Initiation à l'environnement Unix
CM3 : utilisateurs et introduction aux fichiers

Pierre Rousselin
rousselin@univ-paris13.fr

Université Sorbonne Paris Nord
L1 informatique et double-licence
septembre 2024

Utilisateurs et groupes

Fichiers sous Unix, introduction

Contenu d'un fichier normal

Utilisateur normal et super-utilisateur

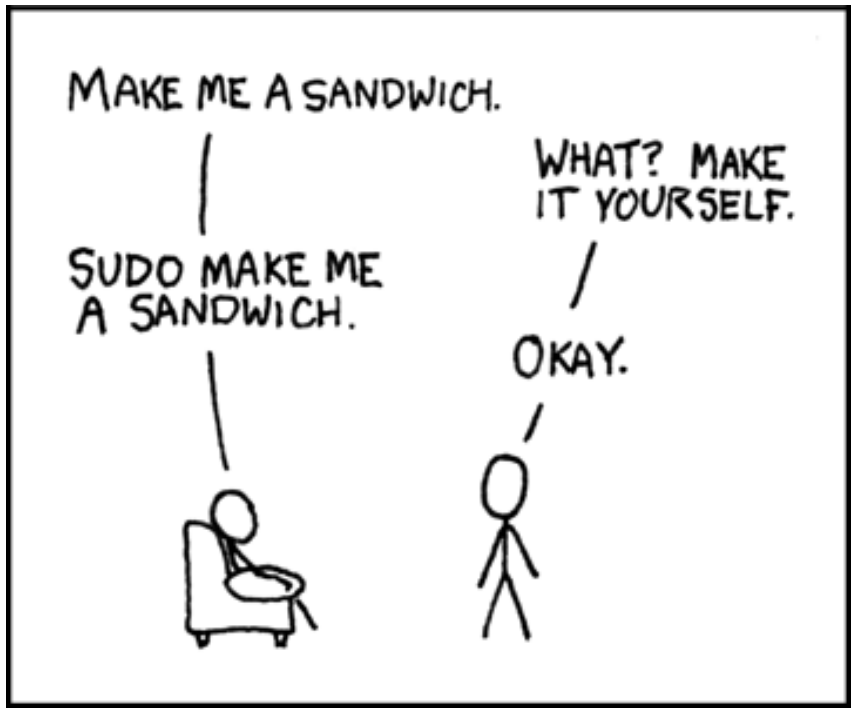
Il y a deux types d'utilisateurs :

- **Un** super-utilisateur, souvent appelé `root` qui peut tout faire sur le système : modifier, supprimer, lire tout ce qu'il veut, lancer ou tuer n'importe quel processus, ...
- Les autres utilisateurs, qui peuvent faire ce que leur permettent leurs permissions sur chaque fichier du système : souvent ils peuvent faire à peu près ce qu'ils veulent dans leur répertoire personnel et pas grand chose en-dehors.
- Certains programmes nécessitent d'être lancés par certains utilisateurs : en particulier, ceux qui modifient le système doivent être lancés par root. Par exemple :
  - ajout d'un utilisateur (`useradd`)
  - installation d'un programme (`make install`) ou utilisation d'un gestionnaire de paquets (`apt` pour Debian, `dnf` pour Fedora, *etc*).

Devenir un autre utilisateur

- Commande `su <user>` pour `switch user`, par exemple `su alice` si `alice` fait partie des utilisateurs. Bien sûr, on vous demande son mot de passe.
- Pour devenir `root` : `su root` ou `su` (sans argument) : parfois impossible, même sur votre propre machine, car le compte `root` n'a pas de mot de passe dans certaines distributions.

La commande `sudo`



Pour ceux qui peuvent être `root`, sur leur machine, ou qui en administrent d'autres, la commande `sudo` (pour *superuser do*) permet de faire quelque chose *en tant que super-utilisateur*, si vous faites partie des utilisateurs qui peuvent le faire (*sudoers*, fichier `/etc/sudoers`).

La commande sudo

```
$ useradd alice # créer un nouvel utilisateur appelé alice
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
$ sudo useradd alice
[sudo] Mot de passe de pierre :
$ sudo userdel alice # supprimer le compte d'alice
$ sudo su # devenir root
# id -un # mon identifiant
root
# echo $HOME
# /root
```

La commande `sudo`

- Évidemment, vous ne pouvez pas tester `sudo` en salle de TP (d'ailleurs moi non plus et c'est très bien comme ça !)
- En pratique, sur votre machine, pour mettre à jour ou installer des paquets. Par exemple sur Debian et dérivés (Ubuntu, Mint, ...)

```
$ sudo apt update # mettre à jour la base de données
$ sudo apt upgrade # mettre à jour les paquets
$ sudo apt install coqide # installer le paquet coqide
```

Être ou ne pas être `root`

- « De grands pouvoirs impliquent de grandes responsabilités. »
- `Permission denied:` la tentation est grande d'ajouter `sudo` en début de commande...
- mais si un utilisateur normal n'a pas le droit de le faire, il y a sans doute une raison !
- Les réseaux importants sont gérés par des *administrateurs systèmes* qui sont les seuls à avoir accès au compte `root` : voir le cours de L2 informatique *Administration système*.

Utilisateurs : être

- Chaque utilisateur a un nom (`id -nu`) et un identifiant numérique : l'`uid` (pour *user identity*, `id -u`).
- La liste de tous les utilisateurs est accessible dans le fichier `/etc/passwd`.
- Il y a des utilisateurs qui ne correspondent à aucun être humain : pour des raisons de sécurité, certains programmes sont lancés en faisant « comme si » ils étaient lancés par un certain utilisateur (évite qu'ils soient lancés par `root` et aient tous les droits).

Utilisateurs : avoir

Un utilisateur a :

- un `SHELL` par défaut, souvent `bash` sous Linux ou `zsh` sous MacOS mais peut être configuré (commande `chsh`) ;
- un `HOME`, répertoire personnel, aussi noté `~` dans le shell ;
- ses propres fichiers de configuration (`.bashrc`, `.zshrc`, `.vimrc`, ...) souvent dans son répertoire personnel ;
- un mot de passe, des clés publiques et privées (répertoire `.ssh`), etc.

Utilisateurs et groupes

En outre, chaque utilisateur a

- un groupe courant (`id -g` pour le numéro (*gid*) et `id -ng` pour le nom) ;
- éventuellement d'autres groupes auquel il appartient (`id -G` pour les numéros et `id -Gn` pour les noms).

Utilisateurs et groupes

- C'est `root` qui a la charge de donner au moins un groupe à chaque utilisateur :
  - `groupadd` et `groupdel` pour créer et supprimer un groupe
  - `usermod` pour modifier les attributs d'un utilisateur, en particulier lui ajouter ou supprimer des groupes.
- La commande `newgrp <groupe>` ouvre un nouveau shell avec `<groupe>` comme groupe courant.

```
$ id -Gn
pierre wheel dialout
$ id -gn
pierre
$ newgrp wheel
$ id -Gn
wheel dialout pierre
$ id -gn
wheel
```

Utilisateurs et groupes

Fichiers sous Unix, introduction

Contenu d'un fichier normal

Tout est fichier

> *Everything in the* Unix *system is a file. This is less an oversimplification than you might think. It is one of the best examples of the "keep it simple" philosophy, showing the power achieved by careful implementation of a few well-chosen ideas.*

Kernighan et Pike, *The Unix Programming Environment (1984)*.

Tout est fichier

En effet, tout ce dans quoi on peut *lire* (appel système `read`) et/ou *écrire* (appel système `write`) des données est un fichier : ceci inclut

- les fichiers contenus sur un périphérique de stockage (disque dur…) qui peuvent contenir des données (dans tel ou tel format) ou des programmes qui seront exécutés par le noyau ;
- les terminaux, physiques (`/dev/tty`) ou virtuels (`/dev/pts`) ;
- les périphériques d'entrée (clavier, souris, …)
- les périphériques de sortie (écran, enceintes, …)
- les périphériques de stockage (disques durs, …)

Types de fichiers

Il y a 7 types de fichiers dans l'arborescence, et c'est tout !

- `-` pour les fichiers normaux (*regular files*) ;
- `d` pour *directory*, répertoire ;
- `l` pour *link*, les liens symboliques (des « raccourcis » sous Windows) ;
- `b` pour *block device* et `c` pour *character device* : périphériques de type bloc ou caractère (pas ou peu abordés dans ce cours) ;
- `p` pour *pipe* ou tube nommé (assez rare) ;
- `s` pour *socket* (en français « prise », comme une prise de courant) permettant les échanges de données entre processus sur le même système ou via le réseau (internet par exemple).

Premier caractère de la sortie de `ls -l`.

Types de fichiers

```
$ ls -ld /etc/passwd /run/ /bin /dev/sda /dev/input/mouse0
lrwxrwxrwx.  1 root root       7 26 janv.  2021 /bin -> usr/bin
crw-rw----.  1 root input 13, 32  5 oct.  07:28 /dev/input/mouse
brw-rw----.  1 root disk   8,  0  5 oct.  07:28 /dev/sda
-rw-r--r--.  1 root root    2690  5 oct.  10:58 /etc/passwd
drwxr-xr-x. 51 root root    1420  5 oct.  07:49 /run/
$ ls -l /run/cups/cups.sock
srw-rw-rw-. 1 root root 0  5 oct.  07:28 /run/cups/cups.sock
$ ls -l /run/initctl
prw-------. 1 root root 0  5 oct.  07:28 /run/initctl
```

Fichiers normaux et données

- Uniformité des fichiers normaux : le système ne traite pas différemment un fichier source C ou un fichier PDF ou un tableau libreoffice.
- Tous sont des fichiers qui ne sont que des suites d'octets : ce sont les processus qui sont chargés de les interpréter et leur donner un sens.
- Il n'y a donc pas différents types de fichiers normaux.
- Conventions pour le nommage : une extension, pour aider à s'y retrouver, par exemple `.c` pour du code en C, `.pdf` pour un document PDF (*portable document file*), etc.
- Mais ce n'est qu'une convention entre humains. Le système s'en fiche (toutefois certains utilitaires s'en servent pour « deviner » à quoi est destiné ce fichier).

La commande `file`

La commande `file` essaie de deviner à quoi sert un fichier en se servant de son contenu.

```
$ file unix_cm3.tex
unix_cm3.tex: LaTeX document, UTF-8 Unicode text
$ file sandwich.png
sandwich.png: PNG image data, 360 x 299, 8-bit grayscale, non-in
$ file unix_cm3.pdf
unix_cm3.pdf: PDF document, version 1.5
```

La commande `file`

La commande `file` essaie de deviner à quoi sert un fichier en se servant de son contenu.

```
$ file unix_cm3.tex
unix_cm3.tex: LaTeX document, UTF-8 Unicode text
$ file sandwich.png
sandwich.png: PNG image data, 360 x 299, 8-bit grayscale, non-in
$ file unix_cm3.pdf
unix_cm3.pdf: PDF document, version 1.5
```

Mais peut toujours se tromper.

```
$ cat a.txt
Pour coder en C, on doit souvent taper
#include <stdio.h>
$ file a.txt
a.txt: C source, ASCII text
```

Utilisateurs et groupes

Fichiers sous Unix, introduction

Contenu d'un fichier normal

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 caractères 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 utiles 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`).

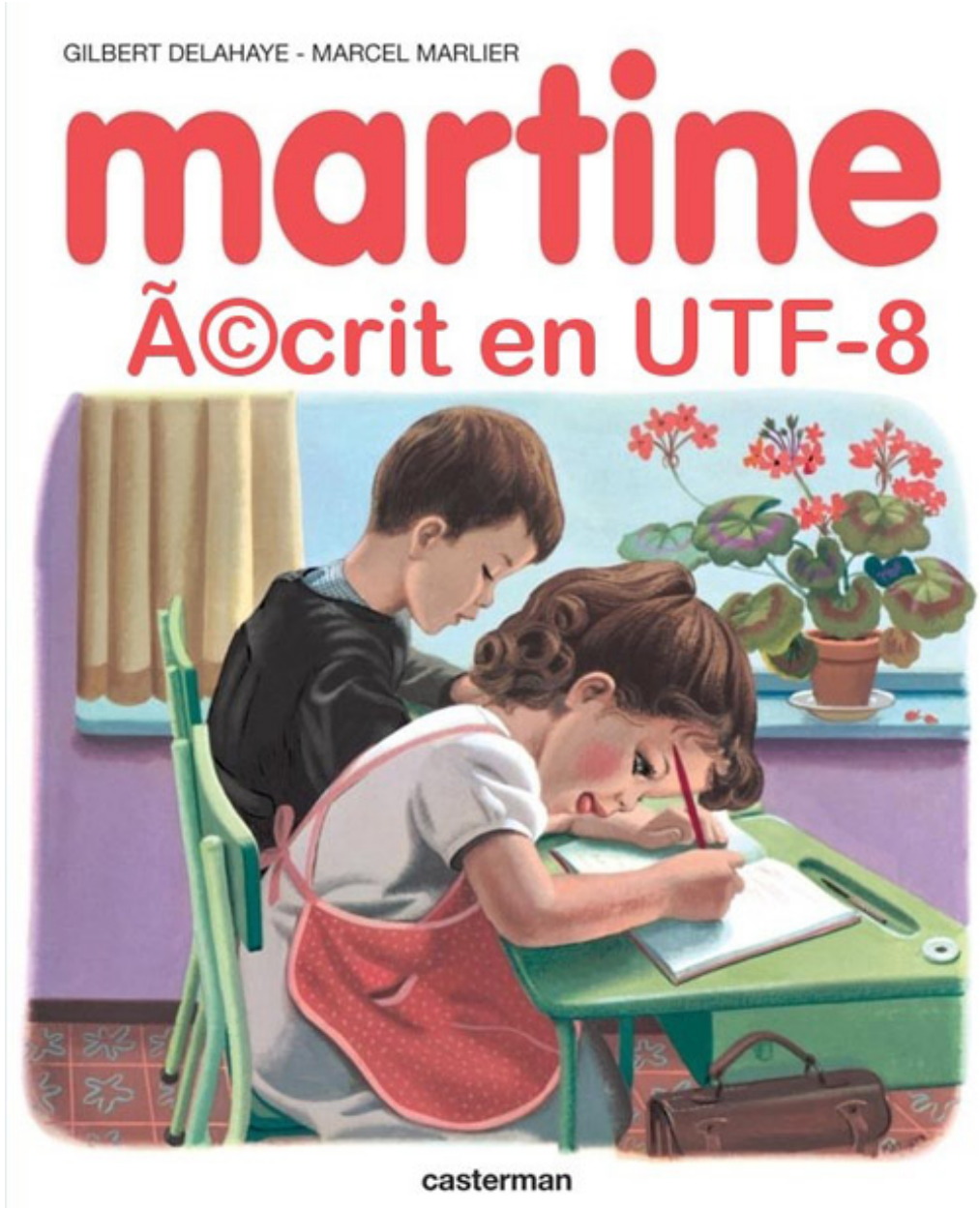
Petite parenthèse sur l'encodage des caractères

- 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 UTF-8 (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`.
- *Universal character set Transformation Format - 8 bits*
- En plus la compatibilité avec ASCII est préservée dans UTF-8.
- Inconvénient d'UTF-8 : 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).


GILBERT DELAHAYE - MARCEL MARLIER
martine
Ã©crit en UTF-8
casterman

Martine écrit en UTF-8

- En UTF-8, le caractère `'é'` est codé par les deux octets `c3` puis `a9`

```
$ printf 'é' | od -t x1 -An
 c3 a9
```

- Remarque : c'est forcément plus d'un octet car le caractère `'é'` ne fait pas partie d'ASCII (et nous sommes en UTF-8).
- Si on *interprète* ces deux octets comme du latin-1 (iso-8859-1), on voit que (`man iso-8859-1`)
  - `c3` correspond à `Ã`
  - `a9` correspond à ©

On s'amuse avec l'UTF-8

- 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\n'
€
$ printf '\xe2\xb4\xb0\n' # tifinagh
```

- ctrl + ⇧ + U (sous Linux) puis unicode, par exemple ctrl + ⇧ + U puis 1D11E pour une clé de sol.