Linux Fondamentaux

David Zarebski

IMIE

2026

[0.0] ⋙ Programme

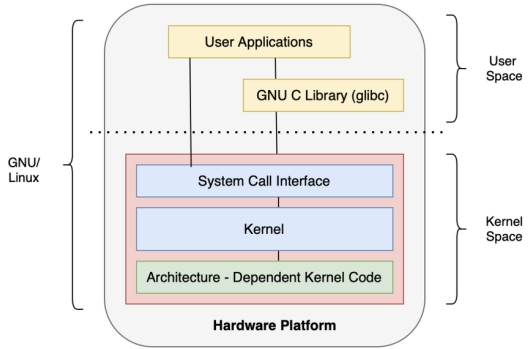
- Introduction
  - Architecture système
  - Histoire
- Installation d'une Debian 13
- Commandes de base d'un Linux / UNIX
- Notions centrales d'un système UNIX
  - Processus
  - Utilisateurs et groupes
  - Système de fichiers et droits

Introduction

Au sens strict du terme, Linux n'est pas un système d'exploitation mais un **kernel** (noyau)

Un kernel, entre autres choses

- gère le hardware
- expose ses ressources matérielles aux programmes via des API système
- expose les IO sous forme de systèmes de fichiers ou interfaces réseau (NIC)



Introduction
≫Architecture système

Architecture CPU

Dans leur conception même, les CPUs distinguent:

mode restreint

Peut exécuter un sous-ensemble d'instructions:

- conditions et boucles (`JMP` `CMP` ..)
- déplacer les données d'un registre à l'autre `MOV`
- copier les données d'un registre
- opérations logiques / mathématiques (`XOR` , `NOT` , `ADD`)

Plus de détails dans cette vidéo

mode privilégié

Peut exécuter n'importe quelle instruction dont:

- communication avec les IO
- gestion de la pagination mémoire

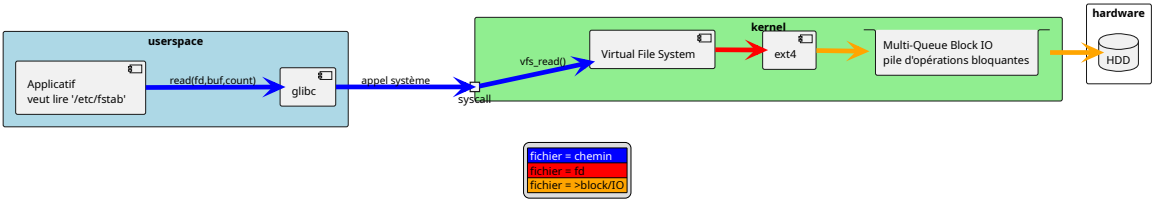
Un processus en kernel mode peut lire la mémoire de n'importe quel processus

Besoin d'une façade unique

Sans OS, chaque applicatif devrait accéder aux ressources matérielles par lui-même, ce qui impliquerait:

- d'embarquer l'ensemble des **pilotes** pour communiquer avec l'ensemble des composants possibles
- que deux applicatifs pourraient **éditer en même temps les mêmes ressources**, écrire sur un même espace mémoire (*race condition*)

[1.1] ⋙ Un exemple: la lecture d'un fichier



Selon le niveau, "*fichier*" ne désigne pas la même chose.

- diverses abstractions:
  - `VFS` : indépendance du système de fichiers (ext4, btrfs)
  - un 'fichier (chemin)' est en réalité constitué de blocks non-contigus sur le disque
- gestion de la concurrence
  - lecture des données en cache (non matérialisé sur le diagramme)
  - pile d'appels bloquant (lecture séquentielle des blocks de plusieurs `vfs_read()` simultanés)
  - optimisations (e.g. ajustement de la priorité de lecture des blocks en fonction de la position du bras sur le disque dur)

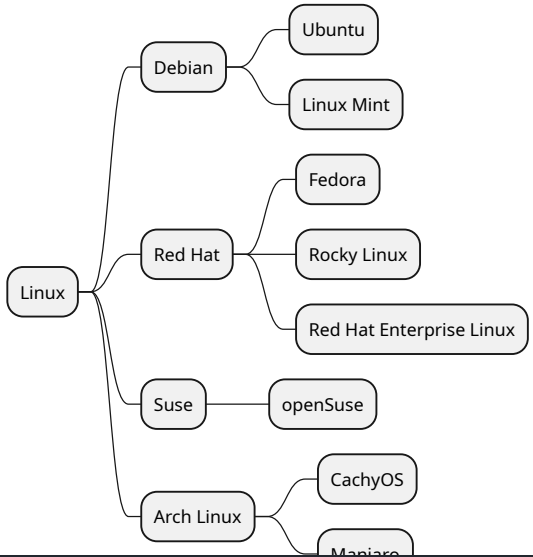
Introduction
≫Histoire

Maintenant que nous avons compris ce qu'est un noyau, revenons a l'histoire. Linux, en 1991, est avant tout un remplacement du noyau **Hurd** dans la distribution GNU (*GNU's Not UNIX*)

- Bourne-Again shell (a.k.a. **bash** Brian Fox)
- **GNU** tools (Richard Stallman)
- noyau: **Linux** (Linus Torvald)

Au sens étendu, cette fois, tout système d'exploitation construit autour d'un noyau Linux peut être qualifié de '*Linux*'. Cela comprends:

- distributions open source (`Fedora`, `Arch Linux`, `Debian`)
- distributions propriétaires (`RHLE`, `Azure Linux`)
- smartphones Android (noyau Linux + JVM)
- systèmes embarqués



Certaines distributions se basent sur d'autres. Cela implique qu'elles en partagent:

- le gestionnaire de package (e.g. `apt`, `dnf`, `zypper`, `pacman`)
- les intégrations principales (e.g. `SELinux` pour les basées Red Hat)
- les dépôts logiciels

> Parcequ'il suffit d'ajouter un composant quelconque pour créer une distributions, l'éco-système est saturé. **Recommandation**: ne vous perdez pas dans le distro-hopping et choisissez parmis les principales

Installation d'une Debian 13

Spécificités

- **une *vraie disto***: minimale, ne faisant pas de choix pour l'utilisateur et lui permettant d'expérimenter librement (comme **Arch**)
- architectures matérielles supportées: s390x, ppc64el, riscv64, ppc64, i386, arm64, amd64...
  ⇒ vous pourriez l'installer sur votre cafetière.
- philosphie libre exclusive
- pas d'intégrations *magiques* ⇒ besoin de comprendre/apprendre

Installation

Nous allons installer une VM à partir d'une ISO plutôt que de partir d'une image cloud toute prête afin:

- de rester au plus proche d'une installation *bare-metal*
- nous familiarisez avec les notions de partitionnement, bootloading, etc

Téléchargez l'ISO depuis le site de Debian puis ouvrez votre hyperviseur préféré

Commandes de base d'un Linux / UNIX

[3.0] ⋙ Guide de survie en environnement shell pour humains

le **shell**: l'interface humain / système la plus proche de ce dernier

Sur la majorité des systèmes, il s'agira de `bash` mais notez que des shells alternatifs existent (`ash`, `zsh`, ...).

Pour connaitre le **shell actuellement utilisé** par votre émulateur de terminal (**pty**), il suffit de chercher `SHELL` dans `printenv`

> l'auriculaire gauche est le doigt le plus important pour un humain dans un **pty**: la completion se fait avec la touche **TAB**

Qui suis-je?

Nous y reviendrons mais **tout processus appartient à un utilisateur**. Votre session de **pty** ne fait pas exception. Pour y répondre:

- `whoami`
- `id`
- `printenv | grep USER`

> Si le nom d'un utilisateur est une propriété accidentelle de ce dernier, son identifiant numérique (**UID**) est la propriété essentielle par laquelle il est identifié

Où suis-je?

Tout processus a une notion de **Dossier de travail actuel** (*current working directory* ou **CWD**). Ici aussi, votre session de **pty** ne fait pas exception. Pour y répondre:

- `pwd`
- `printenv | grep PWD`

[3.0] ⋙ Rangement de la racine *à la UNIX*

- **bin** Commandes essentielles (ls, cp, mv...)
- **boot** Fichiers de démarrage et noyau
- **dev** Fichiers de périphériques
- **etc** Fichiers de configuration système
- **home** Répertoires personnels des utilisateurs
- **lib** Bibliothèques partagées essentielles
- **media** Points de montage des supports amovibles
- **mnt** Points de montage temporaires
- **opt** Logiciels optionnels tiers
- **proc** Informations processus
- **root** Répertoire personnel de l'administrateur
- **run** Données d'exécution temporaires
- **sbin** Commandes système (admin)
- **srv** Données de services (web, ftp...)
- **sys** Informations noyau
- **tmp** Fichiers temporaires
- **var** Programmes et données utilisateurs

Se deplacer

Pour changer de répertoire courant, on utilise `cd` (*change directory*). Cette commande prend comme argument un chemin (relatif ou absolu). On peut ainsi:

- **avancer** dans l'arborescence de fichiers: `cd Documents`
- **remonter** en amont de l'arborescence: `cd ../`
- visiter des endroits exotiques: `cd /proc`

> pour un utilisateur disposant d'un `HOME`, `cd` sans argument ramènera l'utilisateur dans ce dernier

Lister le contenu

Pour lister le contenu du dossier courant, on se sert simplement de `ls` (sans argument)

[3.0] ⋙ Comment lire un man?

L'ensemble des commandes UNIX sont auto-documentées. Jettons un oeil à `man ls`

```
LS(1)                                                                  Use
NAME
     ls - list directory contents
SYNOPSIS
     ls [OPTION]... [FILE]...
DESCRIPTION
     List information about the FILEs (the current directory by default).  So
     Mandatory arguments to long options are mandatory for short options too.
     -a, --all
            do not ignore entries starting with .
     -A, --almost-all
            do not list implied . and ..
     --author
            with -l, print the author of each file
```

Notions centrales d'un système UNIX

Notions centrales d'un système UNIX
≫Processus

Théorie: comment un exécutable devient un processus

Les **exécutables** sont des entités autonomes et réplicables (i.e. je peux télécharger un programme). Les processus ne sont rien de plus qu'une étiquette par laquelle le noyau assigne a cet exécutable

- un espace mémoire dédié
- des entrées et sorties (*file descriptors*)
- du temps CPU
- un identifiant

pour en **suivre l'exécution**

> Vous avez beaucoup plus de **processus qui tournent** que de **coeurs disponibles** sur votre machine. Le kernel est donc responsable de *switcher* en permanence en fonction du **temps CPU** qui lui est alloué

Attributs d'un processus

```
ps aux
USER PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
root   1  0.0  0.0  21008 14968 ?   Ss   07:43 0:03 /sbin/init
...
```

- **USER**: l'utilisateur auquel le processus appartient
- **PID**: identifiant numérique
- **CPU** et **MEM**: le temps CPU et la RAM allouée au processus
- **VSZ**: mémoire allouée
- **RSS**: mémoire utilisée
- **TTY**: le processus dispose-t-il d'un terminal physique?
- **STAT**: état du processus
- **COMMAND**: Binaire à partir duquel le processus a été instancié

[4.1] ⋙ Une structure arborescente I

Tous les processus dépendent d'un précurseur, comme nous pouvons le mettre en évidence par cet exemple

```
python -m http.server &
# utilisez le PID qui apparait
pstree -H <PID>
kill <PID>
```

En remontant la chaîne, nous trouvons:

- le shell ayant lancé `python` (`zsh`)
- l'émulateur de terminal faisant invoquant `zsh` (`konsole`)
- le service par lequel ce **pty** a été lancé

> d'un système à l'autre, le nom des composants peut changer mais l'idée reste la même

Racine de cette arborescence (init)

OK, mais on ne peut remonter indéfiniement. On arrive à la racine, un processus un peu spécial que l'on appelle l'**init** et qui est responsable de démarrer l'ensemble des services. Il s'agit du PID 1 que nous avons rencontré précédemment.

```
ls -la /sbin/init
```

Sur la plupart des systèmes, `systemd` assure ce rôle, mais vous pouvez également rencontrer

- `openrc` (Alpine, Artix)
- `runit` (Void)

Qu'y-a-t'il au delà? le **kernel** (mais c'est une histoire pour un autre jour)

entrée, sorties

A son instantiation, chaque process se voit associé des canaux pour communiquer avec le reste du système:

- 0 **STDIN**: *entrée standard* par laquelle il peut lire ses arguments ou les données qu'il a à traiter
- 1 **STDOUT**: *sortie standard* par laquelle il peut transmettre un résultat, un code de sortie, etc
- 2 **STDERR**: *erreur standard* par laquelle il peut transmetre ses messages d'erreurs

Du point de vue des processus, il s'agit de **flux** (*stream*) de données binaires. Du point de vue du système, ces canaux sont matérialisés par des fichiers (comme tout le reste)

Illustration avec un exemple

Lancez `cat` dans un pty, identifiez son **PID**. Nous allons ensuite

```
echo "Hello cat" > /proc/<PID>/fd/0
```

Vous remarqerez que `cat` vient d'afficher votre message, ce qui est tout à fait normal: ce programme affiche dans sa sortie (ici, votre pty) les données qu'il reçoit en entrée (de quelque type que ce soit).

```
cat /bin/sh > /proc/<PID>/fd/0
```

> même logique, sortie cracra: identifiez en la raison

Analyse des file descriptors

Toujours avec notre `cat` en écoute, interragissons maintenant avec sa sortie standard (`./fd/1`). Identifions cette dernière

```
ls  -la /proc/<PID>/fd
lrwx------ 1 zar3bski zar3bski 64 16 juil. 14:32 0 -> /dev/pts/2
lrwx------ 1 zar3bski zar3bski 64 16 juil. 14:41 1 -> /dev/pts/2
lrwx------ 1 zar3bski zar3bski 64 16 juil. 14:43 2 -> /dev/pts/2
lrwx------ 1 zar3bski zar3bski 64 16 juil. 14:43 3 -> 'socket:[29436]'
lr-x------ 1 zar3bski zar3bski 64 16 juil. 14:43 4 -> 'pipe:[69997]'
l-wx------ 1 zar3bski zar3bski 64 16 juil. 14:43 5 -> 'pipe:[69997]'
```

et nous remarquons que ces trois **liens symboliques** pointent vers `/dev/pts/2` (a.k.a. le pty par lequel nous avons appelé ce programme). En soi, tout ceci est plutôt logique

> Digresser sur les 3 (**D-Bus**),4,5 (**piping Bash**) nous amènerait un peu trop loin.

[4.1] ⋙ Exercice 1: tty

Trouvez l'ensemble des processus associés à un TTY (tout pty est lié à un TTY)

Quel est le dénominateurs commun des processus

- associés à un TTY
- associés à un PTY

[4.1] ⋙ Exercice 1: tty correction

Deux manières

Filtres natifs de ps

```
ps a
```

Un grep sale des familles

```
ps -A | grep -E "(pts|tty)"
```

Les processus associés à un TTY sont des serveurs d'affichages, (`Xorg` , `wayland`)

[4.1] ⋙ Exercice 2: RAM

Identifiez le processus consommant le plus de mémoire vive. Nous cherchons la consommation effective (**RSS**) et non la mémoire allouée.

[4.1] ⋙ Exercice 2: RAM correction

```
ps aux --sort -rss
```

Notions centrales d'un système UNIX
≫Utilisateurs et groupes

[4.2] ⋙ Utilisateurs I

/etc/passwd

Nous allons jeter un oeil à `/etc/passwd`

```
root:x:0:0::/root:/usr/bin/bash
...
nobody:x:65534:65534:Kernel Overflow User:/:/usr/bin/nologin
dbus:x:81:81:System Message Bus:/:/usr/bin/nologin
```

Chaque ligne correspond à un utilisateur. Tout utilisateur dispose des attributs suivants (matérialisé par chaque champ)

- **nom** d'utilisateur: celui-ci peut changer et suit des conventions.
- dispose d'une entrée dans `/etc/shadow` (compte vérifié)
- **UID**: l'identifiant numérique de cet utilisateur qui détermine ses droits sur le système. ('*root*' est `root` parce qu'il est l'utilisateur 0)
- **GID**: l'identifiant de son **groupe** associé par défaut
- description (optionnelle)
- son `HOME` (de dossier dans lequel il dispose, récursivement, de tous les droits)
- son SHELL par défaut

Regardons maintenant `/etc/group`

La relation **UID GID** est une relation *N pour N*:

- ⋙ un même utilisateur peut appartenir à plusieurs groupes
- ⋙ un même groupe peut contenir plusieurs utilisateurs

[4.2] ⋙ Création d'un utilisateur et de ses groupes associés

A l'aide des commandes `useradd` et `groupadd`, créez les groupes et l'utilisateur suivant

username	UID	groupname	GID
george	1111	sales	1200
		managers	1300

de telle manière à ce que:

- george ∈ sales
- george ∈ managers
- george n'ait pas de `HOME` (i.e. `/home/george`)

Verifiez le résultat à l'aide de `id george`

[4.2] ⋙ Création d'un utilisateur et de ses groupes associés (correction)

```
sudo groupadd -g 1200 sales
sudo groupadd -g 1300 managers
sudo useradd -N -M -G 1200,1300 -u 1111
```

Supprimez cet utilisateur et ses groupes

username	UID	groupname	GID
george	1111	sales	1200
		managers	1300

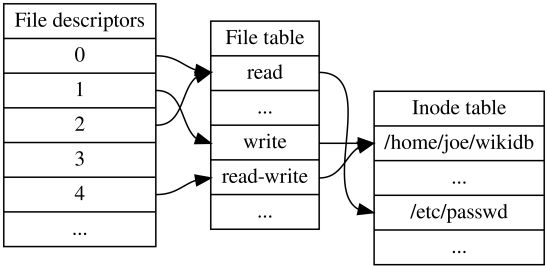
[4.2] ⋙ Suppression d'un utilisateur et de ses groupes associés: correction

```
sudo userdel george
sudo groupdel sales
sudo groupdel managers
```

Notions centrales d'un système UNIX
≫Système de fichiers et droits

En UNIX, il existe 3 types d'opérations possibles sur un fichier (représenté ici avec 3 bits au lieu de 4)

opération	représentation binaire
lecture	100
écriture	010
exécution	001



Si le **contenu** d'un fichier est l'ensemble de ses 01 stockés dans un système de fichier, les droits de ce dernier sont des **métadonnées** stockées dans une table d'**inodes**.

```
ls -i /bin/sh
101194744 /bin/sh # numéro inode du fichier
```

Notation octale des droits

Pour représenter ces droits de manière *user friendly*, on passe en **notation octale** (base 8)

opération	représentation binaire	notation octale
lecture	100	4
écriture	010	2
exécution	001	1

Propriété additive des droits

- $7 = 111 = 100 + 010 + 001 =$ lecture + écriture + exécution
- $6 = 110 = 100 + 010 =$ lecture + écriture
- $5 = 101 = 100 + 001 =$ lecture + exécution ....

vous avez l'idée générale

[4.3] ⋙ Droits UNIX appliqués aux dossiers

Les droits sur les dossiers sont représenté de la même manière. Cependant, le sens des opérations change

opération	fichier	dossier
lecture	lire le contenu	lire les entrées du dossier
écriture	modifier le contenu	ajouter un fichier au dossier
exécution	instancier un processus	lire les métadonnées des fichiers qu'il contient

lecture et **exécution** sont nécessaires pour effectivement lister le contenu d'un dossier. La **lecture** seule liste les entrées sans qu'aucun autre attribut ne soit accessible (propriétaire, taille, date de modification...), l'**exécution** seule permet d'accéder à ces métadonnées, mais il faudrait deviner le nom des fichiers.

[4.3] ⋙ Lien avec la notion de propriétaire d'un fichier

Tout fichier appartient:

- à un utilisateur (**UID**)
- à un groupe (**GID**)

Une description complète des droits d'un fichier nécessite donc de lister les opérations possibles pour

	son propriétaire (**UID**)	le groupe (**GID**)	tout autre utilisateur
777	111	111	111
755	111	101	101
600	110	000	000

> Le groupe auquel un fichier appartient est indépendant des groupes auquel le propriétaire du fichier appartient

```
ls -la /usr/bin/

-rwxr-xr-x 1 root root   47K 11 mars   2025 zvbid
-rwxr-xr-x 1 root root  123K 11 mars   2025 zvbi-ntsc-cc
-rwxr-xr-x 1 root root  161K  7 juil. 20:07 ZXingQtCamReader
-rwxr-xr-x 1 root root  143K  7 juil. 20:07 ZXingReader
-rwxr-xr-x 1 root root   59K  7 juil. 20:07 ZXingWriter
-rwxr-xr-x 1 root root   15K  1 juin   2025 zzcat
-rwxr-xr-x 1 root root   15K  1 juin   2025 zzdir
-rwxr-xr-x 1 root root   15K  1 juin   2025 zzxorcat
-rwxr-xr-x 1 root root   15K  1 juin   2025 zzxorcopy
-rwxr-xr-x 1 root root   15K  1 juin   2025 zzxordir
```

[4.3] ⋙ SUID, GUID & sticky bit I

Outre les niveaux rencontrés (user, groupe, autre), les droits d'un fichier disposent d'un byte additionnel destiné à contenir les **droits spéciaux**

Représentation

	appellation	notation '*humaine*'	notation octale
100	SUID	—s——	4xxx
010	GUID	——s—	2xxx
001	sticky bit	———t	1xxx

Alors même que ces droits correspondent aux 4 premiers bits sur 16 (restitué fidèlement en **notation octale**), la *notation humaine* substitue `s` (resp. `t`) à `x` sur les positions correspondantes

Rôles et implications

- **SUID**: binaire exécuté avec les privilèges de son **propriétaire**
- **GUID**: binaire exécuté avec les privilèges de son **groupe propriétaire**
- **sticky bit**: appliqué à un dossier
  - empêche la suppression des fichiers qu'il contient
  - seul son propriétaire et root peuvent alors les supprimer

[4.3] ⋙ SUID, GUID & sticky bit III

Exemples

- **SUID**: `find / -perm -u=s 2>/dev/null`

```
ls -la /usr/bin/passwd
-rwsr-xr-x 1 root root 80792 30 juil. 12:04 /usr/bin/passwd
```

- **GUID**: `find / -perm -g=s 2>/dev/null`

```
ls -la /usr/bin/write
-rwxr-sr-x 1 root tty 22688  2 sept. 11:36 /usr/bin/write
```

- Sticky bit: `/tmp` contient pas mal d'exemples

```
ls -la /tmp
...
drwxrwxrwt  2 root     root          40 13 sept. 09:21 .font-unix
```

Si tout utilisateur peut écrire dans **.font-unix**, seul root peut supprimer des fichiers qu'il contient

Un type spécifique de fichiers

Comparons

```
ls /lib
# avec
ls -la /lib
lrwxrwxrwx 1 root root 7 12 oct.   2025 /lib -> usr/lib
```

De part `l---------`, nous savons qu'il s'agit d'un type de fichier spécial: un **lien symbolique**

Fonctionnement

Un lien symbolique est un type de fichier pointant sur un autre fichier du système de fichier (un genre de *raccourci*). Leurs principales fonctions:

- exposer certaines ressources sur des chemins attendus par certains composants (e.g. certains applicatifs attendent les librairies en `lib`)
- réunir dans un même dossier des fichiers des **block devices différents**
- exposer la **version courante** d'un applicatif sur un même chemin:

```
ls -la /var/lib/flatpak/app/com.bitwarden.desktop
total 12
drwxr-xr-x 3 root root 4096  2 janv.  2026 .
drwxr-xr-x 6 root root 4096 22 juin  19:03 ..
lrwxrwxrwx 1 root root   13  2 janv.  2026 current -> x86_64/stable
drwxr-xr-x 3 root root 4096  2 janv.  2026 x86_64
```

Création

`ln` (*link*) est l'outil privilégié pour créer des liens symboliques

```
ln -s [chemin_cible] [chemin_lien]
```

D'autres moyens existent: e.g. `cp -s`

liens symboliques

- ‘*soft links*’, ‘*sym links*’
- simples pointeurs vers le fichier d’origine
- `ln -s`

suppression du lien $\not\equiv$ suppression de la cible

$\Rightarrow$ `rm <lien>` suffit à supprimer le lien

liens physiques

- ‘*physical links*’, ‘*hard links*’
- lien et cible partagent le même **inode**
- `ln -P`

suppression du lien $\equiv$ suppression de la cible

$\Rightarrow$ `unlink` pour supprimer le lien sans supprimer la cible

[4.3] ⋙ Exercice 1

Dans `/tmp`, créer un dossier `exercice_1` appartenant à UID 1000. Hormis UID 1000 et GID 1000, personne ne doit pouvoir **lister le contenu** de ce dossier. Vous allez ensuite créer deux fichiers:

- `restricted` que seul UID 1000 peut lire ou écrire
- `public` que UID/GID 1000 peuvent lire et écrire mais que tout le reste du système peut lire

Mettre en évidence qu'un **utilisateur quelconque** peut bien lire `/tmp/exercice_1/public` alors même qu'il ne peut pas lister `/tmp/exercice_1/`

Commandes pertinentes:

- `mkdir`: creation d'un dossier
- `touch`: creation d'un fichier vide
- `chmod`: modifie les droits d'un fichier

```
cd /tmp
mkdir exercice_1
chmod 771 exercice_1
touch exercice_1/restricted
chmod 600 exercice_1/restricted
touch exercice_1/public
chmod 664 exercice_1/public

sudo -u nobody ls exercice_1
 #ls: impossible d'ouvrir le répertoire 'exercice_1': Permission non accordée
sudo -u nobody cat exercice_1/public
```

[4.3] ⋙ Exercice 2

Dans `/tmp`, créer un dossier `exercice_2` appartenant à **root** (UID/GID 0). Seul root et son group doit pouvoir `rwx` l'ensemble de cette arborescence à créer

```
/tmp/
    exercice_2/
        niveau_1/
            niveau_2/
                niveau_3/
                    private
```

> pensez à `umask` et aux options de `mkdir`

[4.3] ⋙ Exercice 2: correction

```
sudo -i
umask 007
mkdir -p exercice_2/niveau_1/niveau_2/niveau_3
touch exercice_2/niveau_1/niveau_2/niveau_3/private

ls -la exercice_2/niveau_1/niveau_2/niveau_3
#total 0
#drwxrwx--- 2 root root 60 10 juil. 19:27 .
#drwxrwx--- 3 root root 60 10 juil. 19:27 ..
#-rw-rw---- 1 root root  0 10 juil. 19:27 private
```

[4.3] ⋙ Exercice 3: Sym links

Dans `/tmp`, vous allez essayer de créer deux liens vers `/bin/bash`

- `soft` : lien symbolique
- `hard` : lien physique

1. Dans le cas de `soft`, ce dernier est-il exécutable?
2. dans le cas de `hard`, pouvez-vous trouver un contournement? Qu'en déduisez-vous?

Supprimez ces deux fichiers sans pourrir votre système

[4.3] ⋙ Exercice 3: Sym links correction

Création

```
cd /tmp
ln -s /bin/bash ./soft
ln -P /bin/bash ./hard
# oups
sudo ln -P /usr/bin/bash ./hard
```

Suppression

```
rm ./soft
sudo unlink ./hard
```