Initiation à l'environnement Unix
CM5 : fichiers, répertoires, inodes, et construction case en shell

Pierre Rousselin

Université Paris 13
L1 informatique
octobre 2021

Répertoires et inodes

Intermède de shell : construction `case`

Ce que ne contient *pas* un fichier

- son (ou ses) nom(s) dans l'arborescence ;
- ses permissions ;
- le nom du périphérique qui le contient et son emplacement sur celui-ci (informations permettant de trouver physiquement le contenu du fichier)
- ...

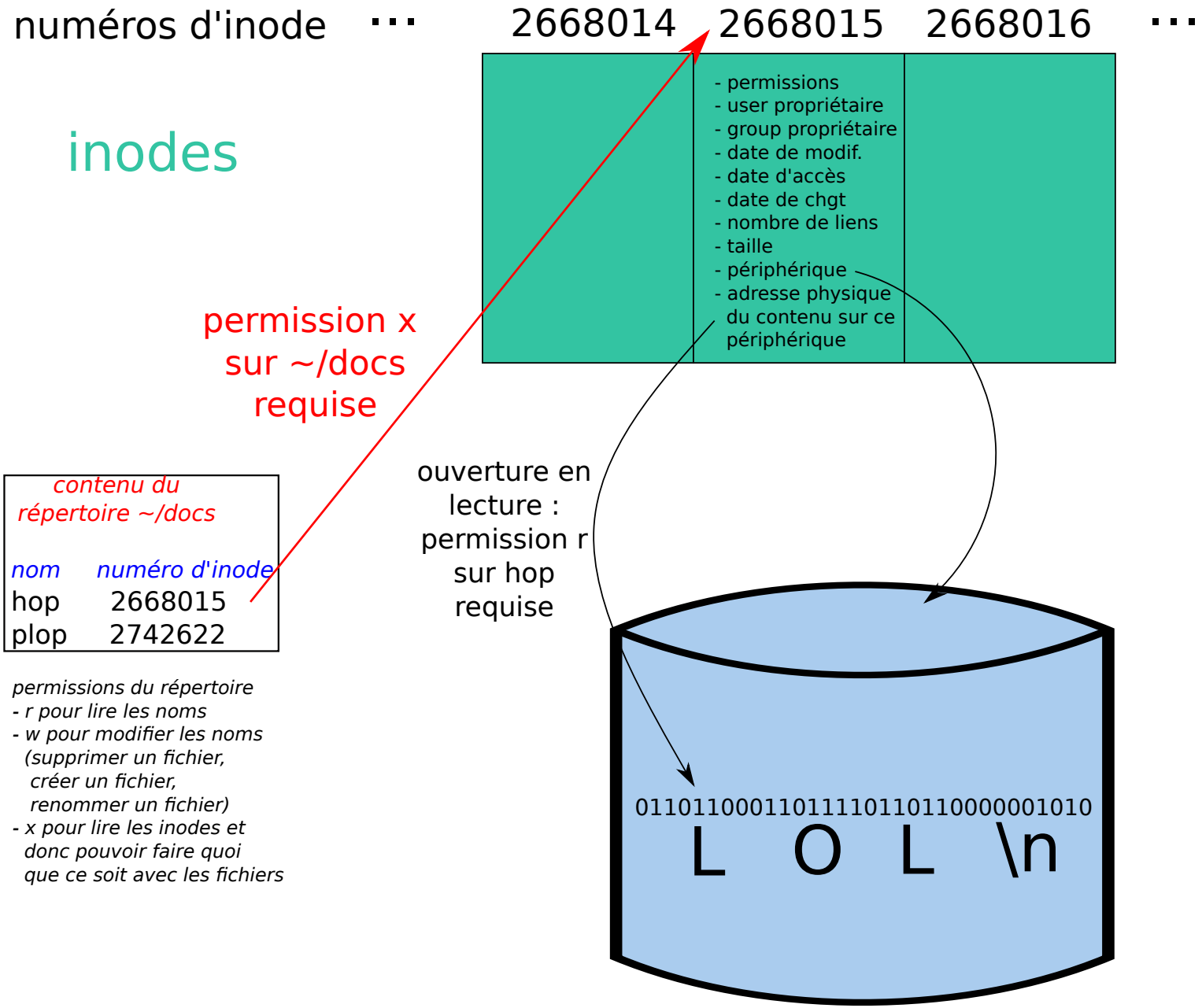
Bref, un fichier ne contient que les données écrites dessus et rien d'autre.

Alors où sont ces données ?

- Le ou les noms sont dans des répertoires ;
- les autres attributs du fichier sont dans une structure gérée par le système appelée inode ;
- un répertoire n'est qu'un tableau permettant d'associer à un nom un numéro appelé numéro d'inode qui permet de récupérer les informations de l'inode.

Ce sont les répertoires qui nomment les fichiers.

```
$ cat ~/docs/hop
lol
```



Nom, inodes et attributs

- La commande `ls`, les développements de noms de chemins, etc permettent de consulter les noms contenus dans un répertoire ;
- la commande `ls` avec l'option `-i` affiche le numéro d'inode ;
- la commande `ls` avec l'option `-l` affiche des attributs contenus dans l'inode ;
- ce que fait aussi la commande non standard `stat`.

Attributs de fichier

- numéro d'inode ;
- utilisateur propriétaire ;
- groupe propriétaire ;
- permissions ;
- dates :
  - de création (le fichier a été créé) ;
  - de dernière modification (le fichier a été ouvert en écriture) ;
  - de dernier accès (le fichier a été ouvert en lecture) ;
  - de dernier changement (l'inode du fichier a changé, par exemple ses permissions).
- taille ;
- périphérique sur lequel se trouve le fichier ;
- nombre de liens physiques, c'est-à-dire de noms dans l'arborescence.

Permissions

ugo : user, group, others

rwx : read, write, execute

```
$ ls -ld hop plop /root
--w-r--r--. 1 pierre dialout   0 25 oct.  14:55 hop
-rw-r-----. 1 alice  wheel     0 25 oct.  14:55 plop
dr-xr-x---. 1 root   root    258 19 oct.  13:09 /root
$ id
uid=1000(pierre) gid=1000(pierre) groupes=pierre,wheel,dialout
```

Ici, l'utilisateur a les permissions :

Permissions

ugo : user, group, others
rwx : read, write, execute

```
$ ls -ld hop plop /root
--w-r--r--. 1 pierre dialout   0 25 oct.  14:55 hop
-rw-r-----. 1 alice  wheel     0 25 oct.  14:55 plop
dr-xr-x---. 1 root   root    258 19 oct.  13:09 /root
$ id
uid=1000(pierre) gid=1000(pierre) groupes=pierre,wheel,dialout
```

Ici, l'utilisateur a les permissions :

- `w` (et c'est tout) sur `hop` car il est utilisateur propriétaire, donc ce sont les permissions de `u` qui s'appliquent ;

Permissions

ugo : user, group, others
rwx : read, write, execute

```
$ ls -ld hop plop /root
--w-r--r--. 1 pierre dialout   0 25 oct.  14:55 hop
-rw-r-----. 1 alice  wheel     0 25 oct.  14:55 plop
dr-xr-x---. 1 root   root    258 19 oct.  13:09 /root
$ id
uid=1000(pierre) gid=1000(pierre) groupes=pierre,wheel,dialout
```

Ici, l'utilisateur a les permissions :

- `w` (et c'est tout) sur `hop` car il est utilisateur propriétaire, donc ce sont les permissions de `u` qui s'appliquent ;
- `r` (et c'est tout) sur `plop` car il n'est pas utilisateur propriétaire du fichier et l'un des groupes auxquels il appartient (`wheel`) est groupe propriétaire du fichier `plop` donc ce sont les permissions `g` qui s'appliquent ;

Permissions

ugo : user, group, others
rwx : read, write, execute

```
$ ls -ld hop plop /root
--w-r--r--. 1 pierre dialout   0 25 oct.  14:55 hop
-rw-r-----. 1 alice  wheel     0 25 oct.  14:55 plop
dr-xr-x---. 1 root   root    258 19 oct.  13:09 /root
$ id
uid=1000(pierre) gid=1000(pierre) groupes=pierre,wheel,dialout
```

Ici, l'utilisateur a les permissions :

- `w` (et c'est tout) sur `hop` car il est utilisateur propriétaire, donc ce sont les permissions de `u` qui s'appliquent ;
- `r` (et c'est tout) sur `plop` car il n'est pas utilisateur propriétaire du fichier et l'un des groupes auxquels il appartient (`wheel`) est groupe propriétaire du fichier `plop` donc ce sont les permissions `g` qui s'appliquent ;
- aucune sur le répertoire `/root` car il n'est pas son utilisateur propriétaire et n'appartient pas à son groupe propriétaire, ce sont les permissions `o` qui s'appliquent.

Permissions et fichiers normaux

Sur les fichiers normaux (en fait les fichiers de tous types sauf les répertoires) :

- `r` permet *d'ouvrir un fichier en lecture*, donc de demander au noyau de pouvoir *lire* (*read*) les octets qu'il contient. Exemples : `cat message.txt`, `eog image.jpg`, ...
- `w` permet *d'ouvrir un fichier en écriture*, donc de pouvoir *écrire* (*write*) des octets dedans. Exemple : `printf 'lol\n' >>fichier.txt` écrit les octets correspondants à la chaîne `lol` (suivi de l'octet `0x`) à la fin de `fichier.txt`, si on peut l'ouvrir en écriture.
- `x` permet *d'exécuter le fichier*, c'est-à-dire de l'envoyer au noyau pour qu'un processus exécute les instructions qu'il contient. Ces instructions sont en langage machine (ou presque) dans le cas d'un programme compilé, ou bien dans un langage interprété (shell, python, ...) dans le cas d'un script commençant par un shebang.

Permissions et répertoire

Image mentale : un répertoire n'est qu'un tableau à deux colonnes : nom et numéro d'inode.

- r permet de *lire les noms* (et seulement les noms), donc en pratique `ls` sans option, autocomplétion dans `bash` et développement de noms de chemins ;
- w permet de *modifier le tableau* en ajoutant, supprimant ou modifiant des noms, mais sans `x`, `w` ne permet pas de faire quoi que ce soit ;
- x permet d'associer un numéro inode à un nom, ce qui est indispensable pour toute opération sur un fichier à partir du nom donné par ce répertoire.

Permissions, exemples

- `ls rep/` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` : besoin de `r` sur `rep/`
- `rm rep/a` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` : besoin de `r` sur `rep/`
- `rm rep/a` : besoin de `wx` sur `rep/` : modification du répertoire et accès à l'inode pour diminuer le nombre de liens (voir plus loin)
  Aucune permission sur le fichier `rep/a` n'est nécessaire, c'est la permission sur le répertoire qui permet de `rm` !
- `cat rep/a` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` : besoin de `r` sur `rep/`
- `rm rep/a` : besoin de `wx` sur `rep/` : modification du répertoire et accès à l'inode pour diminuer le nombre de liens (voir plus loin)
  Aucune permission sur le fichier `rep/a` n'est nécessaire, c'est la permission sur le répertoire qui permet de `rm` !
- `cat rep/a` : besoin de `x` sur `rep/` et `r` sur `rep/` (besoin d'accéder à l'inode pour accéder aux permissions et aux données !)
- `cp rep/a rep/b` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` : besoin de `r` sur `rep/`
- `rm rep/a` : besoin de `wx` sur `rep/` : modification du répertoire et accès à l'inode pour diminuer le nombre de liens (voir plus loin)
  Aucune permission sur le fichier `rep/a` n'est nécessaire, c'est la permission sur le répertoire qui permet de `rm` !
- `cat rep/a` : besoin de `x` sur `rep/` et `r` sur `rep/` (besoin d'accéder à l'inode pour accéder aux permissions et aux données !)
- `cp rep/a rep/b` : besoin de `wx` sur `rep/`, `r` sur `rep/a` et `w` sur `rep/b`
- `rm rep/*` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` : besoin de `r` sur `rep/`
- `rm rep/a` : besoin de `wx` sur `rep/` : modification du répertoire et accès à l'inode pour diminuer le nombre de liens (voir plus loin)
  Aucune permission sur le fichier `rep/a` n'est nécessaire, c'est la permission sur le répertoire qui permet de `rm` !
- `cat rep/a` : besoin de `x` sur `rep/` et `r` sur `rep/` (besoin d'accéder à l'inode pour accéder aux permissions et aux données !)
- `cp rep/a rep/b` : besoin de `wx` sur `rep/`, `r` sur `rep/a` et `w` sur `rep/b`
- `rm rep/*` : besoin de `rwx` sur `rep/`, `r` pour le développement de noms de chemins, `w` pour supprimer les noms contenus dans le répertoire et `x` pour l'accès aux inodes des fichiers de `rep/`
- `/usr/bin/cal` :

Permissions, exemples

- `ls rep/` : besoin de `r` sur `rep/`
- `echo rep/*` : besoin de `r` sur `rep/`
- `rm rep/a` : besoin de `wx` sur `rep/` : modification du répertoire et accès à l'inode pour diminuer le nombre de liens (voir plus loin)
  Aucune permission sur le fichier `rep/a` n'est nécessaire, c'est la permission sur le répertoire qui permet de `rm` !
- `cat rep/a` : besoin de `x` sur `rep/` et `r` sur `rep/` (besoin d'accéder à l'inode pour accéder aux permissions et aux données !)
- `cp rep/a rep/b` : besoin de `wx` sur `rep/`, `r` sur `rep/a` et `w` sur `rep/b`
- `rm rep/*` : besoin de `rwx` sur `rep/`, `r` pour le développement de noms de chemins, `w` pour supprimer les noms contenus dans le répertoire et `x` pour l'accès aux inodes des fichiers de `rep/`
- `/usr/bin/cal` : besoin de `x` sur `/` pour accéder à l'inode correspondant à `/usr`, de `x` sur `/usr` pour accéder à l'inode de `/usr/bin`, de `x` sur `/usr/bin` pour l'inode de `cal` et enfin de `x` sur `/usr/bin/cal` pour l'exécuter

Changer les permissions

Possible si utilisateur propriétaire du fichier (ou root).
On utilise `chmod MODE[, MODE]... FICHIER...` où `MODE` a la forme `[ugoa...][-+=][rwx...]`

- pour qui sont changées les permissions (`user`, `group`, `others`, ou `all` : tout le monde) ;
- `-` pour enlever, `=` pour fixer, `+` pour ajouter.

Exemple

- `chmod og-r secret.txt` : enlever la permission `r` (si elle était présente, sinon ça ne change rien) à `g` et `o` ;
- `chmod a+rwx truc` donner tous les droits, à tout le monde, sur truc ;
- `chmod a=r truc` donner le droit `r` et seulement celui-là à tout le monde (les autres permissions que pouvaient éventuellement avoir des utilisateurs sur `truc` sont ôtées) ;
- `chmod u=rwx,og= truc` : tous les droits pour l'utilisateur propriétaire, aucun droit pour tous les autres utilisateurs.

Liens physiques

- Un lien physique est une association
  nom (dans un répertoire) $\leftrightarrow$ inode.
- La commande `ln` (comme *link*, lier en anglais) permet de donner un nouveau nom dans l'arborescence à un fichier.
- On peut donc avoir des noms différents mais un seul fichier.

Lien physique, exemple

```
$ ls -l foo
total 4
drwxr-xr-x. 1 pierre pierre 8 12 oct.  13:13 bar
-rw-r--r--. 1 pierre pierre 7 12 oct.  13:11 baz.txt
$ cat foo/baz.txt
hop
$ ln foo/baz.txt foo/bar/truc
$ ls -li foo/baz.txt foo/bar/truc
2357623 -rw-r--r--. 2 pierre pierre 7 12 oct. 13:11 foo/bar/truc
2357623 -rw-r--r--. 2 pierre pierre 7 12 oct. 13:11 foo/baz.txt
$ echo plop >>foo/bar/truc
$ cat foo/bar/truc
hop
plop
$ cat foo/baz.txt
hop
plop
$ chmod a+x foo/baz.txt
$ ls -li foo/bar/truc
2357623 -rwxr-xr-x. 2 pierre pierre 4 12 oct.  13:19 foo/bar/truc
```

Suppression d'un nom, suppression d'un inode

- La commande `rm` supprime un lien, c'est-à-dire brise une association nom $\leftrightarrow$ inode dans un répertoire : modification du répertoire, donc besoin de permission `w` dessus ;
- dans l'inode, le nombre de lien diminue de 1 : modification de l'inode donc besoin de permission `x` sur le répertoire ;
- Lorsque le nombre de liens tombe à 0, l'inode est supprimé.
- Le système sait que l'espace anciennement occupé par le fichier est maitenant disponible et peut être réutilisé.
- Il est alors impossible (à part avec de grands moyens et beaucoup de chance et de patience) de retrouver les données contenues dans le fichier.

```
$ ls -li foo/baz.txt
2357623 -rw-r--r--. 2 pierre pierre 7 12 oct. 13:11 foo/baz.txt
$ rm foo/bar/truc
$ ls -li foo/baz.txt
2357623 -rw-r--r--. 1 pierre pierre 7 12 oct. 13:11 foo/baz.txt
$ rm foo/baz.txt # inode supprimé
```

Manipuler un inode

Condition pour le faire : être propriétaire du fichier (ou `root`).

- Modifier les dates : `touch`.
- Modifier l'utilisateur propriétaire : `chown`.
- Modifier le groupe propriétaire : `chgrp`.
- Modifier les permissions `chmod` (voir TP).

Répertoires et inodes

Intermède de shell : construction `case`

Premier exemple

```
#!/bin/sh
# supprimer_tilde

echo "Voulez-vous supprimer tout votre répertoire personnel ?"
read reponse
case $reponse in
[nN]*)
    echo "Ah, je suis soulagé."
    ;;
[oO]* | [yY]*)
    echo "Vous êtes dingue, mais tant pis pour vous."
    sleep 1
    echo "Finalement non, faîtes-le vous-même."
    ;;
*)
    echo "Je n'ai pas compris. Dans le doute je casse tout."
    sleep 1
    echo "Non, je rigolais... Ah ah"
esac
```

Syntaxe

```
case chaine_entre_case_et_in in
motif11 | motif12 | ...)
    commandes1
    ;;
motif21 | motif22 | ...)
    commandes2
    ;;
...
motifn1 | motifn2 | ...)
    commandesn
    [;;]
esac
```

Sémantique

- Dans la construction `case`, la chaîne qui est entre les mots-clés `case` et `in` subit les développements du tilde, de variable, arithmétique, la substitution de commande et enfin la suppression des caractères inhibiteurs.
- Ensuite le shell examine *dans l'ordre* chaque motif. Il lui fait subir les développements du tilde, arithmétique, de variable et la substitution de commande et la suppression des caractères inhibiteurs.
- Dès qu'un motif correspond à la chaîne entre `case` et `in`, le shell exécute les commandes qui sont associées à son cas puis sort de la construction `case`.
- La construction `case` fait le gros du travail dans beaucoup de scripts. À utiliser dès que possible.

Sémantique

- Dans la construction `case`, la chaîne qui est entre les mots-clés `case` et `in` subit les développements du tilde, de variable, arithmétique, la substitution de commande et enfin la suppression des caractères inhibiteurs.
- Ensuite le shell examine *dans l'ordre* chaque motif. Il lui fait subir les développements du tilde, arithmétique, de variable et la substitution de commande et la suppression des caractères inhibiteurs.
- Dès qu'un motif correspond à la chaîne entre `case` et `in`, le shell exécute les commandes qui sont associées à son cas puis sort de la construction `case`.
- La construction `case` fait le gros du travail dans beaucoup de scripts. À utiliser dès que possible.