Systèmes d'exploitation

Fichiers et entrées-sorties

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

http://tropars.github.io/

2026

Références

Beaucoup des slides de ce cours ont été produites par d'autres personnes:

- E. Brunet et G. Thomas – Système de Fichiers
- V. Marangozova – Fichiers et entrées-sorties

Contenu de ce cours

- La notion de système de fichiers
- Manipuler des fichiers
  - A la ligne de commande
  - Les appels systèmes
  - Les redirections
- Les droits d'accès

Agenda

Les systèmes de fichiers

Manipuler des fichiers

Les droits d'accès

Système de Fichiers

- Besoin de mémoriser des informations
  - Photos, PDF, données brutes, exécutables d'applications, le système d'exploitation lui-même, etc.
- Organisation du stockage sur mémoire de masse
  - Localisation abstraite grâce à un chemin dans une arborescence
  - Unité de base = fichier
- Exemples de types de systèmes de fichiers
  - NTFS pour Windows, ext2, ext3, ext4 pour Linux, HFSX pour Mac-OS
  - FAT pour les clés USB, ISO pour les CD
  - … et des myriades d'autres types de systèmes de fichiers

Qu'est-ce qu'un fichier

- Un fichier est la réunion de
  - Un contenu, c'est-à-dire un ensemble ordonné d'octets
  - Un propriétaire
  - Des horloges scalaires (création, dernier accès, dernière modif)
  - Des droits d'accès (en lecture, en écriture, en exécution)

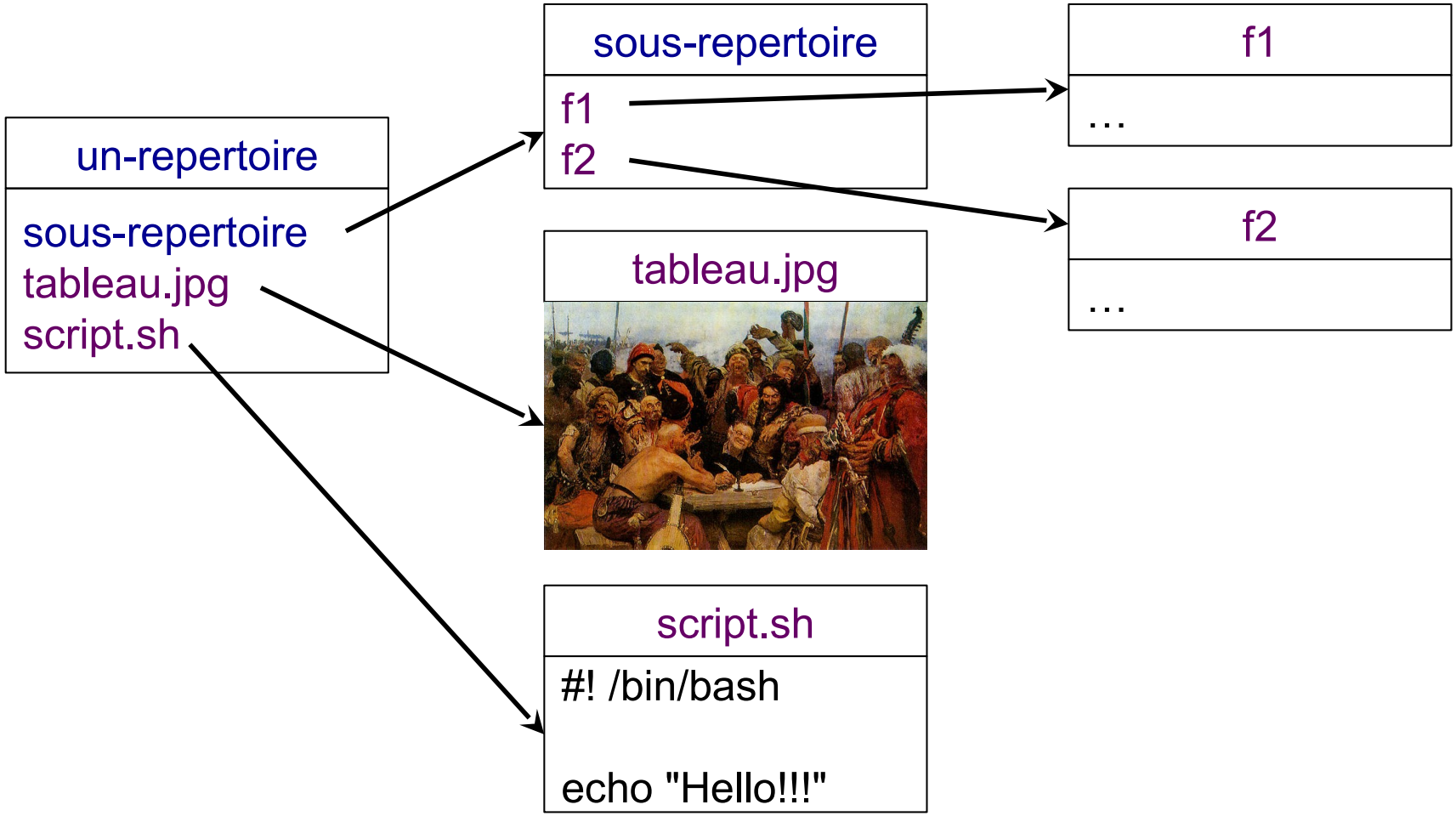
- Attention : c'est inattendu, mais un fichier est indépendant de son nom (c.-à-d., le nom ne fait pas parti du fichier et un fichier peut avoir plusieurs noms)

On stocke de nombreux fichiers

- Facilement plusieurs centaines de milliers de fichiers dans un ordinateur
  - Plusieurs milliers gérés/utilisés directement par l'utilisateur
  - Plusieurs centaines de milliers pour le système et les applications
- Problème : comment retrouver facilement un fichier parmi des centaines de milliers ?
- Solution : en rangeant les fichiers dans des répertoires (aussi appelés dossiers)

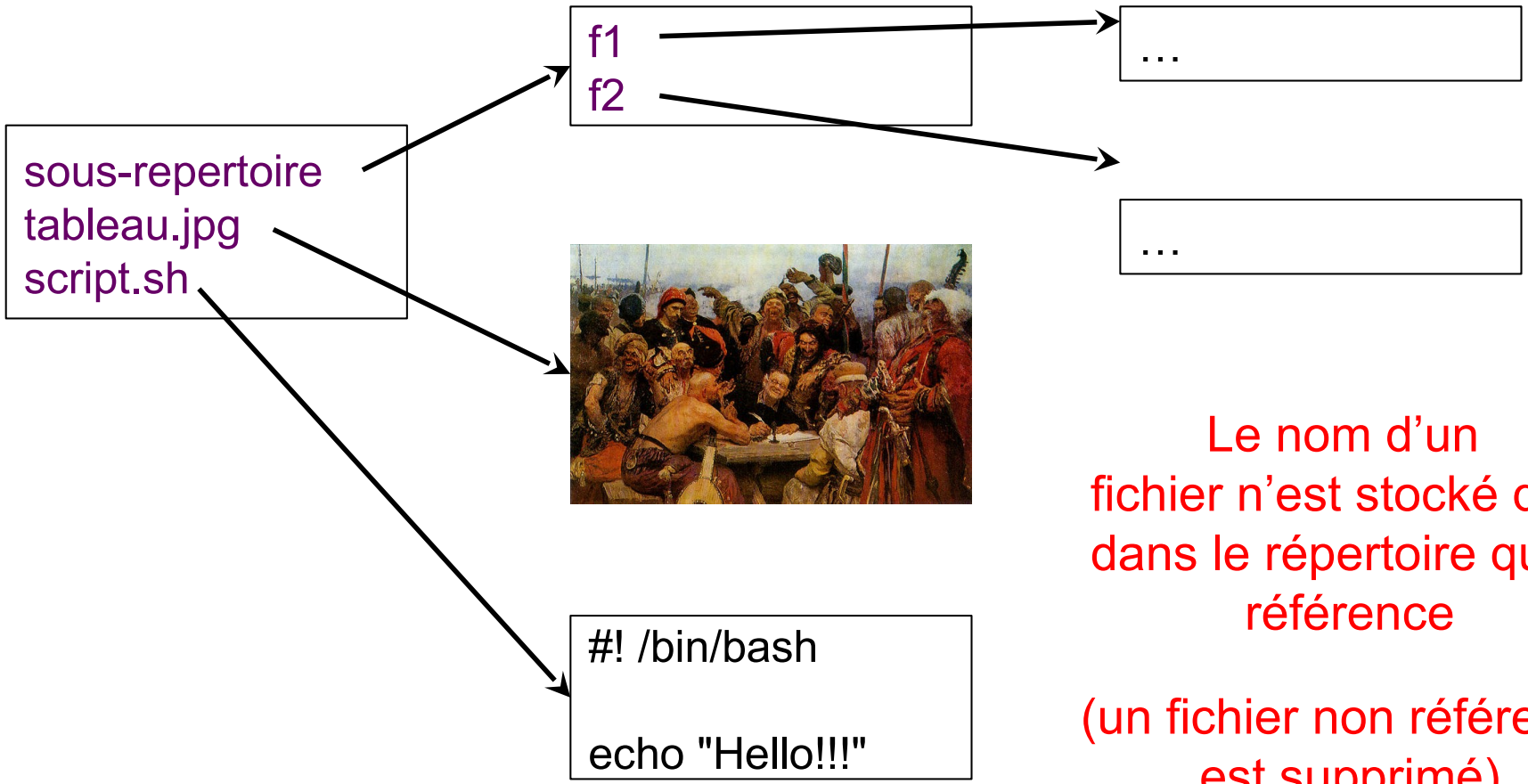
Organisation en répertoires

- Répertoire = fichier spécial qui associe des noms à des fichiers



Organisation en répertoires

- Répertoire = fichier spécial qui associe des noms à des fichiers

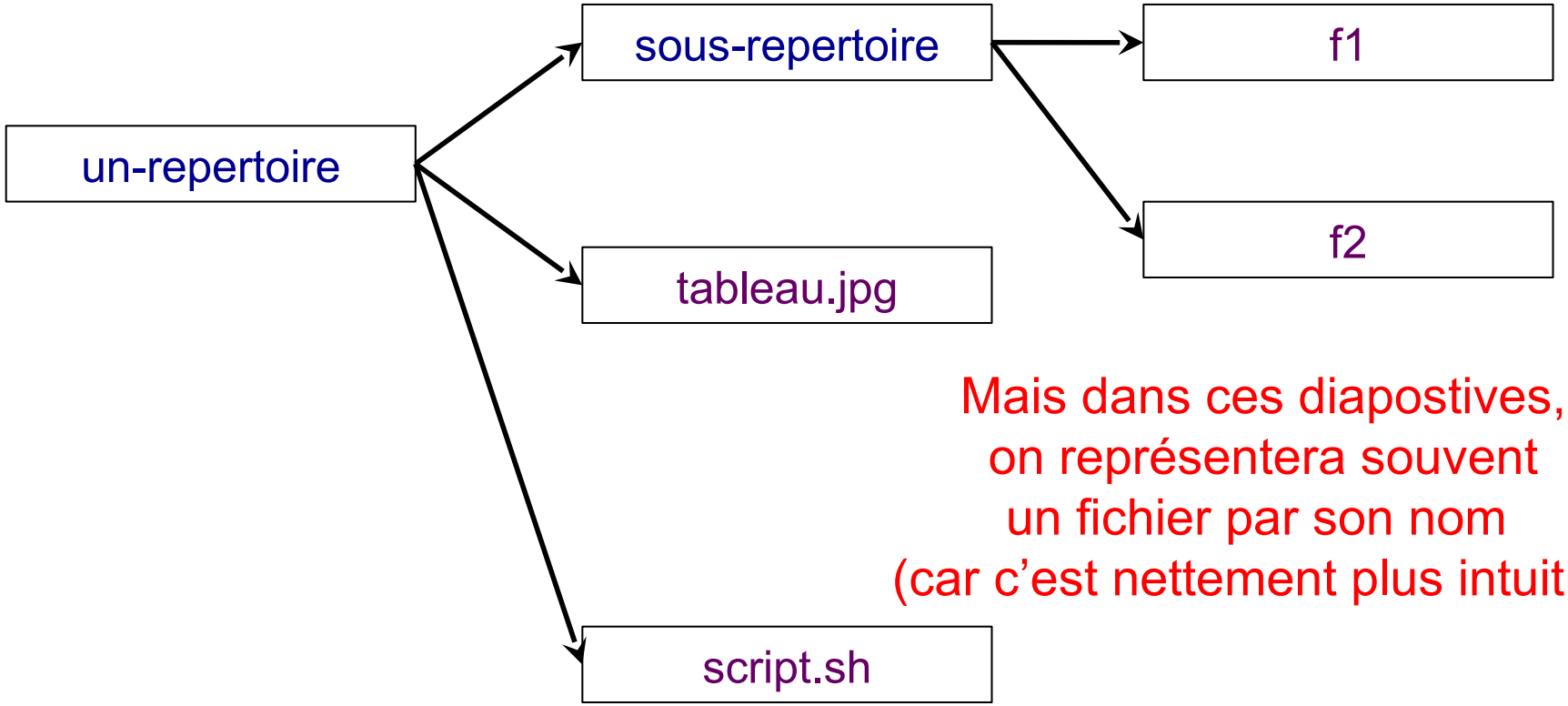


Le nom d'un fichier n'est stocké que dans le répertoire qui le référence

(un fichier non référencé est supprimé)

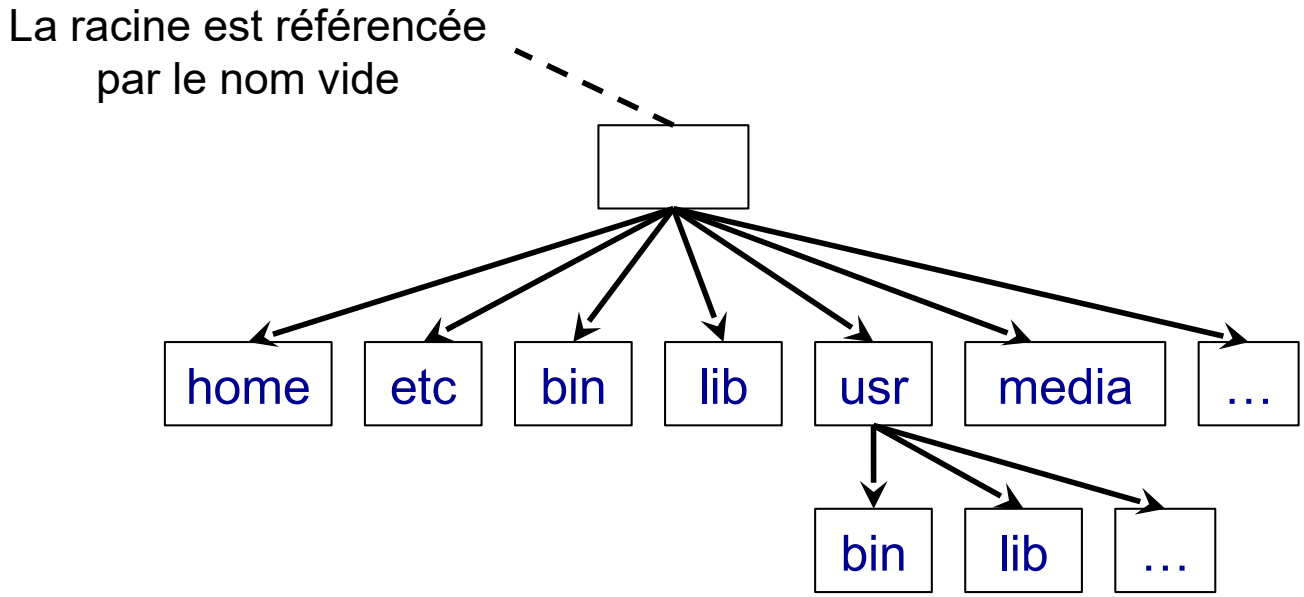
Organisation en répertoires

- Répertoire = fichier spécial qui associe des noms à des fichiers



Mais dans ces diapostives,
on représentera souvent
un fichier par son nom
(car c'est nettement plus intuitif !)

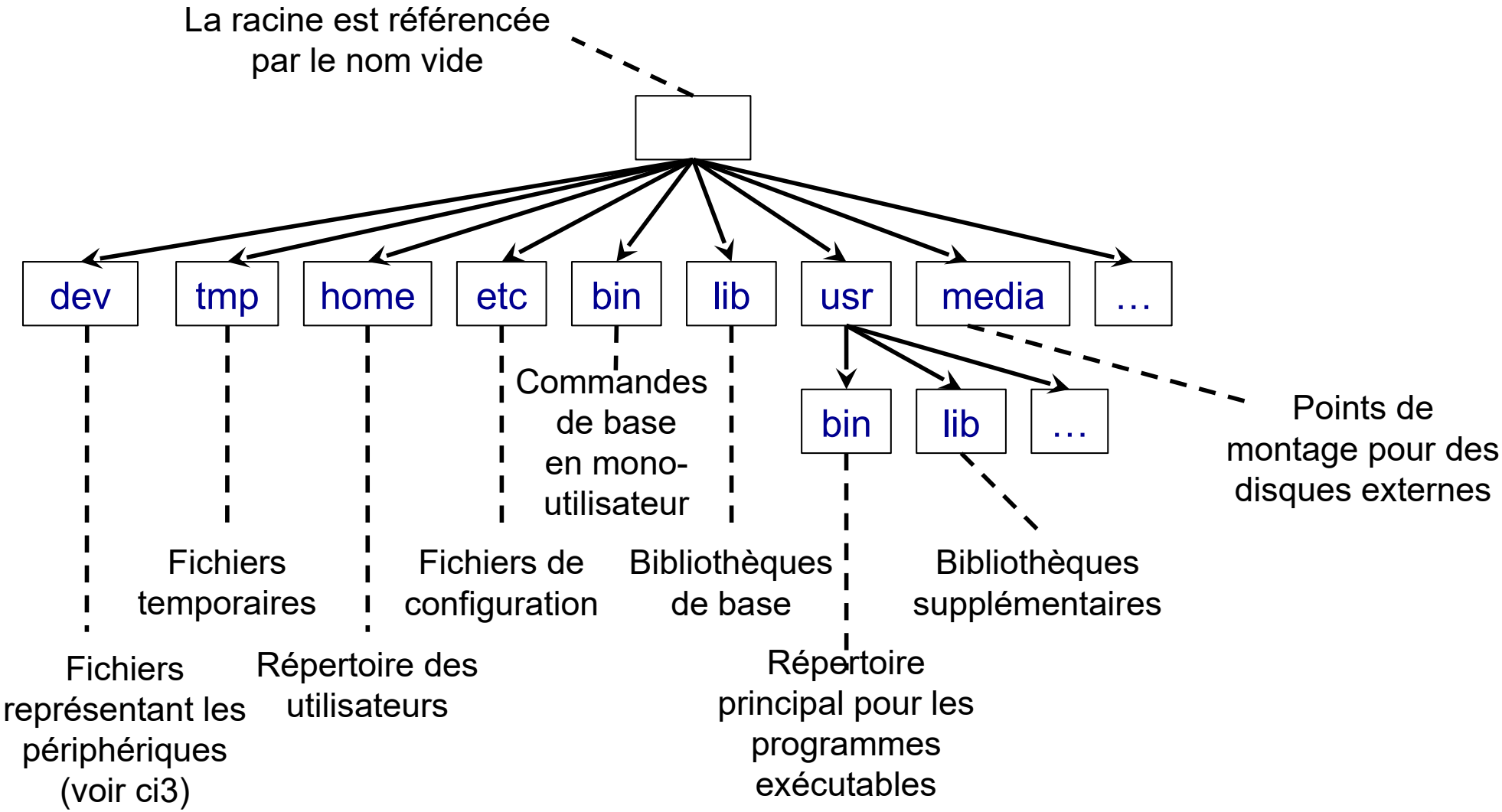
Arborescence standard des systèmes d'exploitation UNIX



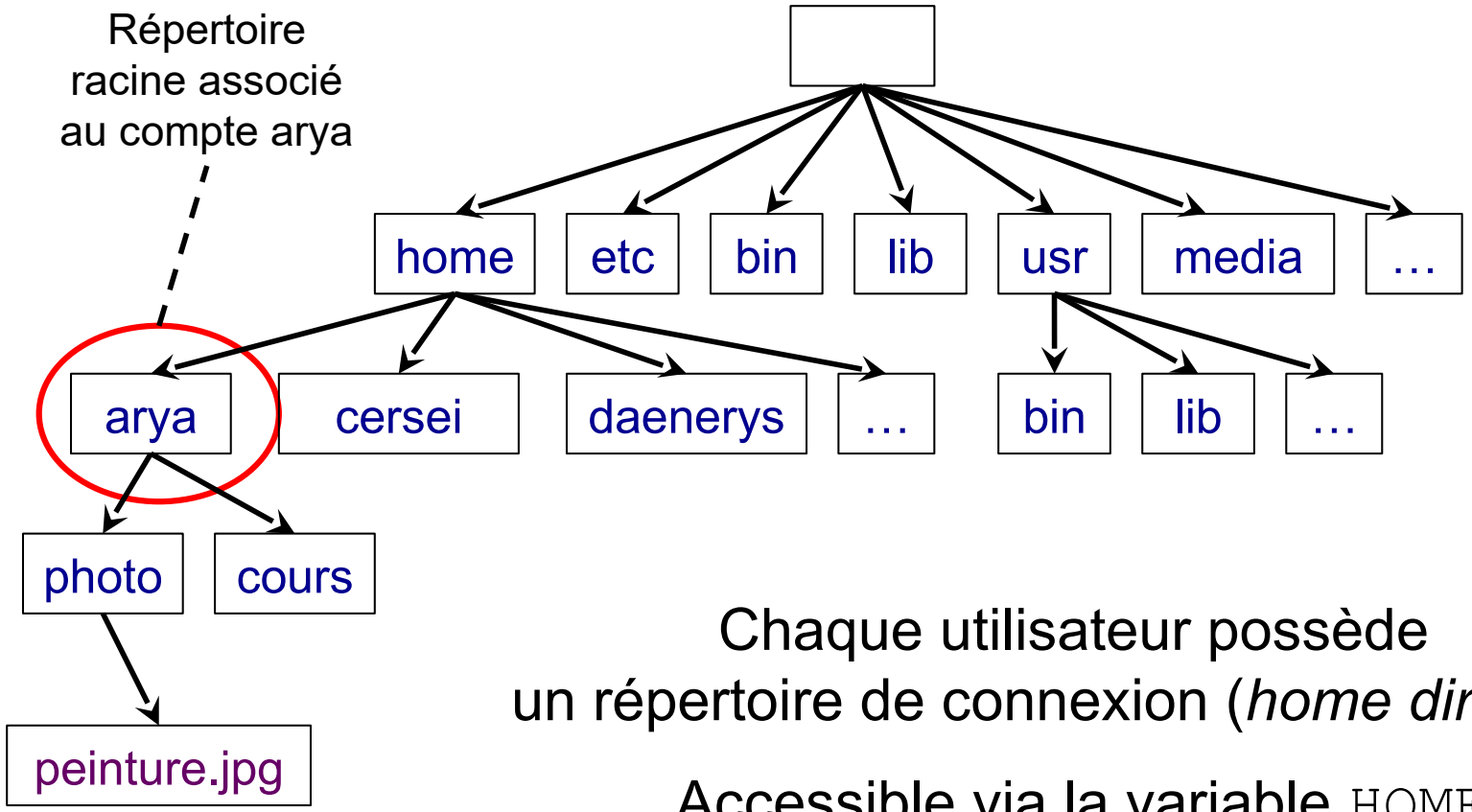
La plupart des systèmes d'exploitation Unix (GNU/Linux, BSD, MacOS…) utilisent une arborescence de base standardisée
(seul Windows utilise une arborescence réellement différente)

Vous pouvez la consulter en faisant : `man hier` (pour hierarchy)

Arborescence standard des systèmes d'exploitation UNIX



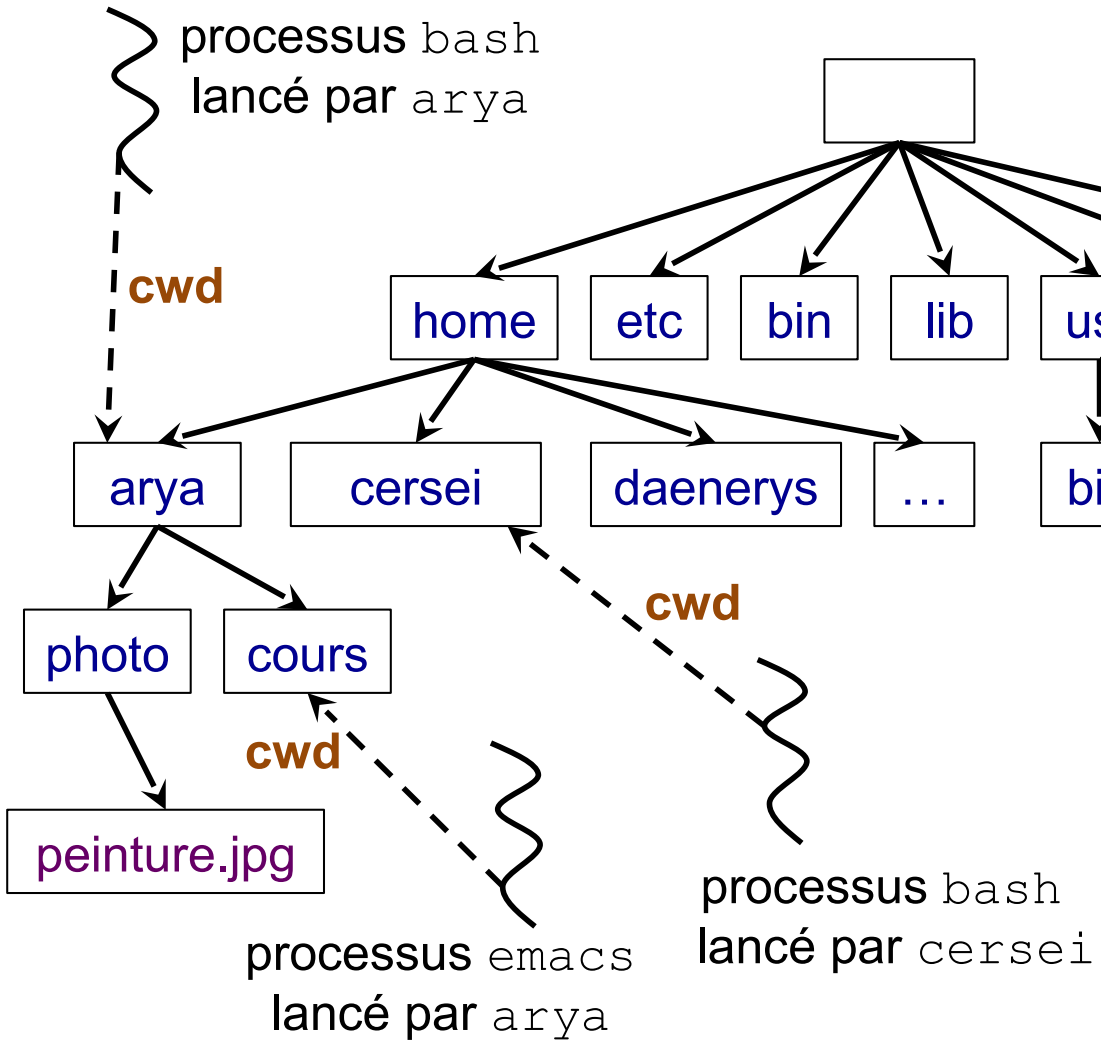
Arborescence standard des systèmes d'exploitation UNIX



Chaque utilisateur possède un répertoire de connexion (*home directory*)

Accessible via la variable `HOME`

Notion de répertoire de travail



Chaque processus possède un répertoire de travail (noté cwd pour *current working directory*)

Notion de chemin

- En `bash`, le séparateur de répertoires est le caractère `/`

- Un chemin s'écrit sous la forme `a/b/c` qui référence
  - le fichier `c`
  - se trouvant dans le répertoire `b`
  - se trouvant lui même dans le répertoire `a`

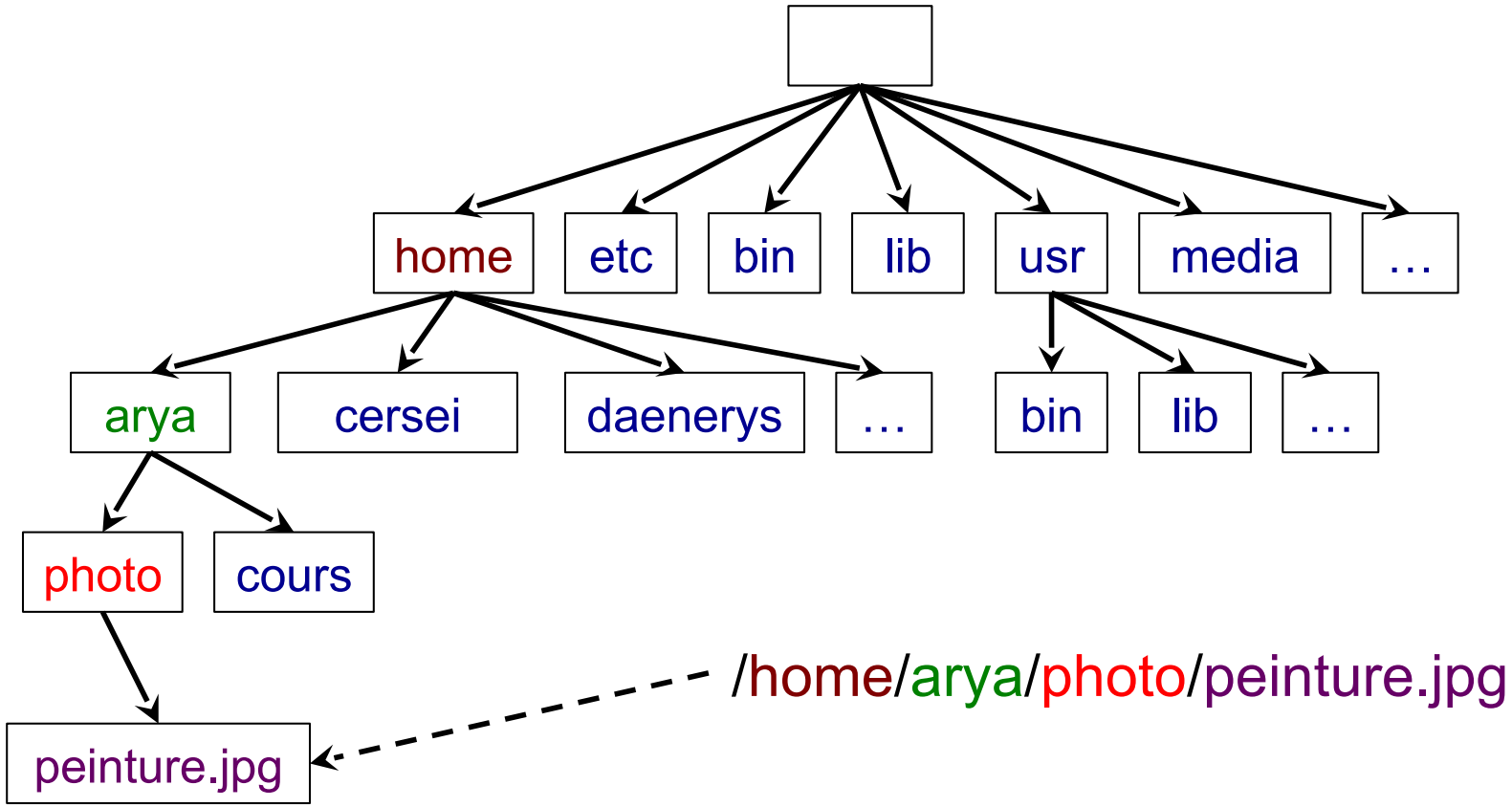
- Un chemin absolu part de la racine du système de fichiers

  Commence par le nom vide (racine), par exemple `/a/b/c`

- Un chemin relatif part du répertoire de travail du processus

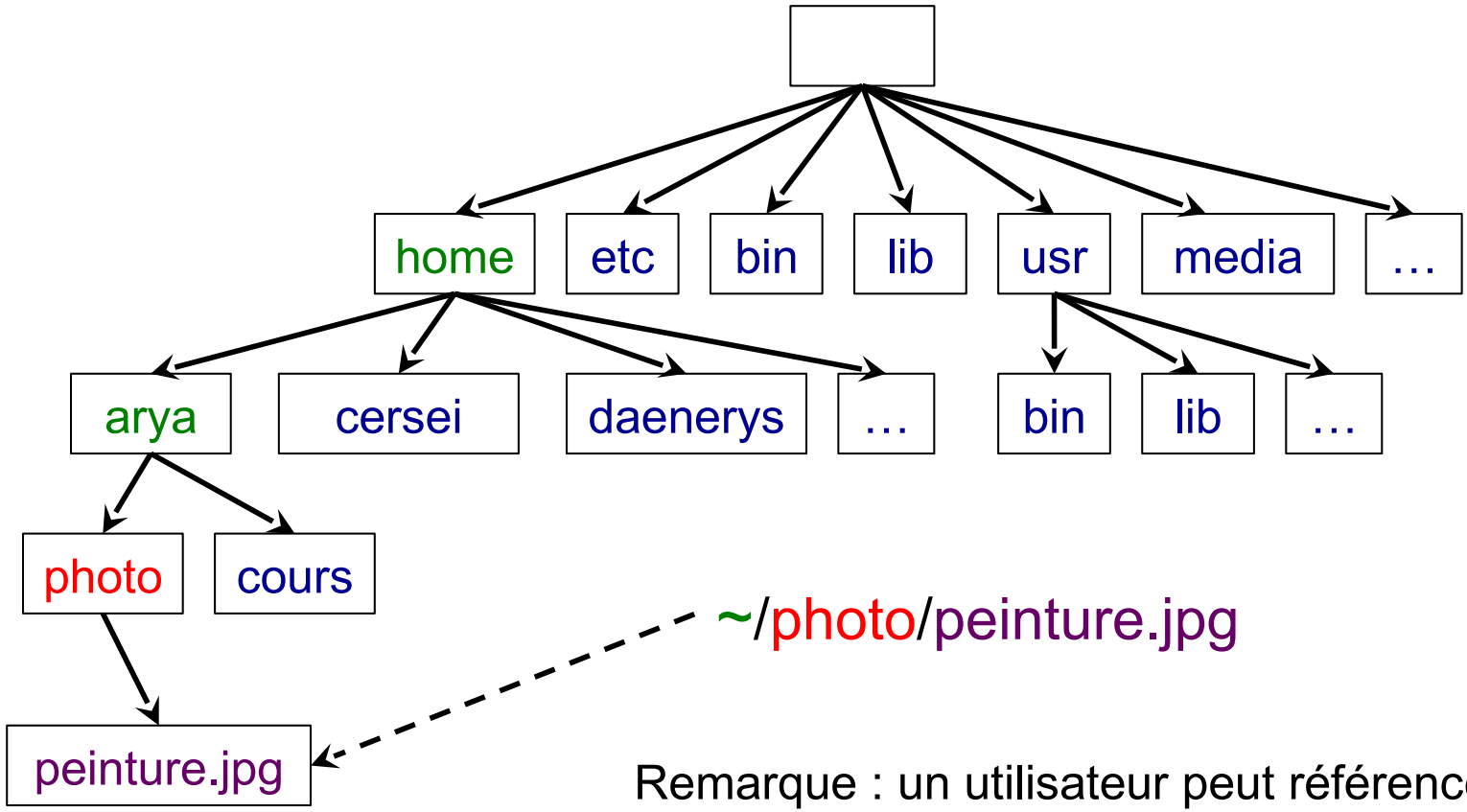
  Commence par un nom non vide, par exemple `a/b/c`

Exemple de chemin absolu (1/2)



Exemple de chemin absolu (2/2)

Un utilisateur peut utiliser ~ comme raccourci pour son répertoire de connexion



Remarque : un utilisateur peut référencer le répertoire de connexion d'un autre utilisateur avec `~name` (par exemple `~arya/photo/peinture.jpg`)

Désignation des fichiers (2)

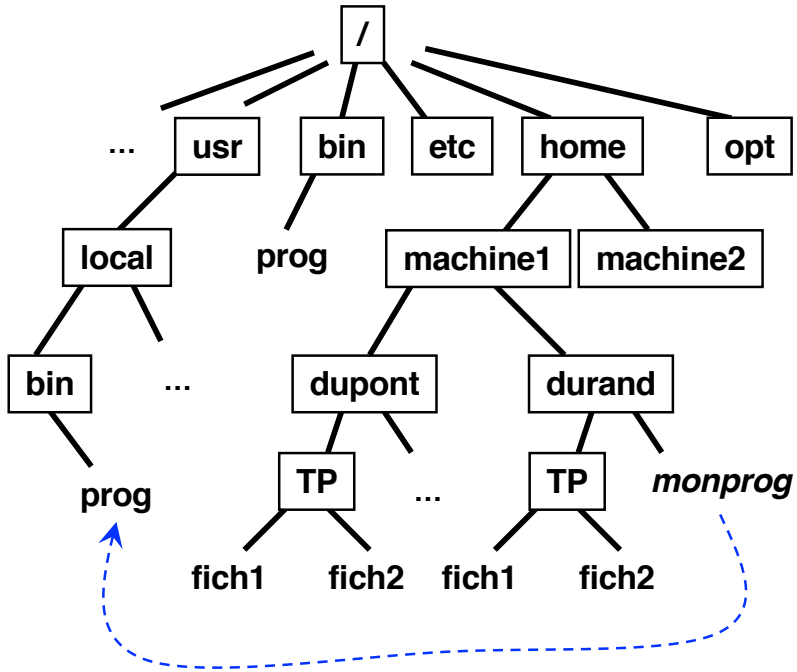
- **Divers raccourcis simplifient la désignation**
  - noms relatifs au répertoire courant

Si répertoire courant = `/home/machine1/dupont/`
alors on peut utiliser les noms relatifs
`TP/fich1, TP/fich2`

  - désignation du père

Si répertoire courant = `/home/machine1/dupont/`
alors on peut utiliser
`../durand/TP/fich1`

  - liens symboliques



Si répertoire courant = `/home/machine1/durand/`

- création du lien : `ln -s /usr/local/bin/prog monprog`
- dans le répertoire courant, le nom `monprog` désigne maintenant le fichier `/usr/local/bin/prog`
- un lien n'est qu'un raccourci : si le fichier cible est supprimé, le lien devient invalide

Désignation des fichiers (3)

- **Répertoire courant**
  - par défaut, tout usager a un répertoire courant de base
    - *home directory*
    - par exemple **`/home/machine/dupont`**
    - un raccourci est **`~dupont`**
  - on peut changer de répertoire courant au moyen de la commande
    `cd <nom du répertoire destination>`
    - cd sans paramètres ramène au répertoire de base
  - le nom **`.`** désigne le répertoire courant
  - **`pwd`** : connaître le nom absolu du répertoire courant
  - **`ls :`** connaître le contenu du répertoire courant par (`ls -l` est plus complet)

```
<unix> ls -l
total 16
lrwxrwxrwx    1 krakowia sardes    19 Jul  9  2004 isr-l3-td.cls -> ../../isr-l3-td.cls
-rw-r--r--    1 krakowia sardes    94 Jul  9  2004 Makefile
drwxr-xr-x    2 krakowia sardes    4096 Dec 20 16:20 Old/
-rw-r--r--    1 krakowia sardes    4320 Jan 18 10:17 td3.tex
```

Désignation des fichiers : règles de recherche

- **Pour exécuter un programme (fichier exécutable), il suffit d'entrer une commande avec son nom simple. Comment le système trouve-t-il le fichier correspondant ?**
- **Règle de recherche**
  - le système explore dans l'ordre une suite de répertoires
  - cette suite est enregistrée dans une variable d'environnement PATH

```
<unix> echo $PATH
/usr/local/bin:/bin:/usr/bin:/opt/gnu/arm/bin:/usr/j2se/bin
```

  - La commande `which` indique le nom absolu du fichier qui sera exécuté par défaut

```
<unix> which gcc
/usr/local/bin/gcc
```

Règle de recherche : exemple d'application

- **Vous avez créé votre propre exécutable, dans le répertoire courant**

```
<unix> ls –l
-rwxr-xr-x  1 vania  staff  55872  6 fév 07:32 alarm
```

- **Pour l'exécuter**
  - soit changer le PATH

```
[unix] alarm
-bash: alarm: command not found
[unix] export PATH=.:$PATH
[unix] alarm
bip
bip
bip
bip
^C
```

  - soit plus simplement `<unix> ./alarm`

Il existe de nombreux types de fichiers

- Fichier ordinaire
- Répertoire
- Lien symbolique
- Device : un fichier qui représente un périphérique
  (disque dur, carte son, carte réseau, …)
  - Par exemple `/dev/sda1`
- Tube nommé : fichier spécial étudié en CI6
- Socket : fichier spécial proche des tubes (non étudié dans ce cours)

Agenda

Les systèmes de fichiers

Manipuler des fichiers

Les droits d'accès

Utilisation des fichiers dans le langage de commande

- **Créer un fichier**
  - Le plus souvent, les fichiers sont créés par les applications, non directement dans le langage de commande. Exemple : éditeur de texte, compilateur, etc
  - On peut néanmoins créer explicitement un fichier
    - **`touch toto`**
- **Créer un répertoire**
  - **`mkdir <nom du répertoire>`** le répertoire est initialement vide
- **Détruire un fichier**
  - **`rm <nom du fichier>`**
  - **`rm -i`** va demander une confirmation
- **Détruire un répertoire**
  - **`rmdir <nom du répertoire>`** le répertoire doit être vide
- **Conventions pour les noms de fichiers**
  - ***** désigne n'importe quelle chaîne de caractères :
    - **`rm *.o`** : détruit tous les fichiers dont le nom finit par .o
    - **`ls *.c`** : donne la liste de tous les fichiers dont le nom finit par .c

Copier

`cp src dest`

- Deux fonctionnements différents
  - Si `dest` est un répertoire, copie `src` dans le répertoire `dest`.
    - Dans ce cas, multiples copies possibles avec:
      `cp file1 file2 ... rep`
  - Sinon, copie src sous le nom dest
- L'option `-r` permet de copier récursivement un répertoire
  - sans `-r`, si src est un répertoire, erreur

Déplacer

`mv src dest`

déplace ou renomme

- `src` : fichier de type quelconque
- Si `dest` est un répertoire, déplace `src` dans le répertoire `dest`
  - Dans ce cas, multiples déplacements possibles avec:
    `mv file1 file2 ... rep`
- Sinon, déplace `src` sous le nom `dest`
  - Si dest est dans le même répertoire : renommage

Interface système pour l'utilisation des fichiers (1)

- **Dans l'interface des appels système, un fichier est représenté par un descripteur. Les descripteurs sont numérotés par des (petits) entiers.**
- **Pour utiliser un fichier, il faut l'ouvrir pour allouer un descripteur**

```
fd = open ("/home/machine/toto/fich", O_RDONLY, 0)
```

  - Fichier ouvert en lecture seule (on ne peut pas y écrire)
  - le numéro de descripteur alloué par le système est fd (renvoie –1 si erreur).
  - Les autres modes d'ouverture possibles sont O_RDWRITE et O_WRONLY.
  - Le fichier pourra être créé s'il n'existe pas

```
fd = open("/path/to/file", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
```

- **Quand on a fini d'utiliser un fichier, il faut le fermer `close (fd)`**

Ouvrir un fichier

Exemple

```
int fd; /* file descriptor */
if ((fd = open("/etc/hosts", O_RDONLY)) < 0) {
   perror("open");
   exit(1);
}
```

Commentaires

- L'OS maintient une tableau de fichiers ouverts par processus
  - Le numéro de descripteur de fichiers correspond à un indice dans ce tableau
- Chaque processus créé par un shell UNIX commence sa vie avec 3 fichiers ouverts associés au terminal:
  - 0: l'entrée standard
  - 1: la sortie standard
  - 2: la sortie d'erreur

Interface système pour l'utilisation des fichiers (2)

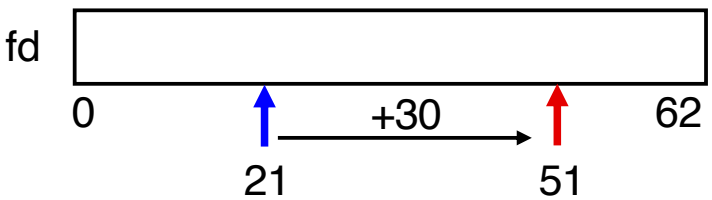
L'ouverture crée un **pointeur courant** (position dans le fichier), initialisé à 0.
Ce pointeur (invisible directement) est déplacé

- indirectement, par les opérations de lecture (`read`) et d'écriture (`write`). (cf détails plus loin)
- directement, par l'opération `lseek` (ci-dessous)

`lseek()` déplace le pointeur courant. Exemples :

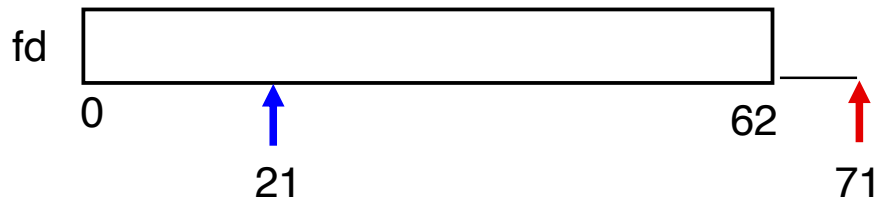
dépl. relatif

`lseek(fd, 30, SEEK_CUR)`
+30 octets depuis position courante



dépl. absolu

`lseek(fd, 71, SEEK_SET)`
place le pointeur à la position 71



Le pointeur peut être placé au-delà de la fin du fichier.

Lire un fichier

```
char buf[512];
int fd; /* file descriptor */
int nbytes; /* number of bytes read */

/* Open file fd ... */
/* Then read up to 512 bytes from file fd */
if ((nbytes = read(fd, buf, 512)) < 0) {
   perror("read");
   exit(1);
}
```

- `read()` copie les données du fichier depuis la position courante dans la mémoire, et met à jour la position courante.
- `read()` retourne le nombre d'octets lus (`nbytes`)
  - `nbytes < 0` indique qu'une erreur s'est produite
  - `nbytes < 512` est possible et n'est pas une erreur

Écrire un fichier

```
char buf[512];
int fd; /* file descriptor */
int nbytes; /* number of bytes read */

/* Open the file fd ... */
/* Then write up to 512 bytes from buf to file fd */
if ((nbytes = write(fd, buf, 512) < 0) {
   perror("write");
   exit(1);
}
```

- `write()` copie les données de la mémoire vers le fichier à la position courante, et met à jour la position courante.
- `write()` retourne le nombre d'octets écrits (`nbytes`)
  - `nbytes < 0` indique qu'une erreur s'est produite
  - `nbytes < 512` est possible et n'est pas une erreur

Un exemple simple de lecture/écriture

Copier les données de l'entrée standard vers la sortie standard octet par octet

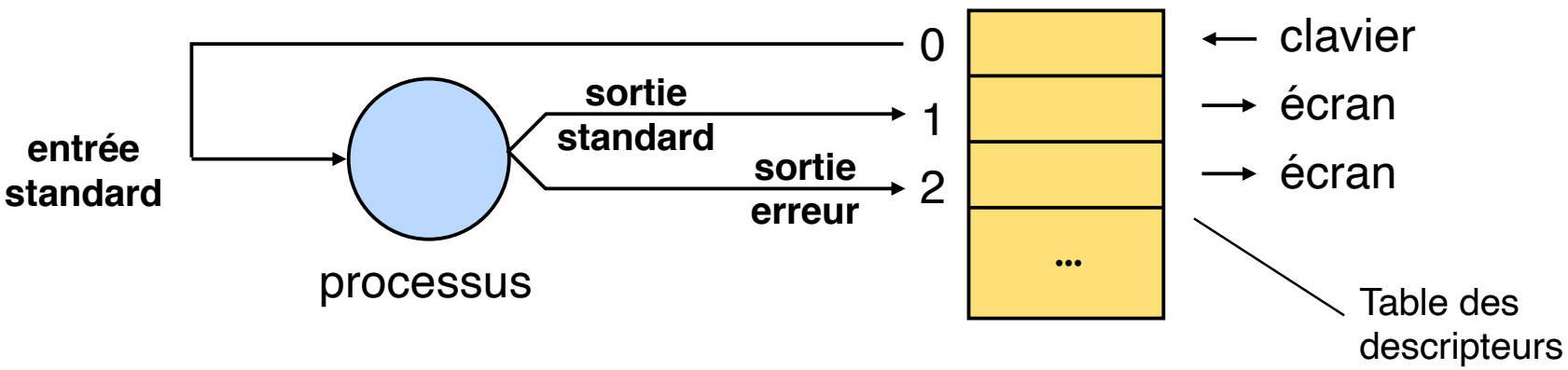
```
int main(void)
{
  char c;
  int len;

  while ((len = read(0 /*stdin*/, &c, 1)) == 1) {
    if (write(1 /*stdout*/, &c, 1) != 1) {
      exit(20);
    }
  }

  if (len < 0) {
    printf ("read-from-stdin-failed");
    exit (10);
  }
  exit(0);
}
```

Interface des fichiers

- Les primitives fournies par le noyau (`open`, `close`, `lseek`, `read`, `write`) sont celles de plus bas niveau. Leur utilisation est souvent délicate (gestion des erreurs, lectures tronquées, etc.)
- Il est en général préférable de les utiliser à travers des bibliothèques qui facilitent leur usage:
  - Par exemple, la bibliothèque dite standard, ensemble de fonctions d'accès de plus haut niveau inclus dans la bibliothèque C : `fopen`, `fread`, `fwrite`, `fscanf`, `fprintf`, `fflush`, `fseek`, `fclose` (et fonctions analogues pour les chaînes : `sprintf`, `sscanf`). Voir `man`.

Fichiers et flots d'entrée-sortie

- **Il y a un lien étroit entre fichiers et entrées-sorties**
- **Les organes d'entrée-sortie sont représentés par des fichiers particuliers (sous Unix, dans le répertoire /dev)**
- **Tout processus utilise des flots d'entrée-sortie qui peuvent être dirigés soit vers un fichier, soit vers un organe d'entrée-sortie : entrée standard, sortie standard, et sortie erreur**
- **Par convention, ces flots sont associés aux descripteurs 0, 1 et 2**



- **Les flots d'entrée-sortie peuvent être réorientés vers des fichiers**

Manipuler les flots d'entrée-sortie (commandes)

- **Dans le langage de commande, on réoriente les flots standard au moyen de $<$ et $>$**

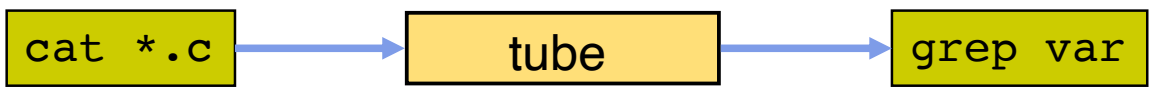
`cat fich` écrit le contenu de `fich` sur la sortie standard (l'affiche à l'écran)
`cat fich > fich1` copie `fich` dans `fich1` (qui est créé s'il n'existe pas)
`cat /dev/null > fich` crée le fichier vide `fich` s'il n'existe pas, sinon le rend vide

- **Les tubes (pipes) permettent de faire communiquer des processus.**
- **Un tube est un fichier anonyme qui sert de tampon entre deux processus fonctionnant en producteur-consommateur.**

```
cat *.c | grep var
```

a) crée un tube et deux processus : p1 qui exécute `cat *.c`, p2 qui exécute `grep var`

b) connecte la sortie de p1 à l'entrée du tube et l'entrée de p2 à la sortie du tube



Question : que fait la commande suivante ?

```
cat f1 f2 f3 | grep toto | wc -l > result
```

Agenda

Les systèmes de fichiers

Manipuler des fichiers

Les droits d'accès

Protection des fichiers (1)

- **Définition (générale) de la sécurité**
  - confidentialité : informations accessibles aux seuls usagers autorisés
  - intégrité : pas de modifications non désirées
  - contrôle d'accès : seuls certains usagers sont autorisé à faire certaines opérations
  - authentification : garantie qu'un usager est bien celui qu'il prétend être
- **Comment assurer la sécurité**
  - Définition d'un ensemble de règles (politiques de sécurité) spécifiant la sécurité d'une organisation ou d'une installation informatique
  - Mise en place de mécanismes (mécanismes de protection) pour assurer que ces règles sont respectées
- **Sécurité des fichiers (dans Unix)**
  - On définit
    - des types d'opérations sur les fichiers : lire, écrire, exécuter (contraintes de confidentialité, intégrité, contrôle d'accès)
    - des classes d'usagers
      - usager propriétaire du fichier
      - groupe propriétaire
      - tous les autres

Droits d'accès

- Toute opération sur un fichier est soumise à droits d'accès
  - Message d'erreur « *Permission non accordée* »

- 3 types d'accès
  - `r` : droit de lecture
    - Si répertoire, consultation de ses entrées (c.-à-.d, `ls` autorisé)
    - Sinon, consultation du contenu du fichier
  - `w` : droit d'écriture
    - Si répertoire, droit de création, de renommage et de suppression d'une entrée dans le répertoire
    - Sinon, droit de modification du contenu du fichier
  - `x` :
    - si répertoire, droit de traverser (c.-à-.d., `cd` autorisé)
    - sinon, droit d'exécution

Droits d'accès

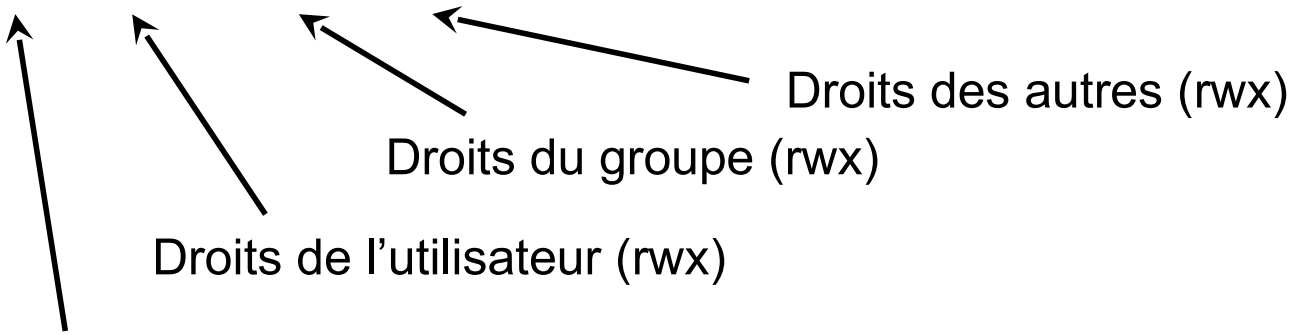
- 3 catégories d'utilisateurs :
  - Propriétaire (`u`)
  - Groupe propriétaire (`g`)
  - Tous les autres (`o`)

- Chaque catégorie possède ses types d'accès `r w x`

Droits d'accès – consultation

- `ls -ld` ⇒ donne les droits des fichiers

- Format de sortie de `ls -l`
  - `- --- --- ---`



Type du fichier :
d : répertoire
l : lien symbolique
- : fichier ordinaire

```
$ ls -l fichier
- rwx r-- --- fichier
$
```

Droits d'accès – modification

- Modification sur un fichier existant

  `chmod droit fichier` : change mode

- Droits à **!appliquer!** au fichier
  - Catégories : `u`, `g`, `o` ou `a` (= `all` c.-à-.d., `ugo`)
  - Opérations : Ajout (+), retrait (−), affectation (=)

```
$ ls -ld fichier
-rwx r-- --- fichier
$ chmod u-x fichier
$ ls -ld fichier
-rw- r-- --- fichier
$ chmod u+x fichier
$ ls -ld fichier
-rwx r-- --- fichier
```

Démonstration

```
$ cp /etc/passwd .
$ ls -l
total 4
-rw-r--r-- 1 gthomas users 1120 19 juil.  2016 passwd
$ chmod u-r passwd
$ cat passwd
cat: passwd: Permission non accordée
$ mkdir rep
$ ls -l
total 8
--w-r--r-- 1 gthomas users 1120 19 juil.  2016 passwd
drwxr-xr-x 2 gthomas users   68 19 juil.  2016 rep
$ cd rep/
$ cd ..
$ chmod u-x rep
$ cd rep
-bash: cd: rep: Permission non accordée
```

Droits d'accès initiaux

- Masque de droits d'accès **!retirés!** à la création de tout fichier
  - Commande `umask` (*user mask*)
  - Le masque est donné en octal (base 8) avec 3 chiffres (`u`, `g`, `o`)
  - En standard, masque par défaut = `022`
    - `r = 100` en binaire = 4 en octal, `w = 010 = 2`
    - Si droits retirés `--- -w- -w-`, alors droits appliqués `rw- r-- r--`
    - Le droit `x` est déjà retiré par défaut en général

  - Modification du masque grâce à la commande `umask`
    - Attention : `umask` sans effet rétroactif sur les fichiers préexistants
    - Attention : `umask` n'a d'effet que sur le `bash` courant

Démonstration

```
$ touch fichier_umask_defaut
$ ls -lh
-rw-rw-r-- 1 amina amina 0 oct. 2 10:49 fichier_umask_defaut
$ mkdir repertoire_umask_defaut
$ ls -lh
-rw-rw-r-- 1 amina amina 0     oct. 2 10:49 fichier_umask_defaut
drwxrwxr-x 2 amina amina   4,0K  oct.  2 10:50 repertoire_umask_defaut
$ umask 007
$ touch fichier_umask_nouveau
$ ls -lh
-rw-rw-r-- 1 amina amina 0     oct. 2 10:49 fichier_umask_defaut
-rw-rw---- 1 amina amina 0     oct. 2 10:52 fichier_umask_nouveau
drwxrwxr-x 2 amina amina 4,0K  oct. 2 10:50 repertoire_umask_defaut
$ mkdir repertoire_umask_nouveau
$ ls -lh
-rw-rw-r-- 1 amina amina 0     oct. 2 10:49 fichier_umask_defaut
-rw-rw---- 1 amina amina 0     oct. 2 10:52 fichier_umask_nouveau
drwxrwxr-x 2 amina amina 4,0K  oct. 2 10:50 repertoire_umask_defaut
drwxrwx--- 2 amina amina 4,0K  oct. 2 10:53 repertoire_umask_nouveau
```

Les groupes

Dans un système UNIX, les utilisateurs sont catégorisés en groupes

- Un utilisateur peut appartenir à plusieurs groupes

Groupes primaires et secondaires

- Chaque utilisateur appartient au moins à un groupe: son groupe primaire
  - Par défaut, le GID (*group identifier*) du groupe primaire est égal à l'UID (*user identifier*) de l'utilisateur
  - Groupe associé aux fichiers créés par l'utilisateur
- Un utilisateur peut aussi être intégré à d'autres groupes: des groupes secondaires

Les groupes – Quelques commandes

- Créer un groupe:
  - `groupadd <groupe>`
- Obtenir la liste des groupes d'un utilisateurs:
  - `groups user`
- Ajouter un utilisateur à un/des groupes:
  - `usermod -aG groupe1,groupe2,groupeN user`
- Changer le groupe primaire d'un utilisateur:
  - `usermod -g groupe user`

Propriétaire d'un fichier

Sous UNIX, un fichier appartient à un utilisateur et à un groupe.

- Par défaut, un fichier appartient à son créateur et à son groupe primaire

Quelques commandes

- Changer le propriétaire d'un fichier:
  - `chown user fichier`
- Changer le propriétaire et le groupe d'un fichier:
  - `chown user:group fichier`
- Changer seulement le groupe d'un fichier
  - `chgrp groupe fichier`

Mécanismes de délégation

Problème

Permettre d'exécuter un programme dont l'exécution nécessite des droits d'accès que n'ont pas les usagers potentiels

- Exemple: `/bin/passwd` pour changer son mot de passe = modifier un fichier dont le propriétaire est `root`

Setuid (et Setgit)

- Autre *permission* pour un fichier.
- Pour l'exécution de ce programme (et uniquement pour ce programme), un usager quelconque reçoit temporairement les droits de l'usager ou du groupe propriétaire du programme si le bit correspondant est à 1
  - Le droit 'x' est remplacé par 's' si le bit est à 1
  - Commande: `chmod u+s <filename>`

Sticky bit

Commande

```
chmod o+t repertoire
```

Principe

- Spécifie que pour les fichiers contenus dans le répertoire avec le sticky bit, seul le propriétaire du fichier peut le supprimer
  - Exemple d'utilisation: `/tmp`
- Par défaut, tout utilisateur ayant les droits d'écriture et d'exécution sur un répertoire peut en modifier le contenu