Initiation à l'environnment Unix
CM4 : fichiers, répertoires et exécution de programmes externes

Pierre Rousselin

Université Paris 13
L1 informatique
octobre 2021

Fichiers sous Unix

Fichiers texte et fichiers binaires

Répertoires et inodes

Contenu d'un fichier

Il n'y a jamais rien d'autre dans un fichier que ce qu'on y écrit.
La commande **od** (comme **octal dump**, déverser en octal) permet de voir les données contenues dans le fichier, sous différentes formes, ici octal et hexadécimal.

```
$ cat comptine.txt
Mon petit lapin
a bien du chagrin.
$ od -An -t x1 -t c comptine.txt
4d  6f  6e  20  70  65  74  69  74  20  6c  61  70  69  6e  0a
 M   o   n       p   e   t   i   t       l   a   p   i   n  \n
61  20  62  69  65  6e  20  64  75  20  63  68  61  67  72  69
 a       b   i   e   n       d   u       c   h   a   g   r   i
6e  2e  0a
 n   .  \n
$ file comptine.txt
comptine.txt: ASCII text
$ wc -c comptine.txt
35 comptine.txt
```

Contenu d'un fichier (2)

Ce qui est contenu dans `comptine.txt` : 35 octets (nombre affiché par `wc -c comptine.txt`).

- Un octet est une suite de 8 bits.
- le bit (*binary digit*) est l'unité d'information fondamentale : un 0 ou un 1, une présence ou une absence, oui ou non, vrai ou faux, ...
- Le contenu d'un octet peut donc être `00000000`, `00000001`, `00000010`, `00000011`, ... `11111110`, `11111111`.
- Il y a 2^8 (soit 256) valeurs possibles dans un octets.
- Lire ou écrire du binaire est pénible pour les humains. Pour cette raison, on a souvent recours à l'écriture hexadécimale (base 16) (et historiquement, parfois à l'octal, c'est-à-dire la base 8, voir la sortie par défaut de `od`).

Les chiffres de l'hexadécimal

- Il nous faut seize chiffres pour écrire en base seize.
- Mais il n'y a que dix chiffres arabes.
- On convient donc d'utiliser les lettres `A`, `B`, `C`, `D`, `E` et `F` en majuscule ou en minuscule (mais ne pas mélanger !) comme symboles supplémentaire.

Hexadécimal, décimal, binaire

$$
\begin{aligned}
0_{16} &= 0_{10} &&= 0000_2 \\
1_{16} &= 1_{10} &&= 0001_2 \\
2_{16} &= 2_{10} &&= 0010_2 \\
3_{16} &= 3_{10} &&= 0011_2 \\
4_{16} &= 4_{10} &&= 0100_2 \\
5_{16} &= 5_{10} &&= 0101_2 \\
6_{16} &= 6_{10} &&= 0110_2 \\
7_{16} &= 7_{10} &&= 0111_2 \\
8_{16} &= 8_{10} &&= 1000_2 \\
9_{16} &= 9_{10} &&= 1001_2 \\
A_{16} &= 10_{10} &&= 1010_2 \\
B_{16} &= 11_{10} &&= 1011_2 \\
C_{16} &= 12_{10} &&= 1100_2 \\
D_{16} &= 13_{10} &&= 1101_2 \\
E_{16} &= 14_{10} &&= 1110_2 \\
F_{16} &= 15_{10} &&= 1111_2
\end{aligned}
$$

Entre le binaire et l'hexadécimal

- Il est évident de convertir un nombre écrit en binaire en un nombre écrit en hexadécimal et inversement.
- Il suffit de transformer chaque séquence de 4 bits en un chiffre hexadécimal et inversement.
- Par exemple l'octet `11010011` peut s'écrire `D3`.
- Et `FFFF` représente les deux octets `1111111111111111`.
- Certaines commandes (ou le shell dans certains contextes) interprètent certaines chaînes de caractères comme des entiers en hexadécimal, si on les fait précéder par `0X` ou `0x` (selon qu'on va utiliser les caractère majuscules ou minuscules).

Retour à `comptine.txt`

- Le premier octet de `comptine.txt` est `4d` (en hexadécimal), c'est-à-dire `01001101`.
- Ce `4d` n'a aucune signification particulière pour le système.
- Mais certains programmes pourront l'interpréter comme ils le souhaitent.
- Ici, c'est *l'émulateur de terminal* qui, dans cette configuration, est fait pour interpréter ce `4d` comme : « Afficher un `M` sur l'écran. »
- Cette interprétation est suffisamment répandue pour faire l'objet d'un standard : ASCII (*american standard code for information interchange*).
- Il s'agit de faire correspondre à certains octets (seulement ceux commençant par un 0) un caractère (symbole ou autres choses utile pour écrire du texte).
- `man ascii` donne la table avec codage en hexadécimal, en décimal, en octal et pour certains caractères, les séquences d'échappement en C (ou pour la commande `printf`).

Fichiers sous Unix

Fichiers texte et fichiers binaires

Répertoires et inodes

Petite parenthèse sur l'encodage des caractères

- ASCII reste omniprésent mais est bien trop limité.
- Il y a encore assez peu de temps, chaque région du monde avait son propre encodage de caractère au-dessus de ASCII : on utilise le premier bit qu'on met à 1 et ça permet d'encoder 128 caractères supplémentaires.
- En France, iso-8859-1 (ou latin 1) qui permet d'avoir les caractères accentués minuscules, mais pas œ, Œ, ou les majuscules accentuées (É, À, etc). Voir `man latin1`.
- Heureusement Unicode et utf8, l'une de ses implémentations les plus courantes sont arrivés : un seul jeu de caractère pour tous les symboles du monde. Voir `man utf8` et `man unicode`.
- En plus la compatibilité avec ASCII est préservée dans utf8.
- Inconvénient d'utf8 : les caractères sont codés sur un nombre variable d'octets (les avantages l'emportent très très largement).

Changer l'encodage : `iconv`

- La commande standard `iconv` permet de changer l'encodage d'un fichier.
- Exemple :

  ```
  $ iconv -f latin1 -t utf8 <texte_latin1.txt >texte_utf8.txt
  ```
- Voir le TP4.
- En espérant que votre génération galère moins que la précédente avec les encodages pourris (c'est bien parti).

On s'amuse avec l'utf8

- Extension non standard (mais répandue) de `printf` : séquence d'échappement `\xNN` pour écrire directement l'octet écrit en hexadécimal `NN`.

```
$ printf 'É' | od -An -t x1
 c3 89
$ printf '€' | od -An -t x1
 e2 82 ac
$ printf '\xe2\x82\xac'
€$ printf '\xe2\xb4\xb0' # tifinagh
```

- Ctrl-Maj-u (sous Linux) puis unicode, par exemple Ctrl-Maj-u puis 1D11E pour une clé de sol.

Fichiers texte

- On referme cette parenthèse sur l'encodage des caractères.
- Le système Unix favorise depuis le début les *fichiers textes.*
- `comptine.txt` est un exemple de tel fichier.

Définition

Un fichier texte est un fichier destiné à être interprété comme une suite de caractères dans un certain jeu de caractères codés (utf8, ascii, latin1, ...) de façon à être lisible et facilement éditable par un humain.

Fichiers texte

Exemples :

- Un fichier texte sans forme particulière, comme `comptine.txt`.
- Un fichier source C, java, OCaml, FORTRAN, ... destiné à être compilé donc à être compris également par un programme appelé compilateur.
- Un script shell, python, perl, javascript, ... destiné à être interprété donc à être compris également par un programme appelé interpréteur.
- Un document HTML destiné à être compris par un navigateur web pour produire un affichage dans sa fenêtre.
- Un document tex destiné à être compris par l'interprète `pdftex` pour produire un document `pdf`.
- Les fichiers de configuration sous Unix, pouvant avoir chacun leur format, par exemple `~/.ssh/config` pour la configuration du client `ssh` de l'utilisateur, les fichiers de configuration du système dans `/etc` (pour *editable text configuration*).

Fichiers dits « binaire »

Définition

Un fichier qui n'est pas un fichier texte est un *fichier binaire.*

Cette appellation est assez malheureuse (les fichiers textes sont aussi codés en binaire...) mais c'est celle qui est répandue. Un humain ne peut pas comprendre son contenu avec un simple éditeur de texte. Ce fichier n'aura de sens que pour le ou les programmes à qui il est destiné.

Exemples :

- Ce que produit un compilateur (C, java, OCaml, etc).
- Une image (jpg, png, etc), une vidéo, des sons...
- Une archive compressée.
- Un fichier utilisé par un logiciel de traitement de texte (comme LibreOffice ou Microsoft Word).

Fichiers binaires exécutables

- Suivant le contexte, un *binaire* peut aussi vouloir dire plus spécifiquement un programme compilé, par opposition au code source du programme qui lui est un fichier texte.
- Raison pour laquelle la plupart des programmes du système Unix sont traditionnellement dans `/bin` et/ou `/usr/bin` (bien qu'ils contiennent aussi des scripts qui sont des fichiers texte !)
- L'existence de binaires (exécutables) dont le code source n'est pas public pose problème :
  - le binaire n'est pas lisible par un humain ;
  - dès lors, comment savoir ce que fait réellement le programme ? Est-il dangereux ?

Le logiciel libre (*free software*)

- Lancé par Richard Stallman : GNU (1983) puis Free Software Foundation (1985).
- Outils logiciels mais aussi juridiques (licence GPL).

Les 4 libertés fondamentales que doit respecter un logiciel libre :

- la liberté de faire fonctionner le programme comme vous voulez, pour n'importe quel usage (liberté 0) ;
- la liberté d'étudier le fonctionnement du programme, et de le modifier pour qu'il effectue vos tâches informatiques comme vous le souhaitez (liberté 1) ; l'accès au code source est une condition nécessaire ;
- la liberté de redistribuer des copies, donc d'aider les autres (liberté 2) ;
- la liberté de distribuer aux autres des copies de vos versions modifiées (liberté 3) ; en faisant cela, vous donnez à toute la communauté une possibilité de profiter de vos changements ; l'accès au code source est une condition nécessaire.

Le logiciel libre

Quelques exemples :

- noyau Linux (licence GPLv2), implémentation libre d'un noyau Unix (entre 25 et 30 millions de lignes de code, plusieurs milliers de contributeurs chaque année) ;
- FreeBSD, implémentation libre d'Unix (noyau et utilitaires), 16 millions de ligne de code ;
- Projet GNU, par exemple ls, sed, bash, emacs, gimp, gcc, ... ;
- Firefox (navigateur), Thunderbird (client mail) chez Mozilla ;
- La suite LibreOffice de la Document Foundation ;
- Chromium (navigateur) et le système Android chez Google ;
- Visual Studio Code chez Microsoft.

Quelques contre-exemples : MS Windows, MacOS, MS Office, Google Chrome, Chrome OS, beaucoup de pilotes de périphériques.

Fichiers sous Unix

Fichiers texte et fichiers binaires

Répertoires et inodes

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.

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.

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--. 2 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/tru
```

Suppression d'un nom, suppression d'un inode

- La commande `rm` supprime un lien.
- 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).