Introduction aux systèmes informatiques

Codage et Systèmes d'exploitation

Pôle ASR – Module M1101 – Semestre 1

Bruno BEAUFILS

⟨bruno.beaufils@univ-lille.fr⟩

https://beaufils.u-lille.fr

Université de Lille, IUT « A », Département informatique

Année 2020/2021

Systèmes d'exploitations

Philosophie

Règle générale

Sous Unix **TOUT EST FICHIER** . . . ou presque

- En interne (vue du noyau)
  - les fichiers ont tous la même structure
- En externe (vue de l'utilisateur)
  - différents *types* de fichiers :
    - catalogues (ou répertoires ou dossiers)
    - liens symboliques
    - spéciaux
    - tubes
    - sockets
    - ordinaires (ou réguliers)
  - représentation hiérarchique du stockage des fichiers

Philosophie

Règle générale

Sous Unix **TOUT EST FICHIER** . . . ou presque

- En interne (vue du noyau)
  - les fichiers ont tous la même structure
- En externe (vue de l'utilisateur)
  - différents *types* de fichiers :
    - catalogues (ou répertoires ou dossiers)
    - liens symboliques
    - spéciaux
    - tubes
    - sockets
    - ordinaires (ou réguliers)
  - représentation hiérarchique du stockage des fichiers

Catalogues

Définition

Un catalogue (*directory*), ou répertoire, ou dossier, est un fichier qui contient une liste de fichiers.

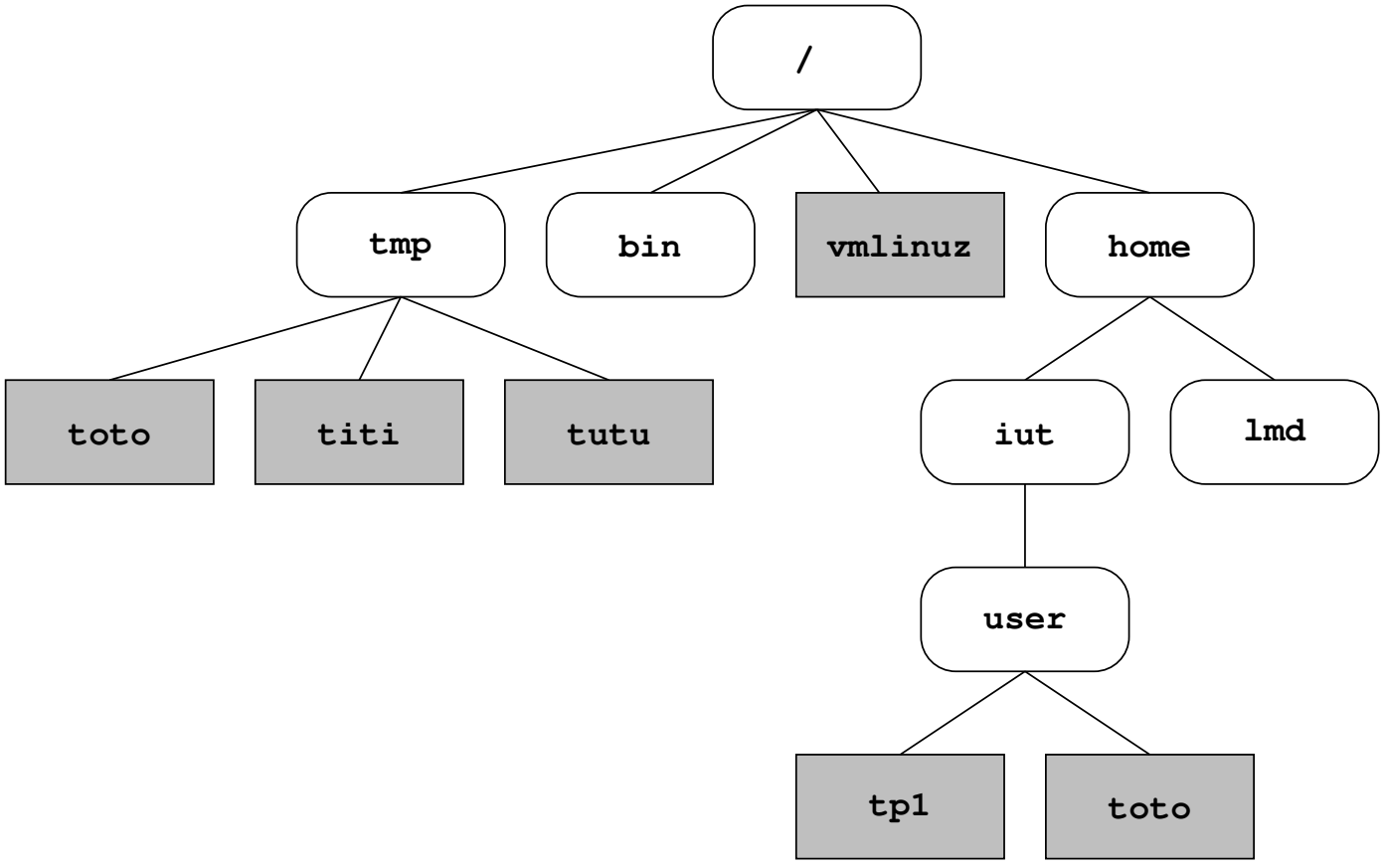
Un répertoire *contient* d'autres fichiers et peut donc contenir un ou des répertoires.

➠ Notion de hiérarchie (ou d'arbre)

Toutes les versions d'Unix ont une hiérarchie unique, dont le sommet est nommé « / » (*slash*).

Ce répertoire de base est la racine (*root*) de l'arbre hiérarchique.

Ce répertoire a toujours comme inode la valeur 2



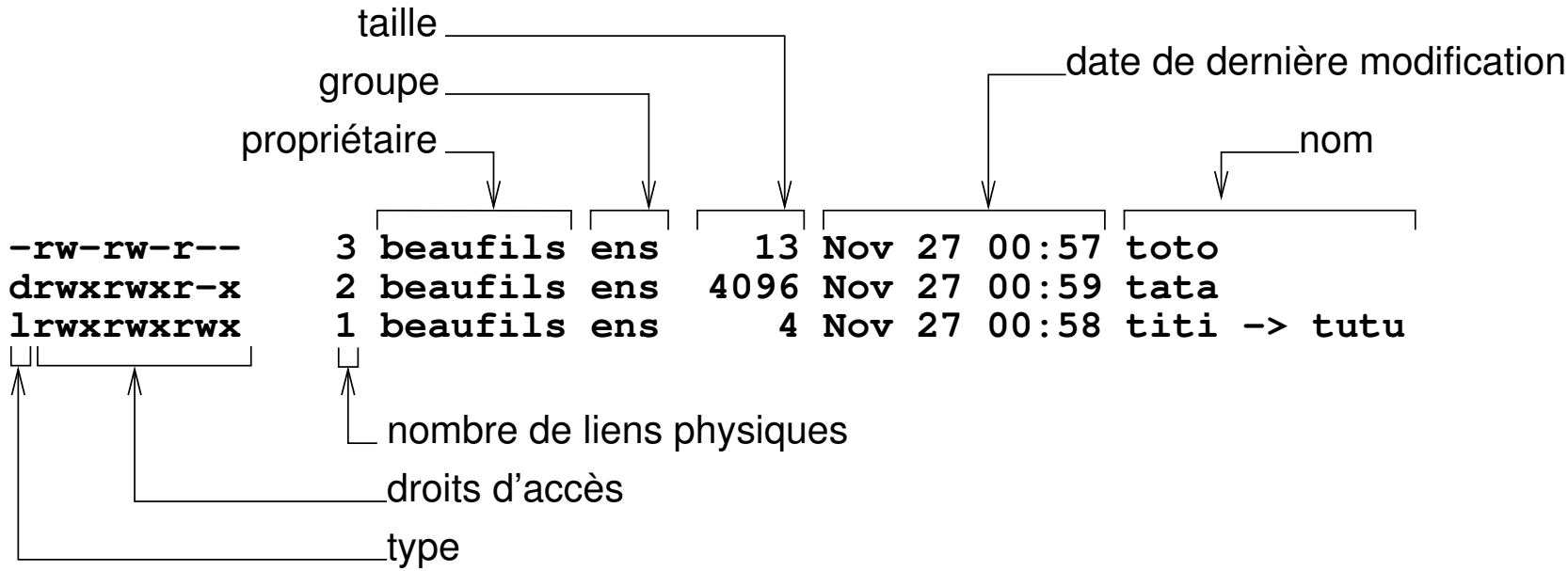

```
/
|
|-- tmp
|    |
|    |-- toto
|    |-- titi
|    `-- tutu
|-- bin
|-- vmlinuz
`-- home
     |
     |-- iut
     |    |
     |    `-- user
     |         |-- tp1
     |         `-- toto
     `-- lmd
```

Quelques commandes utilisables à propos des répertoires :

`pwd`	permet d'obtenir le nom absolu du répertoire de travail courant
`cd`	permet de changer le répertoire de travail courant
`ls`	permet d'obtenir la liste des fichiers contenus dans un répertoire. Il existe de très nombreuses options parmi lesquelles : `-a` permet de voir les fichiers cachés `-i` permet de voir les inodes associées `-l` permet d'avoir les informations pour chaque fichier sur une ligne
`mkdir`	permet de créer un répertoire
`rmdir`	permet de supprimer un répertoire vide
`rm`	permet de supprimer une (ou des) entrée(s) du répertoire

Sans argument `ls` *liste le contenu du répertoire de travail courant*

ls -l



Type	**Caractères**
fichier régulier	`-`
répertoire	`d`
lien (symbolique)	`l`
tube	`p`
socket	`s`
spécial	`c` ou `b`

Hiérarchie standard Unix

`/bin`	Commandes utilisateurs essentielles
`/dev`	Fichiers de périphériques
`/etc`	Fichiers de configuration spécifique à la machine
`/home`	Répertoires des utilisateurs
`/lib`	Librairies partagées
`/sbin`	Commandes d'administration essentielles
`/tmp`	Fichiers temporaires
`/usr`	Seconde hiérarchie
`/var`	Données variables
`/usr/bin`	La plupart des commandes utilisateurs
`/usr/include`	Fichier d'entêtes pour les programmes C
`/usr/lib`	Librairies
`/usr/local`	Hiérarchie locale
`/usr/sbin`	Commandes d'administrations non-vitales
`/usr/share`	Données indépendantes de l'architecture
`/usr/src`	Code source

Plus de détails sur l'effort de standardisation : `http://www.pathname.com/fhs/`

Tous les répertoires contiennent obligatoirement dans leur liste deux fichiers :

- « `.` » qui est un synonyme pour le répertoire lui-même
- « `..` » qui est un synonyme pour le répertoire qui le contient (son père)

Les fichiers dont le nom commence par un point « `.` » sont appelés *fichiers cachés* (par exemple par défaut la commande `ls` ne les montre pas).

Chemins

Pour identifier un fichier dans la hiérarchie on a besoin :

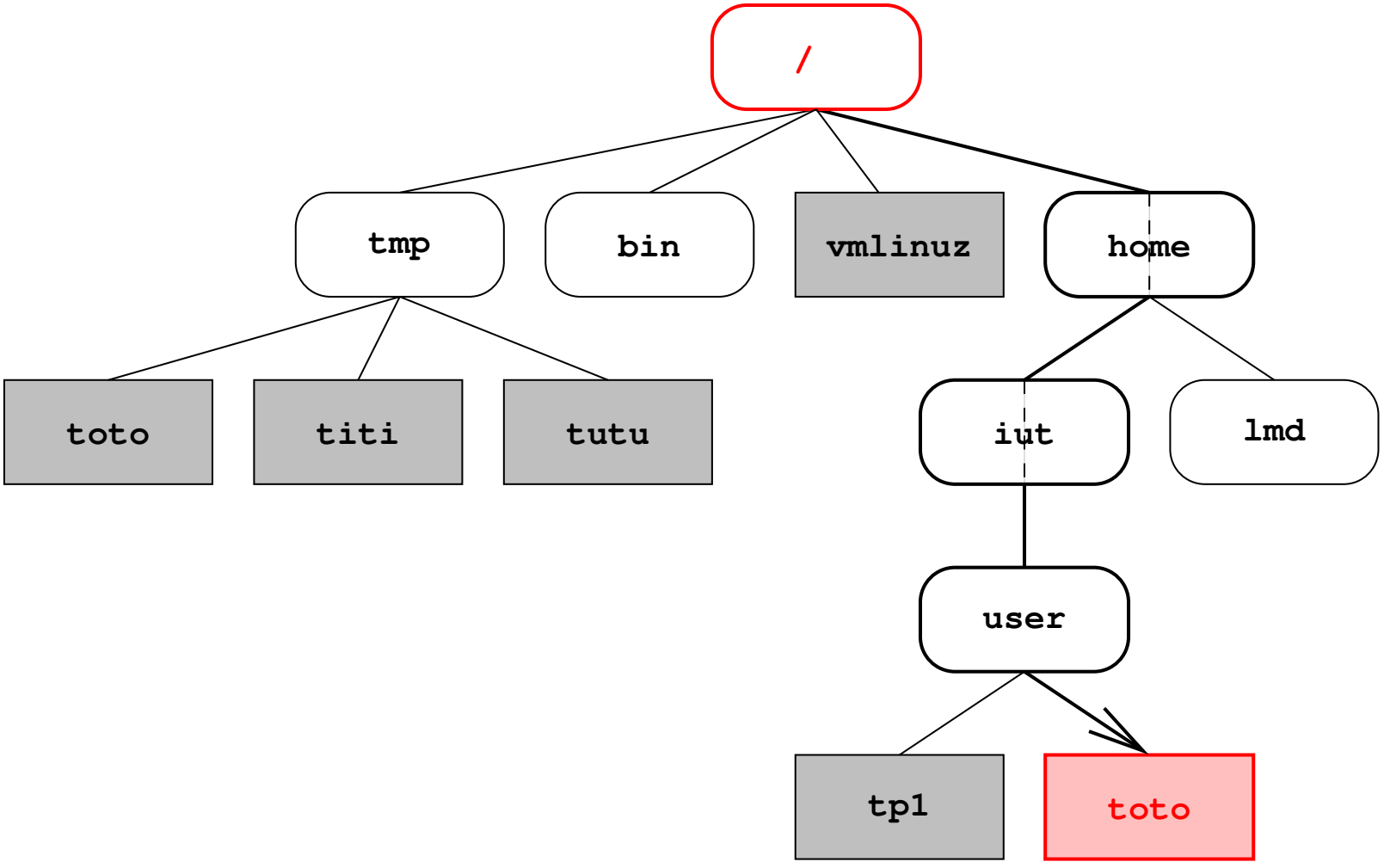
1. du chemin jusqu'au répertoire dans lequel il est stocké
2. de son nom de base

Définition

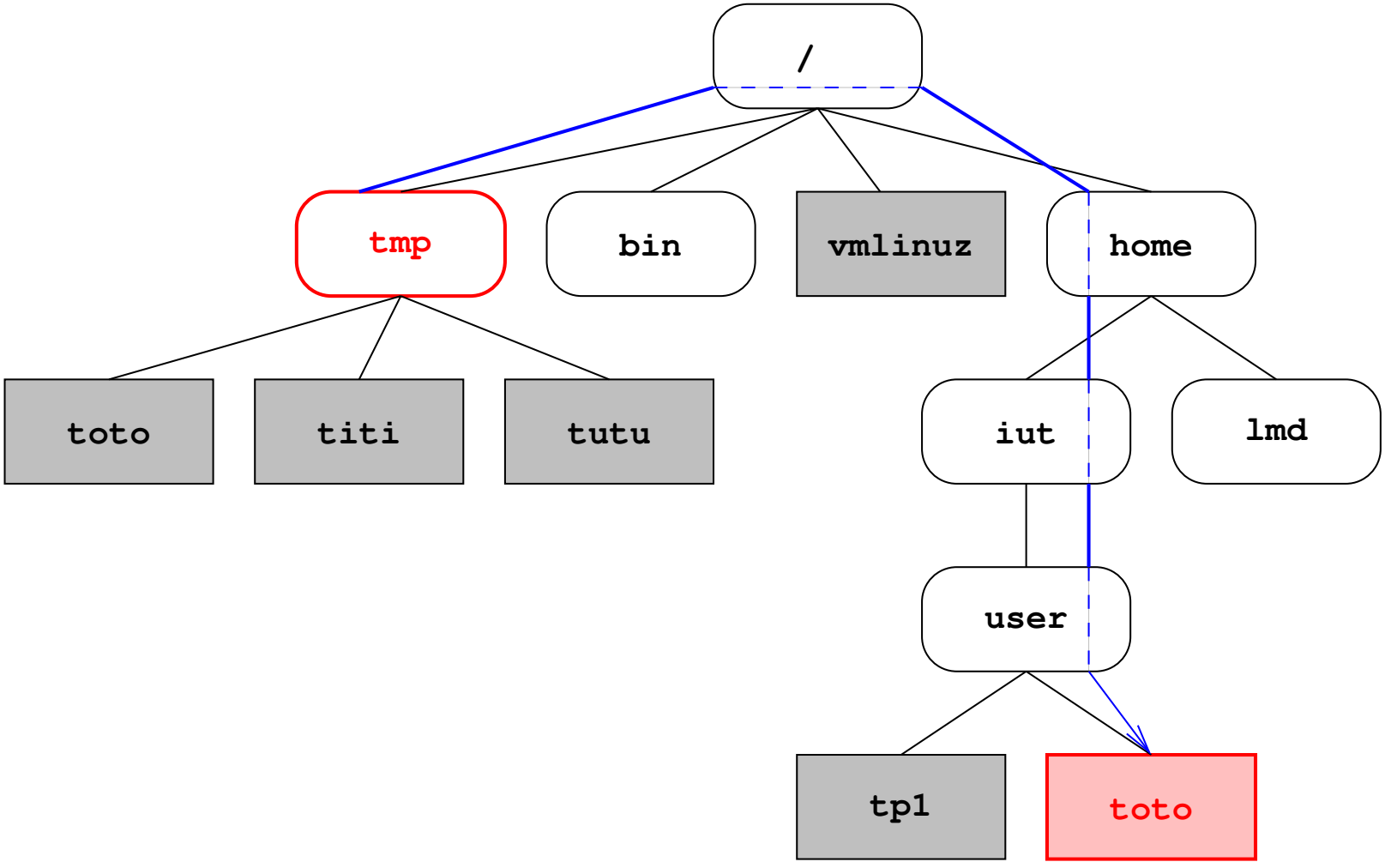
chemin = point de départ + liste des répertoires à traverser pour arriver au répertoire destination

La liste des répertoires est composée de répertoires séparés les uns des autres par le caractère « / »

- si le point de départ est la racine (et chemin le plus court possible)
  ➠ **chemin absolu**
- sinon
  ➠ **chemin relatif à un autre répertoire**

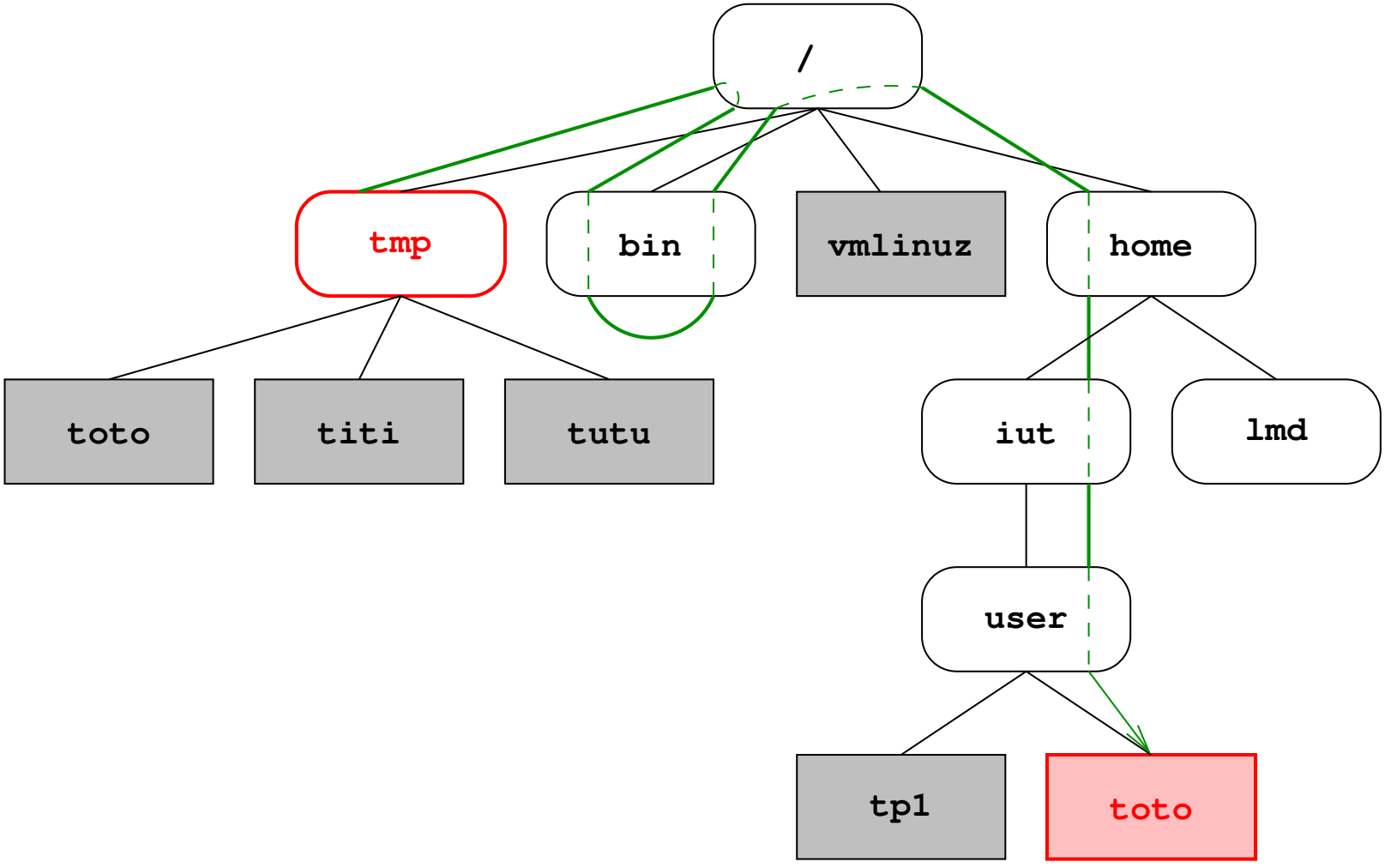


chemin absolu ➙ `/home/iut/user/toto`



chemin absolu ➟ `/home/iut/user/toto`

chemins relatifs à `/tmp` ➟ `../home/iut/user/toto`



chemin absolu ➟ `/home/iut/user/toto`

chemins relatifs à `/tmp` ➟ `../home/iut/user/toto`

ou `../bin/../home/iut/user/toto`

… etc …

Fichiers réguliers

Définition

Un fichier régulier est un fichier qui n'est ni un catalogue, ni un lien, ni un fichier spécial, ni un tube, ni une socket.

Quelques commandes utilisables sur des fichiers réguliers :

`stat`	permet d'afficher les caractéristiques de base de fichier(s)
`touch`	permet de modifier les caractéristiques de dates de fichier(s). Cette commande permet également de créer un (ou des) fichier(s) vide(s)
`od`	permet d'afficher les octets d'un fichier sous différents formats
`cat`	permet d'afficher le contenu de fichier(s)
`file`	permet de déterminer la convention de structure que respecte le contenu du fichier (donc son *type applicatif*)
`diff`	permet de déterminer les différences entre le contenu de deux fichiers

Liens symboliques

Définition

Un **lien symbolique** (*soft link*) est un fichier (de type lien) qui contient le chemin et le nom d'un autre fichier.

Les accès à un lien ne sont rien d'autre que des redirections vers un autre fichier : les commandes qui manipulent un fichier lien manipule en fait le fichier dont le chemin est stocké dans le lien.

➠ Un lien est donc un raccourci (ou un alias) vers un autre fichier

Le contenu du fichier doit être un chemin

- soit absolu
- soit relatif. Dans ce cas le chemin doit être **valide depuis le répertoire dans lequel se trouve le fichier**.

Manipulation du système de fichiers

Les commandes de manipulation des fichiers ont souvent la syntaxe suivante :

`commande` ⟨*source*⟩ . . . ⟨*destination*⟩

- ⟨*source*⟩ et ⟨*destination*⟩ désigne chacun un chemin

`cp` permet de **dupliquer** un fichier :

1. création (ou modification) d'un fichier en dupliquant le contenu d'un autre fichier
2. création (ou modification) d'une entrée dans un répertoire

`mv` permet de **déplacer** un fichier (donc aussi de le renommer) :

1. suppression d'une entrée dans un répertoire
2. création (ou modification) d'une nouvelle entrée dans un répertoire

`ln` permet de **surnommer** un fichier (donc aussi de le renommer) :

1. création (ou modification) d'une nouvelle entrée dans un répertoire

Structuration

Définition

Vue du noyau un fichier est une suite non-structurée d'octets (*byte stream*)

Pas de structuration directe au niveau du noyau mais possible au niveau des applications.

Exemple : les fichiers textes

- fichiers constitués d'une séquence de lignes.
- ligne constituée de caractères terminée par le caractère de passage à la ligne.
- caractère représenté par un octet suivant le code ASCII.
- caractère de passage à la ligne est le caractère de code 10 « `\n` ».

Cette structuration n'est qu'une convention utilisée par des programmes.

Inodes (1)

- Les caractéristiques d'un fichier sont stockées dans une structure de données ..... inode
- Le système gère une table de toutes les inodes disponibles ..... système de fichiers
- Une inode est repérée par son numéro dans cette table ..... son numéro d'inode
- Il y a un système de fichiers par zone de stockage matérielle (partition d'un disque dur par exemple)

Définition

Un fichier est repéré de manière unique par :

1. le système de fichier auquel il est attaché
2. son numéro d'inode

Inodes (2)

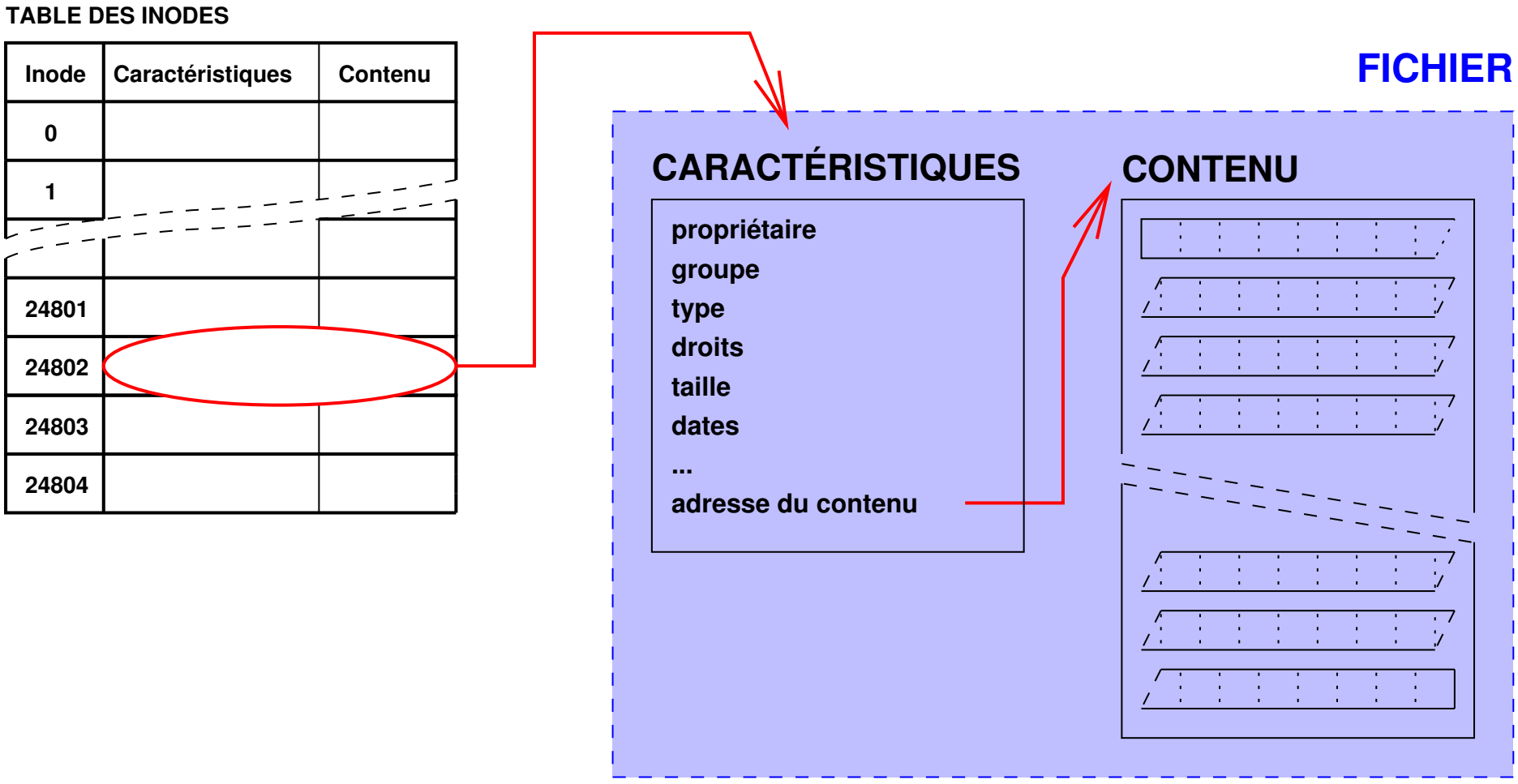
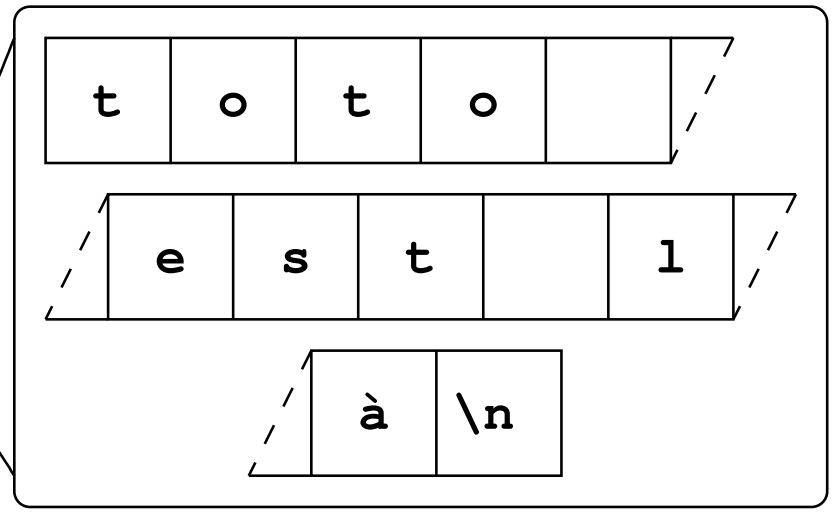


TABLE DES INODES
Inode
Caractéristiques
Contenu
0
1
24801
24802
24803
24804
FICHIER
CARACTÉRISTIQUES
propriétaire
groupe
type
droits
taille
dates
...
adresse du contenu
CONTENU

Fichiers réguliers (vue noyau)

TABLE DES INODES

Inode	Caractéristiques	Contenu
0		
1		
24801	... Répertoire ...	...
24802	... Régulier ...	
24803	... Lien ...	tutu



Fichiers catalogues (vue noyau)

Catalogues	=	Fichier comme un autre
	=	Caractéristiques + Contenu
Contenu	=	Liste de fichiers
	=	Ensemble de couples : (nom, inode)

TABLE DES INODES

Inode	Caractéristiques	Contenu
0		
1		
24801	... Répertoire ...	
24802	... Régulier ...	toto est là
24803	... Lien ...	tutu

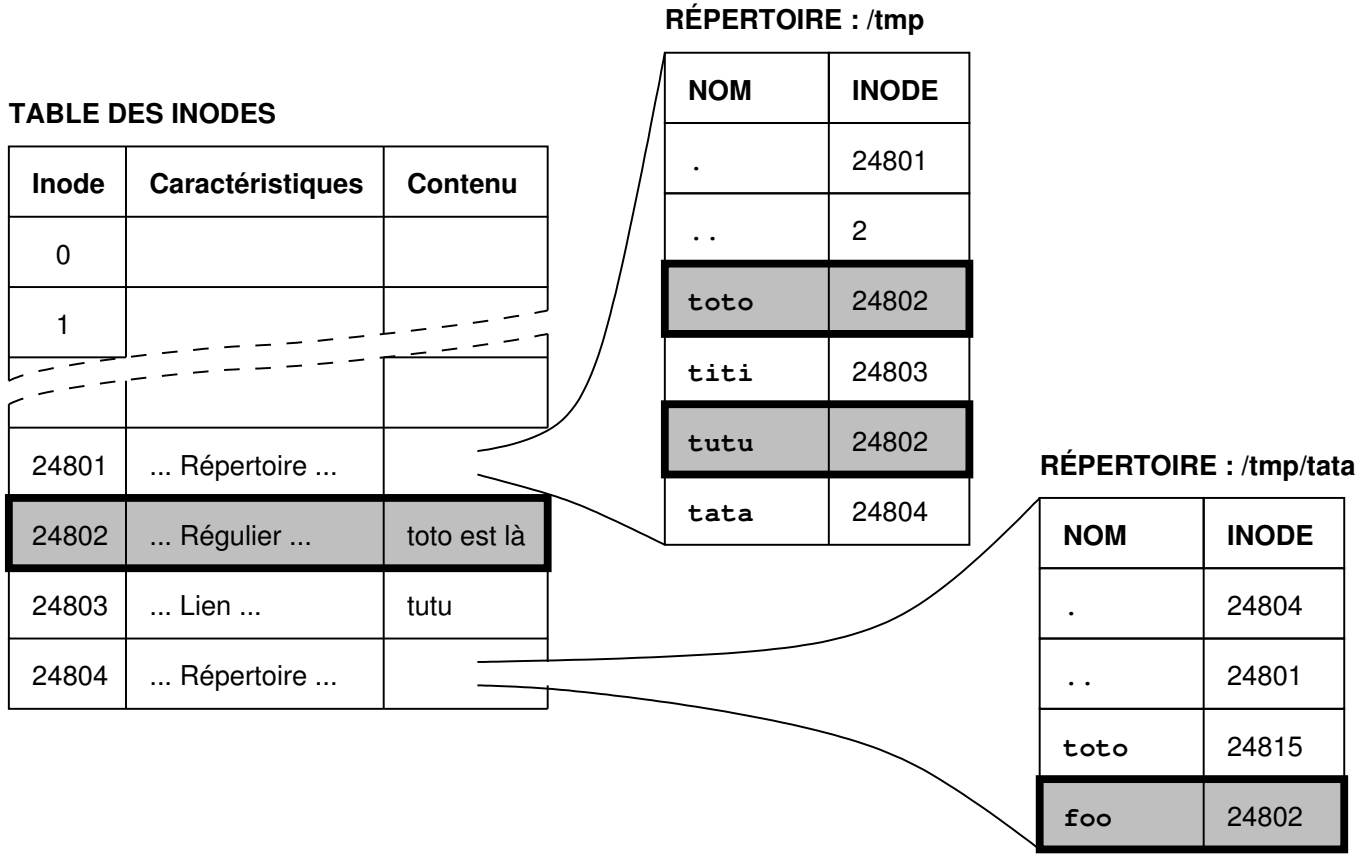
RÉPERTOIRE : /tmp

NOM	INODE
.	24801
..	2
`toto`	24802
`titi`	24803
`tutu`	24802
`tata`	24804

➥ *entrée* d'un répertoire
(un des couples de la liste)

Noms des fichiers (vue noyau)

Le nom de fichier n'est rien d'autre qu'une entrée dans un répertoire. Un nom est donc un **lien physique** (*hard link*) vers un fichier.



Plusieurs entrées de répertoires peuvent utiliser la même inode.

➠ **Un même fichier peut avoir plusieurs noms.**

Abus de langage

Par abus de langage on a donc 2 notions différentes :

- les **liens physiques** ou **hard**, plusieurs entrées de répertoires utilisant la même inode ..... (fichiers de type régulier)

- les **liens symboliques** ou **soft**, plusieurs inodes différentes dont le contenu désigne un même fichier régulier ..... (fichiers de type lien)

La commande `ln` permet de créer des liens :

- sans option elle permet de créer des liens physiques
- avec l'option `-s` elle permet de créer des liens symboliques

Utilisateurs/Groupes

Unix est un système multi-utilisateurs. Les utilisateurs y sont rassemblés par groupe. Chaque utilisateur est donc identifié par le système par :

1. son *login* ..... au niveau noyau c'est un numéro unique : l'`uid`
2. son *groupe* ..... au niveau noyau c'est un numéro unique : le `gid`

Le système gère la correspondance entre identifiant symbolique et numérique via des fichiers textes :

- login et `uid` via le fichier `/etc/passwd`
- groupe et `gid` via le fichier `/etc/group`

Un utilisateur peut appartenir à plusieurs groupes, mais possède un groupe principal (spécifié dans le fichier `/etc/passwd`) dans lequel il est enregistré lors de chaque connexion.

Droits d'accès

Chaque fichier :

- appartient à un utilisateur (son *propriétaire*) et à un groupe.
- posssède des droits d'utilisation applicables :
  1. à son propriétaire
  2. aux utilisateurs appartenant à son groupe
  3. aux utilisateurs n'appartenant pas à son groupe

Pour chacune de ces trois catégories, il existe trois types de droits :

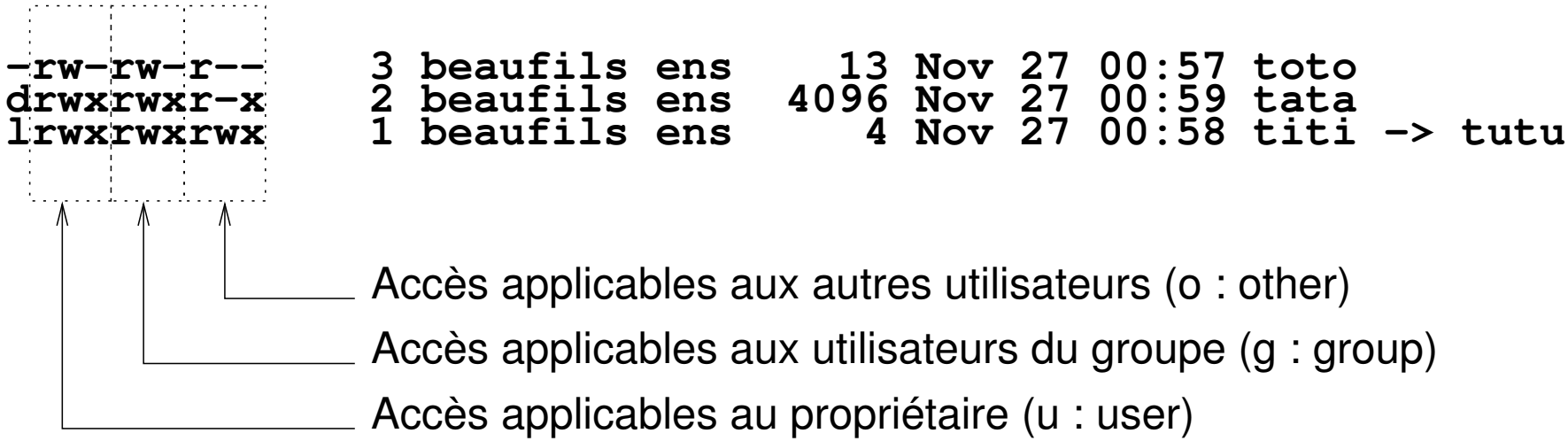
1. **lecture** : autorise la lecture du contenu du fichier
2. **écriture** : autorise la modification du contenu du fichier
3. **exécution/franchissement** :
   - autorise l'exécution d'un fichier régulier,
   - permet de traverser un répertoire

Remarque

Pour manipuler le système de fichier (copie, déplacement, etc.) un utilisateur doit avoir les droits correspondants sur les fichiers qu'il veut manipuler

L'option `-l` de la commande `ls` permet de voir les droits d'accès d'un fichier. Pour chacun des trois cas d'applicabilité les droits sont affichés par une chaîne de caractère avec la représentation suivante :

- `r` : l'accès en lecture est autorisé
- `w` : l'accès en écriture est autorisé
- `x` : l'accès en exécution/franchissement est autorisé
- `-` : à la place de `r`, `w` ou `x` signifie que l'accès correspondant n'est pas attribué.



chmod

Définition

Le mode d'utilisation d'un fichier est l'ensemble de ses droits d'accès.

La commande `chmod` permet au propriétaire d'un fichier de modifier son mode d'utilisation.

La syntaxe de `chmod` est la suivante :

`chmod` ⟨*mode*⟩ ⟨*fichiers*⟩

Le mode peut être précisé de deux manières :

- via la spécification des modifications à effectuer sur le mode courant :
  ➟ forme **symbolique**.
- via la spécification complète du nouveau mode :
  ➟ forme **numérique octale** (base 8)

`chmod` (forme symbolique)

Les modifications à effectuer sur le mode courant sont spécifiées par un code dont la syntaxe est :

$$\langle personne \rangle \langle action \rangle \langle accès \rangle$$

⟨*personne*⟩	
`u`	propriétaire
`g`	groupe
`o`	autres
`a`	tous

⟨*action*⟩	
`+`	ajouter
`-`	enlever
`=`	initialiser

⟨*accès*⟩	
`r`	lecture
`w`	écriture
`x`	exécution/franchissement

- Il ne peut y avoir qu'une action par code
- Plusieurs modifications peuvent être spécifiées si elles sont séparées les unes des autres par des virgules « `,` ».

Les différentes combinaisons de droits d'accès peuvent être représentées par :

symbolique	binaire	octal
`---`	000	0
`--x`	001	1
`-w-`	010	2
`-wx`	011	3
`r--`	100	4
`r-x`	101	5
`rw-`	110	6
`rwx`	111	7

Le mode d'un fichier peut alors être spécifié par un nombre en base 8, dont les chiffres représentent, de gauche à droite, les droits d'accès pour :

1. le propriétaire du fichier
2. les membres du groupe du fichier
3. les autres utilisateurs

755 représente les droits `rwx r-x r-x`

777 représente les droits `rwx rwx rwx`

644 représente les droits `rw- r-- r--`

`umask`

Lorsqu'un programme crée un fichier, il spécifie les droits d'accès qu'il demande pour ce fichier.

Certains des droits demandés seront accordés d'autres seront refusés en fonction d'un *masque de protection*.

La commande `umask` permet :

- de connaître la valeur du masque si elle est utilisée sans argument
- de modifier la valeur du masque si elle est utilisée avec un argument

Dans tous les cas elle utilise des masques sous forme numérique octale.

Les droits accordés sont déterminés en retirant aux droits demandés les droits spécifiés par le masque.

Pour les répertoires :

droits demandés :	`rwx rwx rwx`	777
- masque :	`--- -w- rwx`	027
droits accordés :	`rwx r-x ---`	750

Pour les fichiers ordinaires :

droits demandés :	`rw- rw- rw-`	666
- masque :	`--- -w- -w-`	022
droits accordés :	`rw- r-- r--`	644

```
$ umask
0

$ mkdir toto
$ ls -l
total 1
drwxrwxrwx    2 beaufils ens     1024 Nov  6 07:36 toto


$ rmdir toto
$ umask 022
$ mkdir toto
$ ls -l
total 1
drwxr-xr-x    2 beaufils ens     1024 Nov  6 07:37 toto


$ rmdir toto
$ umask 077
$ mkdir toto
$ ls -l
total 1
drwx------    2 beaufils ens     1024 Nov  6 07:37 toto
$ umask
77
```