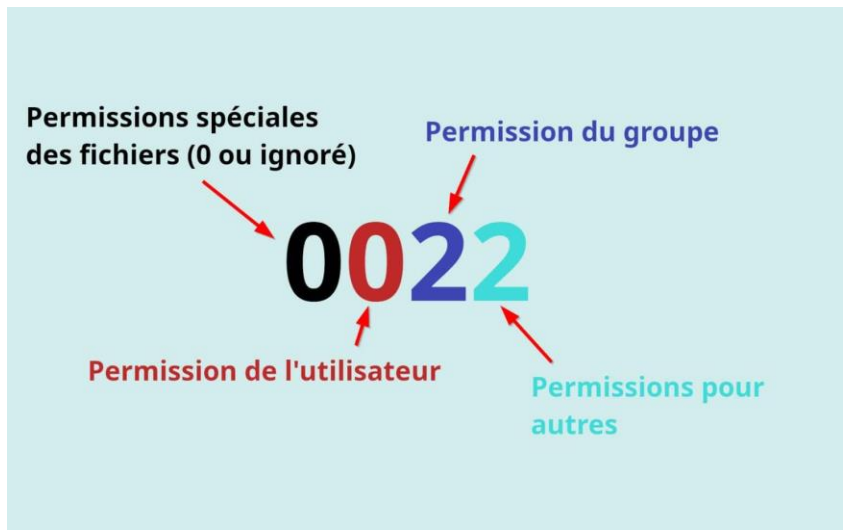
# LES DROITS D'ACCÈS UMASK

- Les droits de création par défaut d'un fichier/répertoire lors de sa création sont défini par la commande **umask**

\$ umask <valeur>

- pour calculer les bits d'autorisation pour un nouveau fichier ou un nouveau répertoire, nous soustrayons simplement la valeur umask de la valeur par défaut, comme ceci :

- $666 - 022 = 644$  — pour les fichiers
- $777 - 022 = 755$  — pour les répertoires



# LES DROITS D'ACCÈS UMAUSER1

- Les valeurs des autorisations en mode littéral:

```
$ umask u=rwx,g=rwx,o=rwx
```

```
$ umask a=rwx
```

- Exemple:

```
$ umask 0022
```

```
$ umask
```

```
0022
```

```
$ umask -S
```

```
u=rwx,g=rx,o=rx
```


```
$ touch fichier; mkdir repertoire; ls -ld fichier repertoire
```

```
-rw-r--r-- 1 tux zoo ... fichier
```

```
drwxrwxr-x 2 tux zoo ... repertoire
```

# LES DROITS D'ACCÈS

## SUID | GUID | STICKY BITS

- SUID: ou Set Owner User ID est un indicateur de bit de permission qui s'applique aux exécutable.
- GUID: est similaire au SUID. S'il s'agit d'un exécutable, il s'exécute avec les permissions du groupe. Si c'est un répertoire, il en résulte que tous les nouveaux fichiers et répertoires créés appartiennent au groupe.
- SUID/SGID bits (s)  celui qui exécute prend temporairement l'identité du propriétaire ou du groupe propriétaire du fichier en question.

# LES DROITS D'ACCÈS

SUID | GUID | STICKY BITS

## ■ Exemple:

```
$ ls -l fichier
```

```
-rwxr-xr-x 1 User1 User1 0 2022-01-16 23:02 fichier
```

```
$ chmod u+s fichier
```

```
$ ls -l fichier
```

```
-rwsr-xr-x 1 User1 User1 0 2022-01-16 23:02 fichier
```

```
$ chmod g+s fichier
```

```
$ ls -l fichier
```

```
-rwsr-sr-x 1 User1 User1 0 2022-01-16 23:02 fichier
```



# LES DROITS D'ACCÈS

## SUID | GUID | STICKY BITS

- Les sticky bits s'appliquent aux répertoires. Lorsque les sticky bits sont définis sur un répertoire particulier, tout utilisateur ayant accès au répertoire et à son contenu ne peut supprimer que ses propres fichiers et ne peut pas toucher ou supprimer des fichiers appartenant à quelqu'un d'autre. Les sticky bits sont généralement utilisés lors de l'utilisation d'un dossier partagé.

- Exemple:

```
$touch fichier
```

```
$ chmod a+rxw fichier
```

```
$ chmod +t .
```

```
$ sudo chown root . fichier
```

```
$ ls -la
```

```
total 12
```

```
drwxr-xr-t 2 root User1 4096 2022-0-117 00:17 .
```

```
drwxr-xr-x 3 User1 User1 4096 2022-01-16 23:02 ..
```

```
-rwxrwxrwx 1 root User1 0 2022-01-17 00:17 fichier
```

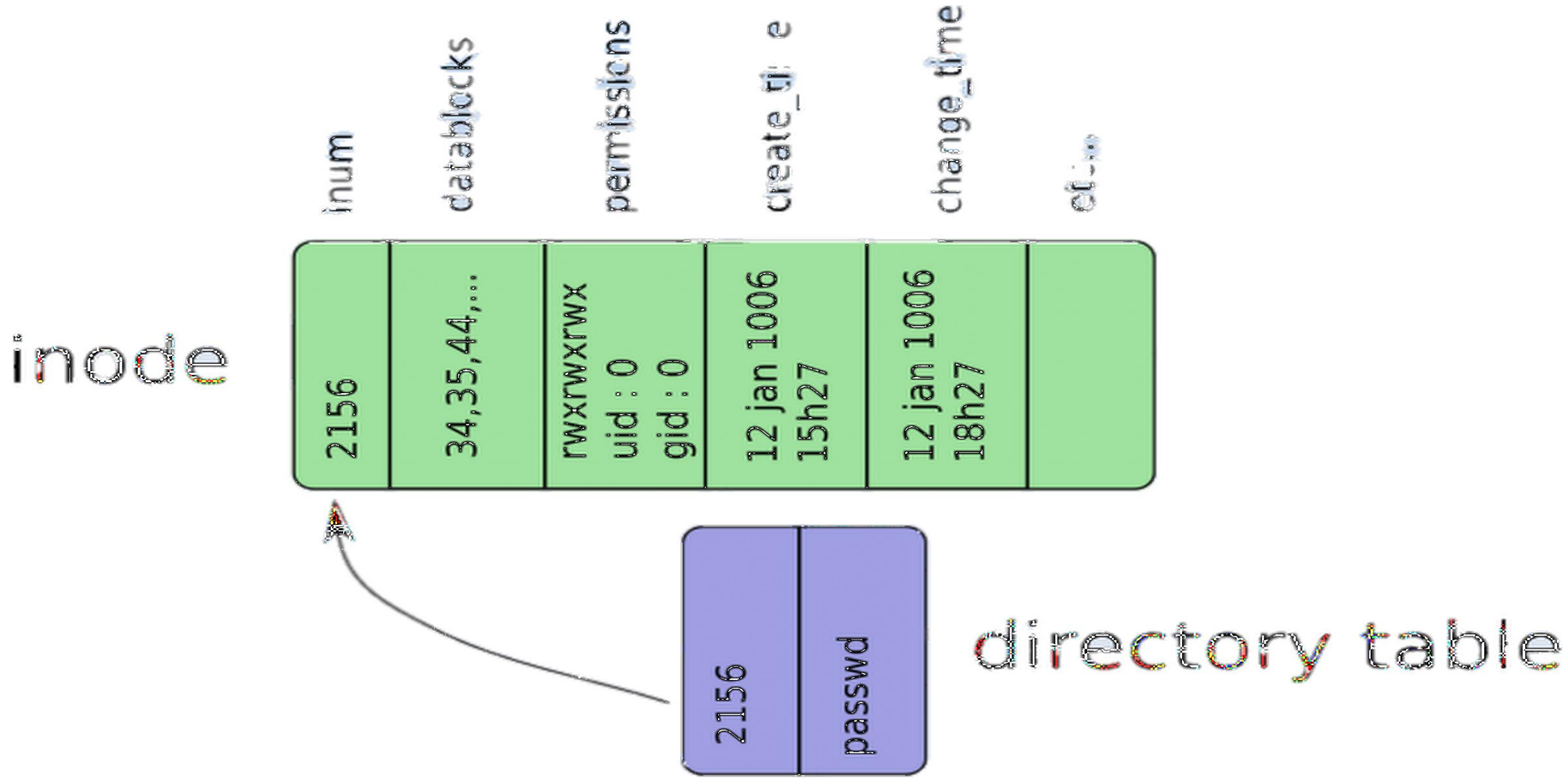
```
$ rm -f fichier
```

```
rm: ne peut enlever `fichier': Permission non accordée
```

# LES DROITS D'ACCÈS INODES

- Dans un FS Unix/Linux, chaque fichier est représenté par un **inode**
- Chaque fichier correspond un numéro d'inode dans le FS dans lequel il réside, unique au périphérique sur lequel il est situé.
- Le FS contient une table d'association "nom de fichier"  $\Leftrightarrow$  "inode"
- L'inode contient toutes informations nécessaires concernant le fichier :
  - numéro d'inode
  - permissions, propriétaire
  - dates (création, accès, modification...)
  - références vers les blocs de données

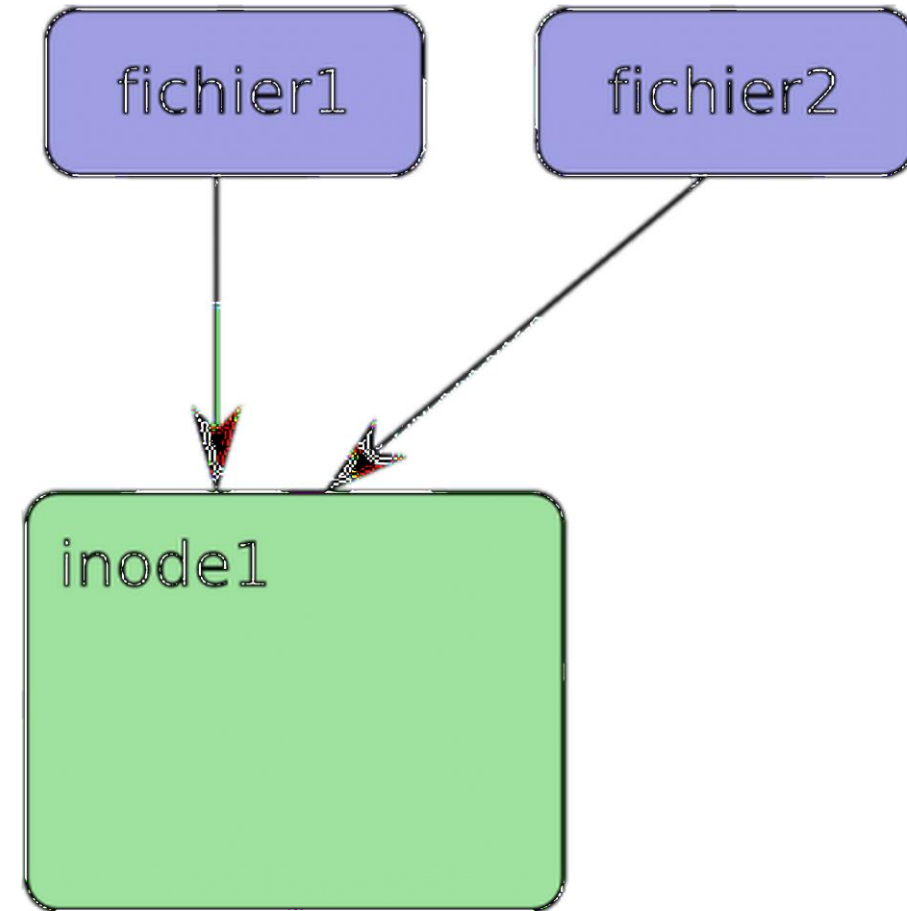
# LES DROITS D'ACCÈS INODES



# LES DROITS D'ACCÈS

## HARD LINKS

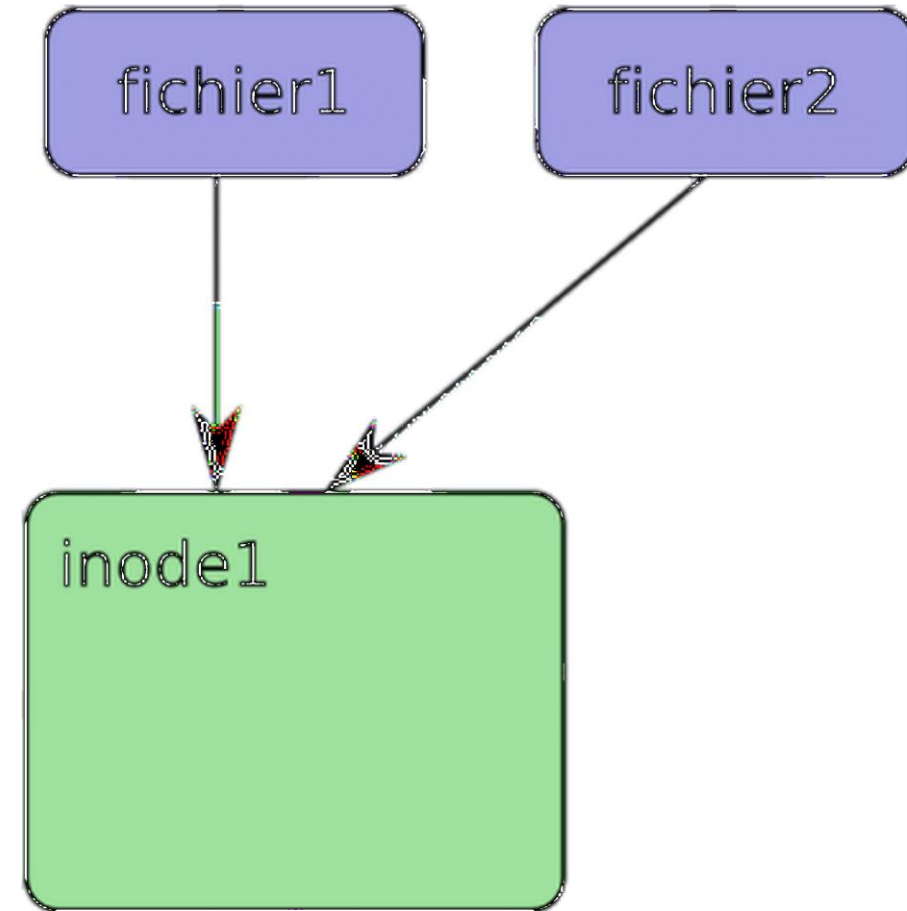
- Un hard link est simplement est une copie miroir du fichier original.
- Il peut y avoir plusieurs hard-links sur un inode (et donc plusieurs « noms » pour un même contenu)
- Les inodes existent tant qu'il y a au moins un hard-link pointant dessus.
- Même si vous supprimez le fichier original, le hard link aura toujours les données du fichier original.



# LES DROITS D'ACCÈS

## SOFT LINKS

- Les soft links (symbolic links / symlinks) pointent vers un autre « fichier » de la DT
- Le lien symbolique n'existe que par l'objet qu'il pointe : sans lui, plus de fichier.
- Si vous supprimez le fichier original, le soft link n'a aucune valeur, car il pointe vers un fichier inexistant.



# LES DROITS D'ACCÈS

## SOFT LINKS VS HARD LINKS

Soft link	Hard link
peuvent fonctionner entre les filesystems	ne peuvent fonctionner que sur un même filesystem (la tables des inodes est spécifique au filesystem)
permet de faire des liens entre les répertoires	ne peut pas lier des répertoires
a un numéro d'inode et des permissions de fichier différents de ceux du fichier original	a le même numéro d'inode et les mêmes permissions que le fichier original
les permissions ne seront pas mises à jour	les permissions seront mises à jour si nous changeons les permissions du fichier source
n'a que le chemin du fichier original, pas le contenu.	a le contenu réel du fichier original, de sorte que vous pouvez toujours voir le contenu, même si le fichier original a été déplacé ou supprimé

# LES DROITS D'ACCÈS

## CRÉATION DES LIENS

- La commande **ln** permet de créer des liens:

- crée un hard link (destination) pointant sur l'inode original

`$ ln <original> <destination>`

- crée un soft link (destination) pointant sur l'original

`$ ln -s <original> <destination>`

- La commande **stat** permet de connaître toutes les infos d'un inode:

`$ stat <fichier>`

 Affiche le contenu de l'inode pointé par <fichier>

# LES DROITS D'ACCÈS

## CRÉATION DES LIENS

### ■ Exemple: Softlink

```
$ echo "Bonjour les informaticiens" > source.file
```

```
$ cat source.file
```

```
Bonjour les informaticiens
```

```
$ ln -s source.file softlink.file
```

```
$ cat softlink.file
```

```
Bonjour les informaticiens
```

```
$ ls -lia
```

```
total 12
```

```
11665675 drwxrwxr-x  2 User1 User1 4096 Oct 22 11:39 .
```

```
4325378 drwxr-xr-x 37 User1 User1 4096 Oct 22 11:39 ..
```

```
11665731 lrwxrwxrwx  1 User1 User1  11 Oct 22 11:39 softlink.file -> source.file
```

```
11665692 -rw-rw-r--  1 User1 User1  21 Oct 22 11:39 source.file
```

```
$ rm source.file
```

```
$ cat softlink.file
```

```
cat: softlink.file: No such file or directory
```



# LES DROITS D'ACCÈS

## CRÉATION DES LIENS

### ■ Exemple: Hardlink

```
$ echo "Bonjour les informaticiens" > source.file
```

```
$ cat source.file
```

```
Bonjour les informaticiens
```

```
$ ln source.file hardlink.file
```

```
$ cat hardlink.file
```

```
Bonjour les informaticiens
```

```
$ ls -lia
```

```
total 16
```

```
11665675 drwxrwxr-x 2 User1 User1 4096 Oct 22 11:58 .
```

```
4325378  drwxr-xr-x 37 User1 User1 4096 Oct 22 11:39 ..
```

```
11665692 -rw-rw-r-- 2 User1 User1 21 Oct 22 11:57 hardlink.file
```

```
11665692 -rw-rw-r-- 2 User1 User1 21 Oct 22 11:57 source.file
```

```
$ rm source.file
```

```
$ cat hardlink.file
```

```
Bonjour les informaticiens
```



# LES ENTRÉES/SORTIES

# LES ENTRÉES/SORTIES

Un programme a pour rôle de traiter des données. Il peut donc prendre des données, les manipuler et renvoyer le résultat de ces manipulations. Les commandes linux sont des programmes( ou bien processus) destiner à être exécutés par l'utilisateur:

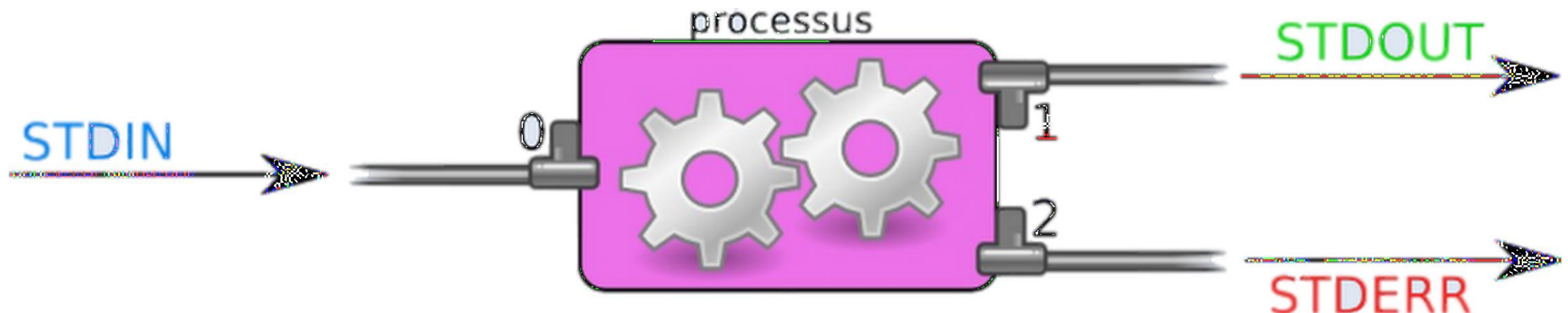
- Par défaut tout processus en linux possède :
  - 1 Entrée
  - 2 Sorties
- Ces E/S sont distinctes pour chaque processus
- Pour les programmes interactifs (comme les shells) :
  - les entrées proviennent du clavier
  - les sorties s'affichent sur l'écran

# LES ENTRÉES/SORTIES

**STDIN** (entrée standard) : ce qui est envoyé vers le processus

**STDOUT** (sortie standard) : ce qui est envoyé par le processus

**STDERR** (sortie erreur standard) : les erreurs renvoyés par le processus



# LES ENTRÉES/SORTIES CAT

- Ces entrées/sorties standard sont en fait des noms symboliques, correspondant à des « descripteurs de fichiers » :

**STDIN** (INput) : descripteur de fichier 0

**STDOUT** (OUTput) : descripteur de fichier 1

**STDERR** (ERRor) : descripteur de fichier 2

- cat : copie STDIN (ou un fichier) sur STDOUT (écrit les erreurs, s'il y a, sur STDERR)

```
user@host:~$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      michel

# The following lines are desirable for IPv6 capable hosts
::1          ip6-localhost ip6-loopback
fe00::0      ip6-localnet

user@host:~$ cat /etc/bidon
cat: /etc/bidon: Aucun fichier ou répertoire de ce type
```

# LES ENTRÉES/SORTIES REDIRECTION

- Les E/S peuvent être redirigées de ou vers un fichier:

- processus < fichier

**STDIN** provient du fichier (et non plus du clavier)

- processus > fichier

**STDOUT** est écrit dans fichier (et non plus sur le terminal)

- processus 2> fichier

**STDERR** est écrit dans fichier (et non plus sur le terminal)

- processus > fichier1 2> fichier2

**STDOUT** est écrit dans fichier1 et **STDERR** dans fichier2

# LES ENTRÉES/SORTIES REDIRECTION

- Donner le résultat des suites des commandes suivantes:

```
$ cat < /etc/hostname
```

```
pluton
```

```
$ cat /etc/hostname
```

```
$ cat /etc/hostname > /tmp/test
```

```
$ cat /tmp/test
```

```
$ cat /etc/hostname > /dev/null
```

```
$ cat < /etc/hostname > /dev/null
```

```
$ cat /etc/hostname
```

```
$ cat /etc/portnaouak
```

```
$ cat /etc/portnaouak 2> /dev/null
```

```
$ cat /etc/hostname /etc/portnaouak > out.txt 2> err.txt
```

```
$ cat out.txt
```

```
$ cat err.txt
```

# LES ENTRÉES/SORTIES REDIRECTION

```
user@pluton:~$ cat < /etc/hostname
pluton
user@pluton:~$ cat /etc/hostname
pluton
user@pluton:~$ cat /etc/hostname > /tmp/test
user@pluton:~$ cat /tmp/test
pluton
user@pluton:~$ cat /etc/hostname > /dev/null
user@pluton:~$ cat < /etc/hostname > /dev/null
user@pluton:~$ cat /etc/hostname
pluton
user@pluton:~$ cat /etc/portnaouak
cat: /etc/portnaouak: Aucun fichier ou répertoire de ce type
user@pluton:~$ cat /etc/portnaouak 2> /dev/null
user@pluton:~$ cat /etc/hostname /etc/portnaouak > out.txt 2> err.txt
user@pluton:~$ cat out.txt
pluton
user@pluton:~$ cat err.txt
cat: /etc/portnaouak: Aucun fichier ou répertoire de ce type
```



# LES ENTRÉES/SORTIES REDIRECTION

- Donner le résultat des suites des commandes suivantes:

```
$ cat /etc/hostname
```

```
$ cat /etc/hostname >> /tmp/test
```

```
$ cat /tmp/test
```

```
$ cat /etc/hostname > /tmp/test
```

```
$ cat /tmp/test
```

```
$ cat /etc/hostname > /tmp/test
```

```
$ cat /tmp/test
```

```
$ cat /etc/hostname >> /tmp/test
```

```
$ cat /tmp/test
```

# LES ENTRÉES/SORTIES REDIRECTION

```
user@pluton:~$ cat /etc/hostname
pluton
user@pluton:~$ cat /etc/hostname >> /tmp/test
user@pluton:~$ cat /tmp/test
pluton
pluton
user@pluton:~$ cat /etc/hostname > /tmp/test
user@pluton:~$ cat /tmp/test
pluton
user@pluton:~$ cat /etc/hostname > /tmp/test
user@pluton:~$ cat /tmp/test
pluton
user@pluton:~$ cat /etc/hostname >> /tmp/test
user@pluton:~$ cat /tmp/test
pluton
pluton
```

# LES ENTRÉES/SORTIES REDIRECTION

- Plusieurs "devices" (fichiers dans /dev) ont une vocation particulière :
  - /dev/null  
trou noir annihilant tout ce qui lui est envoyé
  - /dev/zero  
envoie des zéros ad-vitam
  - /dev/random /dev/urandom  
fournisseurs officiels de hasard
  - /dev/full  
dispositif hypochondriaque : se plaint toujours (d'être plein)

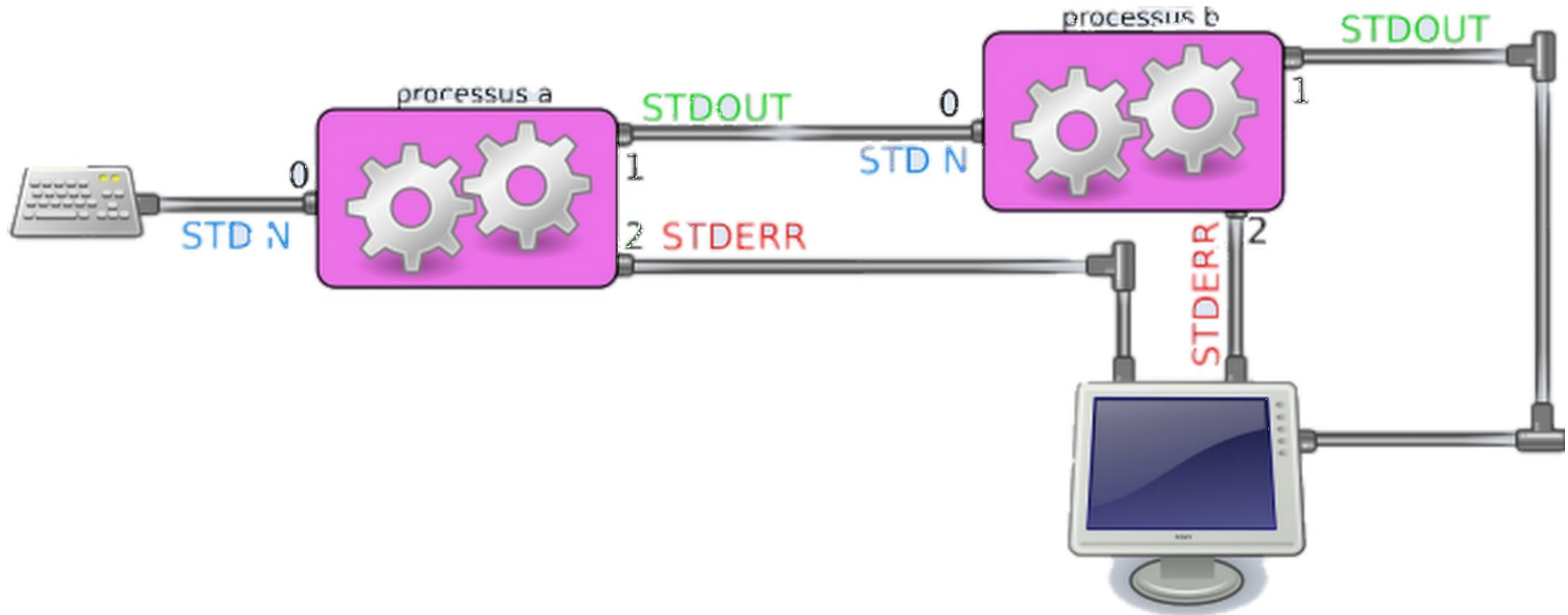
# LES ENTRÉES/SORTIES REDIRECTION

## ■ Exercice:

- 1) Copier le contenu de `/etc/passwd` dans le fichier `/tmp/users.txt`
- 2) Ecrire «linus» à la fin de `users.txt`
- 3) Vider le fichier `users.txt`
- 4) Remplir `users.txt` de "zéros" (utiliser le dispositif fournisseur de zéros : `/dev/zero`)
- 5) Rediriger l'erreur standard de `'ls -lR /'` dans `/tmp/users.txt`
- 6) Vider le fichier `users.txt` (d'une autre manière qu'en 3)

# LES ENTRÉES/SORTIES PIPES

- Les «pipes» (pipelines) permettent d'envoyer la sortie d'une commande (**STDOUT**) à l'entrée d'une autre (**STDIN**) :



# LES ENTRÉES/SORTIES PIPES

- On trouve très souvent la commande **grep** au milieu de pipelines
- **grep** permet de n'afficher une ligne que si elle contient une chaîne de caractères donnée. Sa syntaxe est :

```
$ grep <chaîne> <fichier>
```

affiche les lignes de fichier contenant "chaîne"

```
$ grep <chaîne>
```

affiche les lignes lues sur l'entrée standard (**STDIN**) contenant "chaîne"

# LES ENTRÉES/SORTIES PIPES

- Exemple: Exécutez les commandes suivantes sur votre terminal et donnez suite à chaque commande:

```
$ cat /etc/passwd | grep root
```

```
$ ls | grep test
```

```
$ ip link | grep UP
```

```
$ ip link | grep UP > uplinks.txt
```

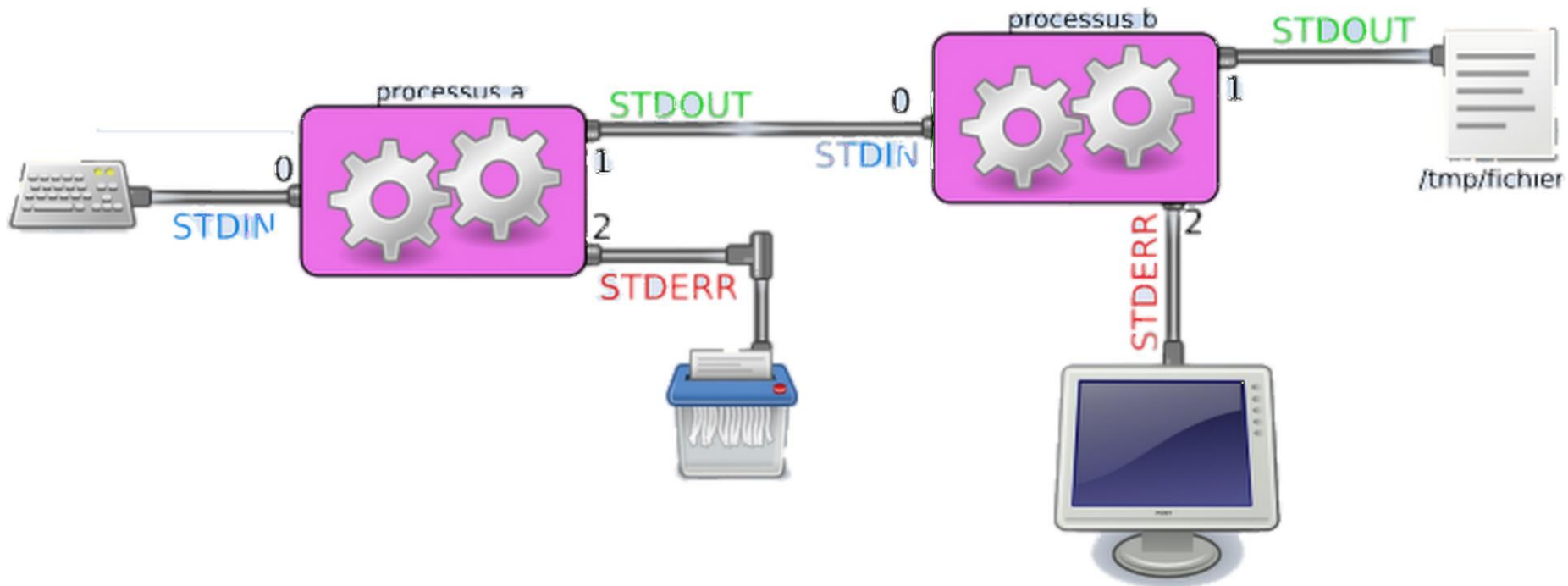
```
$ cat uplinks.txt | grep eth
```

```
$ ip link | grep UP | grep eth
```

```
$ history | awk '{ print $2 }' | sort | uniq -c | sort -nr | head 10
```

# LES ENTRÉES/SORTIES PIPES

- Les pipes et les redirections peuvent être combinées:



➡ `$ a 2> /dev/null | b > /tmp/fichier`



# LES ENTRÉES/SORTIES PIPES

- Attention à l'ordre des redirections et des pipelines:

- `$ a | b 2> c`

- le résultat de la commande a est passé à b

- les erreurs de b sont redirigées dans le fichier c

- `$ a 2> c | b`

- le résultat de la commande a est passé à b

- les erreurs de a sont redirigées dans le fichier c



# SCRIPTS SHELL

# SCRIPTS SHELL BASH

- Un script shell permet d'automatiser une série d'opérations. Il se présente sous la forme d'un fichier contenant une ou plusieurs commandes qui seront exécutées de manière séquentielle.
- Le shebang, représenté par **#!/**, est un en-tête d'un fichier texte qui indique au système d'exploitation que ce fichier n'est pas un fichier binaire mais un script.
- **#!/bin/bash** : indique au système qu'il s'agit d'un script qui sera interprété par **bash** on placera le shebang sur la première ligne du fichier.
- Pour créer un script, il suffit d'écrire les commandes que l'on souhaite dans un fichier. Par exemple dans un fichier **nom\_script.sh** on peut écrire une première instruction à exécuter.
- Afin d'exécuter votre script, il faut lui attribuer les droits d'exécution.

# SCRIPTS SHELL BASH

```
#!/bin/bash  
# firstScript.sh  
echo "Hello World"  
exit
```

- Ensuite on peut lancer le script dans la console, comme si c'était une commande, en indiquant le chemin de celui-ci.

```
$ ./firstScript.sh
```

- Exemple:

On souhaite écrire un script qui sera capable de copier tout les fichiers .png se trouvant dans le dossier camera vers un dossier photos, et de donner les droits en lecture/écriture uniquement au propriétaire des nouveaux fichiers. Enfin le script liste les fichiers en question pour afficher les permissions.

**Indice:** Créer un dossier Camera avec deux sous-dossier: photos et Videos. Puis créer plusieurs fichiers avec des extensions .mp4 et .jpg avant de créer le script.

# SCRIPTS SHELL BASH

- Solution:

```
#!/bin/bash  
cp camera/*.png photos/  
chmod 600 photos/*  
ls -l photos
```

# SCRIPTS SHELL VARIABLES

- Une **variable** est un emplacement mémoire utilisé dans un programme ou un script pour conserver une valeur et pouvoir l'utiliser par la suite.
- Que ce soit dans un script ou directement dans la console, il est possible de stocker des valeurs dans des variables:

NomVariable = "Valeur"

- La commande **echo** permet d'afficher du texte ou une variable dans le terminal. Ici on fait appel à la variable "NomVariable", tout simplement à l'aide du symbole "\$" suivi du nom de la variable

echo \$NomVariable

# SCRIPTS SHELL VARIABLES

- Il est possible de récupérer le résultat d'une commande dans une variable. Pour cela il suffit d'entour la commande avec un `$()`.

```
$ resultat=$(ls -l fichier)
```

```
$ echo $resultat
```

```
-rwxr-xr-x 1 User1 User1 69 9 déc. 16:50 fichier
```

# SCRIPTS SHELL VARIABLES

## ■ Les variables prépositionnées:

- `$0` : nom du script. Plus précisément, il s'agit du paramètre 0 de la ligne de commande
- `$1`, `$2`, ..., `$9` : respectivement premier, deuxième, ..., neuvième paramètre de la ligne de commande
- `$*` : tous les paramètres vus comme un seul mot
- `$@` : tous les paramètres vus comme des mots séparés : “`$@`” équivaut à “`$1`” “`$2`”
- `$#` : nombre de paramètres sur la ligne de commande
- `$-` : options du shell
- `$?` : code de retour de la dernière commande
- `$$` : PID du shell
- `$_` : PID du dernier processus lancé en arrière-plan
- `$_` : dernier argument de la commande précédente
- `${u:0:10}` : sous-chaine de la variable `u` à partir de l'offset 0 pour une longueur de 10



# SCRIPTS SHELL VARIABLES

- Exécuter ce script sur vos machines et donner le résultat:

```
#!/bin/bash

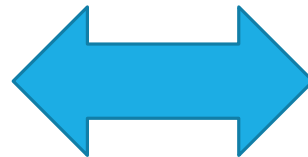
echo "Nom du script $0"
echo "premier paramètre $1"
echo "second paramètre $2"
echo "PID du shell " $$
echo "code de retour $?"

exit
```

# SCRIPTS SHELL VARIABLES

- Une interaction entre le script et l'utilisateur est possible:
  - La commande **echo** pose une question à l'utilisateur
  - La commande **read** lit les valeurs entrées au clavier et les stocke dans une variable à réutiliser.

```
#!/bin/bash  
echo "Question"  
read reponse  
echo $reponse  
exit
```



```
#!/bin/bash  
read -p "question" reponse  
echo $reponse  
exit
```

# SCRIPTS SHELL LES STRUCTURES CONDITIONNELLES

- En programmation, une structure conditionnelle est une structure qui permet, dans un script ou un programme, d'exécuter des instructions en fonction de conditions basées sur les valeurs de variables. Voici l'algorithme suivant:

SI "mavariabale" est plus grande que "12" ALORS

Afficher un message d'erreur

FIN SI

- Syntaxe bash:

```
if [ $mavariabale -gt 12 ] then
```

```
    echo "Error"
```

```
fi
```

# SCRIPTS SHELL

## LES STRUCTURES CONDITIONNELLES

- Il est possible de proposer une alternative si la condition première n'est pas respectée:

```
if [ $mavARIABLE -gt 12 ] then
    echo "Error "
else
    echo " OK"
fi
```

- Il est possible de complexifier les conditions en ajoutant plusieurs alternatives conditionnelles avec le mot clef **elif** (contraction de else if)

```
if [ $mavARIABLE -gt 12 ]
then
    echo "Error "
elif [ $mavARIABLE -lt 5 ]
then
    echo "Perfect"
else
    echo " OK"
fi
```

# SCRIPTS SHELL

## LES STRUCTURES CONDITIONNELLES

- Pour se servir de la structure conditionnelle IF-ELSE, Il existe de nombreux opérateur de comparaison des variables contenant du texte:

<code>\$texte1 = \$texte2</code>	Vérifie que la variable texte1 contient la même chose que texte2
<code>\$texte1 != \$texte2</code>	Vérifie que les deux variables sont différentes
<code>-z \$texte</code>	Vérifie si la variable est vide (pas de texte) ou inexistante.
<code>-n \$texte</code>	Vérifie si la variable est non vide

# SCRIPTS SHELL

## LES STRUCTURES CONDITIONNELLES

- D'autres permettent la comparaison de nombres:

<code>\$num1 -eq \$num2</code>	Vérifie que les 2 nombres sont égaux
<code>\$num1 -ne \$num2</code>	Vérifie que les 2 nombres ne sont pas égaux
<code>\$num1 -lt \$num2</code>	Vérifie si num1 est inférieur (<) à num2
<code>\$num1 -le \$num2</code>	Vérifie si num1 est inférieur ou égal (<=) à num2
<code>\$num1 -gt \$num2</code>	Vérifie si num1 est supérieur (>) à num2
<code>\$num1 -ge \$num2</code>	Vérifie si num1 est supérieur ou égal (>=) à num2

- Enfin certains permettent de faire des tests sur des fichiers:

<code>-e \$nomfichier</code>	Vérifie si le fichier existe
<code>-d \$nomfichier</code>	Vérifie que le fichier est un répertoire.
<code>-f \$nomfichier</code>	Vérifier que le fichier est un fichier (donc pas un répertoire)

# SCRIPTS SHELL

## LES STRUCTURES CONDITIONNELLES

- Exercice d'application:

Ecrire un script qui permet de compter le nombre de fichier avec l'extension .png dans un sous-dossier fournit en tant qu'argument. Puis modifier le script pour vérifier si l'utilisateur a fournit l'argument cité auparavant.

- Solution:

```
#!/bin/bash
if [ -z $1 ]
Then
    echo "Erreur: Argument manquant"
else
    dossier=$1
    count=$(ls $dossier | wc -l)
    echo "Il y a $count fichiers dans le dossier: $dossier"
Fi
```

# SCRIPTS SHELL

## LES STRUCTURES CONDITIONNELLES

- Vous pouvez utiliser plusieurs instructions `if...elif` pour effectuer un branchement à plusieurs voies. Toutefois, ce n'est pas toujours la meilleure solution, surtout lorsque toutes les branches dépendent de la valeur d'une seule variable.
- Shell prend en charge l'instruction `case...esac` qui gère exactement cette situation, et ce de manière plus efficace que les instructions `if...elif` répétées.
- La syntaxe de base de l'instruction `case...esac` consiste à donner une expression à évaluer et à exécuter plusieurs instructions différentes en fonction de la valeur de l'expression. L'interpréteur vérifie chaque cas par rapport à la valeur de l'expression jusqu'à ce qu'il trouve une correspondance. Si rien ne correspond, une condition par défaut sera utilisée.



# SCRIPTS SHELL

## LES STRUCTURES CONDITIONNELLES

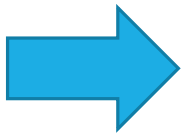
- Syntaxe:

```
case word in
    word1)
        Instruction(s) à exécuter si word1 correspond
        ;;
    word2)
        Instruction(s) à exécuter si word2 correspond
        ;;
    word3)
        Instruction(s) à exécuter si word3 correspond
        ;;
    *)
        Instruction(s) par défaut à exécuter dans le cas de non-correspondance
        ;;
esac
```

# SCRIPTS SHELL

## LES OPÉRATEURS LOGIQUES

- Les opérateurs logiques permettent de combiner deux conditions pour en former une nouvelle, plus complexe.
- Opérateur ET (AND): Il est possible d'assembler 2 conditions entre-elles pour former condition plus complexe qui n'est valide que si les deux sous-conditions sont vérifiées. C'est l'opérateur ET qui s'écrit **&&**. L'instruction sera exécutée que si les 2 conditions sont validées
- Opérateur OU (OR): L'opérateur OU, qui s'écrit **| |**, permet de faire la même chose que **&&** mais seulement si au moins une des conditions est correcte.



Les opérateurs logiques permettent de former des conditions plus complexe, et donc de réaliser des algorithmes plus poussés.

```
#!/bin/bash
if [ -z $1 ] || [ -z $2 ]
Then
    if [ -z $1 ]
    then
        echo "Erreur: Argument manquant! Vous devez fournir le dossier source "
    elif [ -z $2 ]
    then
        echo "Erreur: Argument manquant! Vous devez fournir le dossier destination "
    else
        dossierSource=$1
        dossierDestination=$2
        cp *.mp4 $1 $2
        echo "Déplacement réussit..."
        count=$(ls $dossierDestination | wc -l)
        echo "Il y a $count fichiers dans le dossier: $dossierDestination"
    Fi
```

# SCRIPTS SHELL

## LES BOUCLES

- En programmation une boucle est une structure qui permet de répéter plusieurs fois un même bloc d'instructions.
- La boucle **while**: (signifiant "tant que" en anglais) permet de répéter un bloc d'instruction tant qu'une condition est remplie(Vraie). Elle se base sur les mêmes types de conditions que les structures conditionnelles.
- **Syntaxe**: Pour écrire cette boucle, il suffit d'utiliser le mot clé **while** suivi des conditions à remplir pour continuer la boucle. Le bloc d'instructions est délimité par un **do** en début et un **done** à la fin.

```
#!/bin/bash
while [ -z $reponse ] || [ $reponse != 'bonjour' ]
do
    read -p 'Dites "bonjour" : ' reponse
done
```

# SCRIPTS SHELL LES BOUCLES

- La boucle **for** permet de répéter un bloc d'instructions un nombre prédéfini de fois.

- Syntaxe:

```
for ELEMENT in LISTE
do
    INSTRUCTIONS
done
```

- La boucle **while** est intéressante lorsque l'on ne sait pas à l'avance combien de fois on va refaire la boucle (on parle d'itérations). Cependant dans certains cas on sait à l'avance combien de tours de boucle on souhaite faire, par exemple si l'on souhaite appliquer un même traitement à une liste de variables. Dans ce cas, on utilise la boucle **for**.

# SCRIPTS SHELL LES BOUCLES

- Exemples: Boucle sur une variable

```
for nom in "Amine" "Salma" "Kamal"  
do  
    echo "Bonjour $nom"  
done
```



```
personnes="Amine Salma Kamal"  
for nom in $personnes  
do  
    echo "Bonjour $nom"  
done
```

- Exemples: Boucle sur une liste

```
for i in {1..10}  
do  
    echo "$i"  
done
```



```
for ((i=1; i<=10; i++)); do  
    echo "$i"  
done
```

# SCRIPTS SHELL LES BOUCLES

- **Exercice d'application:** Soit le script suivant:

On souhaite améliorer ce script afin de copier les fichiers .png un par un tout en affichant un message "Copie de CHEMIN\_DU\_FICHER" à chaque étape.

```
#!/bin/bash

if [ -z $1 ]
then
    dossier="photos"
else
    dossier=$1
fi

cp camera/*.png $dossier/
count=$(ls $dossier | wc -l)
echo "Il y a $count photos dans le dossier"
```

# SCRIPTS

- Solution:

```
#!/bin/bash

if [ -z $1 ]
then
    dossier="photos"
else
    dossier=$1
fi

for fichier in $(ls camera/*.png)
do
    echo "Copie de $fichier"
    cp $fichier $dossier/
done

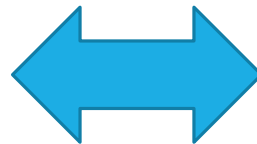
count=$(ls $dossier | wc -l)
echo "Il y a $count photos dans le dossier"
```



# SCRIPTS SHELL LES FONCTIONS

- Une fonction est un ensemble d'instructions qui permettent d'effectuer une tâche particulière, elle peut prendre(en entrée) des paramètres. On peut dire aussi qu'une fonction est vu comme un sous script.
- Pour définir une fonction il existe deux syntaxes en Bash. L'une comme l'autre permettent de faire exactement la même chose mais chacun peut avoir ses préférences notamment en terme de lisibilité du code.

```
function mafonction {  
    # Un traitement  
}
```



```
mafonction() {  
    # Un traitement  
}
```

# SCRIPTS SHELL LES FONCTIONS

- On peut simplement appeler une fonction quand elle a été définie. On utilise une syntaxe similaire à l'appel des commandes, à savoir le nom de la fonction éventuellement suivi de ses paramètres(s'ils existent).

Appel d'une fonction sans paramètre

```
#!/bin/bash
function bonjour {
    echo "Bonjour $(whoami)"
}

bonjour
```

Appel d'une fonction avec des paramètres

```
#!/bin/bash
function carre {
    echo "Le carré de $1 est $((($1*$1)))"
}

carre 3
```

# SCRIPTS SHELL EXERCICES

1. Écrivez un script Shell qui demande à l'utilisateur de saisir un mot, puis vérifiez s'il s'agit d'un palindrome (un mot qui se lit de la même manière de gauche à droite et de droite à gauche). La longueur du mot est obtenue avec `${#mot}`
2. Écrivez un script Shell qui demande à l'utilisateur de saisir le chemin d'un fichier et le chemin d'un répertoire, puis copie le fichier dans le répertoire spécifié.
3. Écrivez un script Shell qui affiche les fichiers modifiés dans un répertoire au cours des dernières 24 heures (Pensez à utiliser la commande `find`).



# LES TACHES PLANIFIÉES

# LES TACHES PLANIFIÉES

- La planification des tâches est gérée avec l'utilitaire **cron**. Il permet l'exécution périodique des tâches.
- **at**, **cron** et **systemd** sont les programmes Linux natifs qui permettent de planifier des tâches.
- **crontab** est le diminutif de '**chrono table**' mais peut être considéré comme une table de planification des tâches.
- Pour accéder à l'utilitaire **cron**, nous utiliserons **crontab**.
- Par défaut, tous les utilisateurs peuvent planifier l'exécution de tâches. C'est pourquoi chacun dispose de sa propre **crontab**, où il peut consigner les commandes à planifier.

Le service **cron** sert notamment pour :

- Les opérations d'administration répétitives
- Les sauvegardes
- La surveillance de l'activité du système
- L'exécution de programme

N.B.: Pour configurer un schedule, le système doit être réglé à l'heure exacte.

# LES TACHES PLANIFIÉES

## LA COMMANDE AT

- **at** est une commande Unix qui permet de programmer des commandes à n'exécuter qu'une fois(par opposition à **cron**) à un moment donné.
- La commande enregistrée hérite de l'environnement courant utilisé au moment de sa définition.
- **at** dispose d'options telles que :
  - **at -l** ou **atq** : affiche la liste des jobs introduits par la commande "at".
  - **at -r <JOB>** ou **atrm <JOB>** : efface le job identifié par son numéro de job.
  - **at** : sans paramètre, donne la ligne "Garbled time".

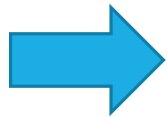
N.B.: Si la commande at n'est pas installée, procédez à l'installation via la commande: **#apt-get install at**

# LES TACHES PLANIFIÉES

## LA COMMANDE AT

- L'utilisation de **at** a son propre format. Lorsque vous souhaitez planifier un travail, vous tapez dans votre terminal:

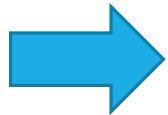
`$at [time] [date/jour]`



L'**heure** est **obligatoire**, mais la **date** est **facultative**. Si vous n'entrez rien, le système fera une supposition basée sur la date actuelle et l'heure du système.

- Par exemple pour créer une tâche après 5 minutes :

`$at now + 5 minutes`



Cela va ouvrir l'interface d'at, saisissez les différentes commandes.

Une fois terminé, tapez CTRL+D pour sauvegarder et quitter.

Quand la tâche est validée, le message <EOT> s'affiche et le numéro du travail 'Job 3'.

# LES TACHES PLANIFIÉES

## LA COMMANDE AT

- Expérimentez la commande **at** sur vos machines:

```
blackhole@blackhole-virtual-machine:~$ at now + 2 minutes
warning: commands will be executed using /bin/sh
at> echo "Première utilisation de la commande AT" > /home/blackhole/Bureau/Test
AT.txt
at> <EOT>
job 3 at Tue Jan 16 10:44:00 2024
blackhole@blackhole-virtual-machine:~$ at -l
3      Tue Jan 16 10:44:00 2024 a blackhole
```

- Vérifiez, si la tâche que vous avez planifié, a été bien exécuter



# LES TACHES PLANIFIÉES

## LA COMMANDE AT

- Nous pouvons planifier un travail sans l'invite interactive **at** en transmettant des commandes à **at** et en spécifiant la durée d'exécution.
- **Syntaxe:** `$echo "<commande à exécuter>" | at [durée d'exécution]`
- **Exemple:** Exécutez la commande suivante et observez son résultat  
`$echo "touch Ficher_Test.txt" | at now + 2 minutes`

# LES TACHES PLANIFIÉES

## LA COMMANDE AT

- Les expressions de temps de la commande **at**:

1

Spécifiez une expression de temps relatif en ajoutant un signe plus (+), la quantité et l'un des éléments suivants :

- minutes
- hours
- days
- weeks
- months
- years

2

Expressions de temps	Description
YYMMDDhhmm[.ss]	Spécifier une année, un mois, un jour, une heure, une minute et éventuellement une seconde
now	Indique le jour et l'heure actuels et l'exécution immédiate
midnight	Indique 00:00 (minuit)
noon	Indiquant 12:00 PM (midi)
teatime	Interprété comme 16 heures
AM	Indique l'heure avant 12h00
PM	Indique l'heure après 12h00
Today	La journée en cours
Tomorrow	Le jour suivant le jour en cours

# LES TACHES PLANIFIÉES

## LA COMMANDE AT

- Pour visualiser les taches programmées, il existe deux façons d'afficher les travaux en attente :
  - at avec l'option -l : `$at -l`
  - l'utilitaire atq: `$atq`
- **atq** répertorie tous les travaux actuellement programmés pour l'utilisateur connecté. Pour voir tous les travaux sur le système, vous devrez utiliser des privilèges élevés.
- Pour obtenir la liste des travaux en attente de tous les utilisateurs, exécutez la commande avec **sudo**.
- La syntaxe suivante permet d'afficher le contenu d'un travail at programmé :  
`$at -c [numéro_tâche]`
- Pour supprimer des tâches planifiées, il existe deux façons:
  - `$atrm [numéro_tâche]`
  - `$at -r [numéro_tâche]`

# LES TACHES PLANIFIÉES

## LA COMMANDE AT

1. Planifiez l'exécution d'une commande toutes les 2 minutes pendant les 10 prochaines minutes. La commande peut être l'affichage d'un simple message.
2. Planifiez l'exécution d'un script (prenez un script parmi les exemples précédents de notre cours) pour 10 minutes plus tard.
3. Planifiez les scripts (que nous avons vu dans ce cours) avec la commande at.