Les fichiers & les dossiers

1 - Système de fichiers

Un système de fichier (FS ou File System) est l'organisation du stockage des données, généralement nommées des fichiers, par un OS.

Cette organisation utilise une logique de classement, souvent hiérarchique, pour un accès rapide et simple aux données par l'OS.

L'OS met à disposition de l'utilisateur et des applications, des outils permettant d'accéder aux données du FS sans forcément avoir besoin de connaître l'organisation sous-jacente du FS.

Dans le cas des principaux FS sous Unix, cette organisation est hiérarchique.

Quelques noms de FS sous Linux : UFS, ext2, ext3, ReiserFS. Clés USB : généralement FAT16 ou FAT32.

2 - Qu’est-ce qu’un fichier ?

(CM #1) - Un fichier est :

- Une zone allouée généralement sur un espace de stockage (disque dur, clé USB, Cloud).
- Identifié par un nom.
- Identifié par un ID unique (ls -i)
- Contient des données quelconques (image, texte, son, vidéo).

Un nom et un ID uniques ?

Oui et non.

On va voir dans quelles conditions ils doivent être uniques et dans lesquelles ils peuvent ne pas l'être.

Unité de mesure

En informatique, l'unité de mesure est l'octet.

L'atome informatique est le bit, binary digit, nombre binaire, 0 ou 1, courant/pas de courant.

Racine oct- pour 8. Octet = groupe de 8 bits.

Chaque bit étant binaire, peut prendre seulement 2 valeurs : 0 ou 1.

Un octet (8 bits) peut donc représenter :

(2 valeurs par bit)$^{8 \text{ bits}}$ = 2^8 valeurs = 256 valeurs.

Arithmétique décimale vs. binaire

L'informatique est binaire, son arithmétique aussi.

Pour un être humain (barbu ou pas), le binaire demande plus d'effort que le décimal, c'est moins naturel ou plutôt moins conventionnel.

En décimal (base 10) :

- Kilo : 10^3 = 1 millier, kilowatt (1.000 W)
- Méga : 10^6 = 1 million, mégawatt (1.000 KW)
- Giga : 10^9 = 1 milliard, gigawatt (1.000 MW)

L'Académie Royale des Sciences (France), définit en 1795 que Kilo = 1000 (Kilogramme, Kilomètre).

Le SI (Système international d'unités), a adhéré à cette définition. Mais, problème...

En binaire (base 2), avant 1998 :

- Kilooctet : 2^{10} = 1024 (~1.000)
- Mégaoctet : 2^{20} = 1.048.576 (~1 million)
- Gigaoctet : 2^{30} = 1.073.741.824 (~1 milliard)

Ce nommage était en décalage avec les préfixes de nommage du SI.

Depuis 1998 :

- Kilo (Ko), Méga (Mo), Giga (Go) etc. sont maintenant des puissances de 10.
- Nouvelle norme, nouveaux préfixes : Kibi (Kio), Mébi (Mio), Gibi (Gio) etc. bi pour binaire.
- Kilooctet (Ko) pré 98 = Kibioctet (Kio) post 98, etc.
- Nouveaux Ko, Mo, Go reprennent le système décimal (mille, million, milliard d'octets).

Différences relativement minimes :

- 1 Ko (1.000 o) vs. 1 Kio (1.024) : 2,3%
- 1 Mo (1.000 Ko) vs. 1 Mio (1.024 Kio) : 4,6%
- 1 Go (1.000 Mo) vs. 1 Gio (1.024 Mio) : 6,9%
- 1 To (1.000 Go) vs. 1 Tio (1.024 Gio) : 9,2%

Vous pouvez continuer à parler en Kilo, Méga, Giga, Tera. Ca reste une échelle de grandeur. Il faut juste connaître cette subtilité. Les fabricants l'ont bien comprise ! :-)

Equipements de stockage

Par ordre de taille :

- CD-Rom, DVD :
  - Stockage en lecture seule, externe et local.
  - Données statiques (bases documentaires, films).
  - Lent.
  - Assez peu fragile (sauf rayures).
  - Généralement quelques Go.
  - Se fait de plus en plus rare, au profit du Cloud.

Par ordre de taille (suite) :

- Clé USB :
  - Stockage d'appoint (R/W), externe et local.
  - Usage : Sauvegarde, partage/transfert de fichiers.
  - Fiabilité moyenne-faible.
  - Assez lent, peu fragile.
  - Généralement quelques dizaines de Go.

Par ordre de taille (suite) :

- Disque dur :
  - Stockage local (R/W), dans l'ordinateur ou connecté en USB par exemple.
  - Fiabilité moyenne.
  - Rapide.
  - Généralement plusieurs centaines de Go ou quelques To.

Par ordre de taille (suite) :

- NAS (Network Attached Storage) et SAN (Storage Area Network) :
  - Stockage externe (R/W).
  - Accès par le réseau local généralement.
  - Rapide mais dépendant de l'infrastructure réseau.
  - Plusieurs dizaines (voire centaines) de To.

Par ordre de taille (suite) :

- Le Cloud :
  - Stockage distant (R/W).
  - Accès par le réseau Internet.
  - Rapidité aléatoire, dépendant de la bande passante de l'accès Internet.
  - Aucune limite autre que le portefeuille !

Pour avoir le tournis, le Cloud est estimé à :

- 2016 : 286 Eo (Exaoctets, 10^{18}, 1 milliard de milliard d'octets)
- 2021 : 1,3 Zo (Zetaoctets, 10^{21}, 1.000 milliards de milliard d'octets)
  X 5 en 5 ans.
  ~120 milliards de films Full HD.

(Source : www.lebigdata.fr)

Règle d'Or

Peu importe l'équipement de stockage, une sauvegarde n'est jamais inutile ! Défaillance, erreurs de manipulation, malveillance... Celui qui n'a jamais perdu un fichier n'a sans doute pas fait grand chose en informatique.

Règle d'Or

On sauvegarde surtout les données qui ne peuvent pas être reconstruites facilement. Pour un particulier : photos, vidéo perso... Pour une société : données d'entreprise, données clients...

Découpage espace de stockage

Les données ne sont pas jetées en vrac dans un espace de stockage. Ce n'est pas une chambre d'étudiant, l'OS doit pouvoir retrouver les choses très rapidement.

Un espace de stockage est découpé en petits morceaux, des blocs. Un bloc fait généralement quelques Kio. Exemple : 4 Kio (4096 octets).

Quand on stocke un fichier de 10 Ko, il va occuper 3 blocs, le 3ème étant évidemment plus large que nécessaire, on perd alors un peu de place.

Les blocs ainsi alloués par l'OS pour un fichier ne sont pas obligatoirement consécutifs.

Comment l'OS retrouve-t-il les données ?

Pour s'y retrouver, l'OS stocke :

- Les données utiles du fichier.
- Des données de localisation sur l'espace de stockage (métadonnées), pour qu'il puisse retrouver et accéder aux données du fichier quand il/on en aura besoin.

Plus les blocs sont gros, moins on a besoin de stocker de métadonnées mais plus on perd d'espace aussi. En moyenne on perd 50% de la taille d'un bloc X nb de fichiers.

4 Ko est une taille estimée comme un bon compromis.

Un disque de 1 To fait ainsi 1024 x 1024 x 1024 x 1024 / 4096 = 268 Millions de blocs. Y stocker 10 millions de fichiers fait perdre 20 Go d'espace (2% du disque).

3 - Qu’est-ce qu’un dossier ?

Un dossier (folder), aussi appelé répertoire (directory), est un moyen de découper l'espace de stockage en sous-espaces.

C’est un conteneur. Il peut contenir des fichiers ou encore d'autres dossiers qui eux mêmes sont des conteneurs etc.

Physiquement on a la même chose avec des dossiers papier :

- Une armoire avec des dossiers qui contiennent des chemises cartonnées qui contiennent des sous chemises qui contiennent des feuilles.
- Armoire, dossiers, chemises, sous-chemises sont des conteneurs, des dossiers.
- Les feuilles de papier sont des fichiers, ce qui contient les écrits, les données.

Arborescence

On parle d'arborescence pour évoquer l'ensemble des objets et les relations qui les lient :

- Un dossier est le tronc ou une branche.
- Un fichier est une feuille.

On ne peut pas aller plus loin que le fichier : le fichier est un objet final de l'arborescence.

Un fichier appartient obligatoirement à un dossier et à un seul, comme une feuille est sur sa branche et pas sur une autre branche.

L'arborescence est étudiée en détail un peu plus loin.

Répertoire

L'appellation répertoire est une analogie aux répertoires qui permettent de lister des objets ou des individus, suivant un ordre spécifique (alphabétique, chronologique, numérique) :

- Répertoire téléphonique.
- Répertoire théâtral.
- Répertoire d'œuvres d'art.
- Et donc : le Répertoire sur un espace de stockage.

Un fichier "spécial"

Ainsi, sous Unix, un dossier est un fichier spécial dont les données ne sont pas une image ou un texte, mais simplement une liste d'autres objets (fichiers ou dossiers) et leur localisation sur l'espace de stockage.

Cependant, seul l'OS sait lire et exploiter les données de ce fichier spécial, et il le fait pour nous : on peut faire un ls -l sur un fichier mais aussi sur un dossier.

Nommage

Puisqu'un dossier est un fichier (spécial), il peut être nommé comme un simple fichier standard :

- nom de base
- .extension

Avec l'un, l'autre ou les 2. Absolument aucune différence avec les fichiers.

Contenu et limitations

Il n'y a théoriquement pas de limite au nombre de fichiers/dossiers pouvant être mis dans un dossier.

Cependant, en pratique, il est déconseillé de mettre des dizaines de milliers de fichiers ou plus. Ca peut devenir lourd et lent à traiter pour l'OS. Si besoin, on peut répartir de trop nombreux fichiers en les répartissant dans des sous-dossiers.

Le nombre d'imbrications (dossier dans un dossier dans un dossier etc.) est aussi quasiment illimité en théorie.

En pratique, dépasser une quinzaine d'imbrications va sans doute devenir compliqué à gérer pour l'utilisateur.

4 - Retour sur le nommage

Unicité de nom

Dans un dossier il ne peut y avoir qu'un et un seul objet tous types confondus (dossier ou fichier) avec un nom donné (nom de base + extension).

Le nom de base peut être le même si l'extension est différente : image.png et image.jpg.

Il peut y avoir 2 objets du même nom s'il n'ont pas le même dossier parent, s'ils ne sont pas sur la même branche immédiate de l'arbre.

Il peut y avoir 2 rues Victor Hugo en France, mais pas dans la ville de Lannion. Pour les objets du système de fichiers Unix, c'est pareil.

Unicité d'ID

Un objet a donc un nom unique dans son conteneur, dans son dossier parent.

Un objet a aussi un ID unique. Mais pas unique seulement dans son dossier parent. Cet ID est unique dans l'espace de stockage tout entier.

ls -i affiche cet ID appellé un inode (nœud d'index).

Objets cachés

Il est possible de cacher des objets, invisibles quand on liste avec la commande ls standard et dans certains logiciels.

Cacher est une commodité, mais rien de très secret.

La commande ls -a (ou --all) permet de montrer ce qui est normalement caché.

Un objet est caché si son nom de base commence par un . (point).

Dans le home, le répertoire dans lequel l'utilisateur se trouve après le login, il y a déjà plusieurs objets cachés (principalement des dossiers).

Ils sont utilisés par certains logiciels (comme le bash) pour y stocker leurs paramètres de fonctionnement.

Généralement ce sont des fichiers textes lisibles avec un cat par exemple et modifiables avec un simple éditeur de texte (vi, nano, geany).

Deux dossiers spéciaux . et .. sont aussi présents. On verra leur signification plus loin.

5 - Manipulation des dossiers

mkdir - Crée un dossier

Origine du nom : make directory.
Syntaxe : mkdir [option] dossier [dossier...]

Quelques exemples :

- mkdir img
- mkdir png jpg mp3

rmdir - Supprime un dossier

Origine du nom : remove directory.
Syntaxe : rmdir [option] dossier [dossier...]

Quelques exemples :

- rmdir old
- rmdir bad old backup

Attention, un dossier doit être totalement vide (ni fichier, ni dossier) pour que rmdir fonctionne.

S'il n'est pas vide, rien n'est supprimé.

Parfois il semble vide mais ne l'est pas.
Penser aux objets cachés (faire un ls -a pour vérifier).

6 - Arborescence

L'arborescence est une représentation des relations parent-enfants qui existent entre les objets du système de fichiers.

Je suis ton père

On a parlé des dossiers comme étant des conteneurs d'autres fichiers/dossiers.

Donc un dossier est un conteneur mais il est aussi un contenu, il a un dossier parent qui lui même a un dossier parent etc.

Mais jusqu'où ? Est-ce infini ?

Comme pour toute chose, il y a une origine. On pourrait dire que c'est le tronc de l'arborescence.

Sous Linux cette origine s'appelle la racine (root), on l'écrit “/” (slash ou barre oblique). “/” est un dossier comme les autres, sa seule originalité est qu'il n'a pas de parent.

Pour les puristes et les curieux, il a bien un parent : c’est lui même. Il est le seul à pouvoir dire je suis mon père !

L'origine de l'arborescence et l'administrateur (le "Dieu" sous Unix) s'appellent tous les deux root et c'est sans doute lié.

On le verra plus tard en parlant des droits sur les objets.

Les objets ont une seule relation : leur dossier parent.

Il ne peut donc pas y avoir d'incohérence dans l'arborescence car la notion de grand-parent d'un objet n'est pas une relation établie, elle n'existe que parce que le parent d'un objet a aussi un parent, qui est donc, par transitivité, le grand-parent de l'objet.

Il suffit de déplacer un objet ailleurs pour lui changer instantanément tout son arbre généalogique !

Tout objet du FS a obligatoirement un dossier parent. Il ne peut pas être isolé quelque part.

De la même manière, 2 objets sont frères simplement parce qu'ils ont le même parent. Il n'y a rien d'autre dans le FS pour matérialiser cette relation de frère.

Home directory

- Chaque utilisateur de l'OS possède un home.
- C'est un dossier de travail personnel.
- Il contient souvent des dossiers cachés.
- C'est le dossier dans lequel il est après s'être loggué.
- Il porte généralement le même nom que son login.
- Tous les home sont en principe dans un même dossier parent (par convention : /home ou /users).
- Dans les commandes, on fait référence à lui par ~.

Répertoire de travail

Il n'y a pas un répertoire de travail (Working directory) précis dans l'arborescence. Le répertoire de travail est simplement le répertoire dans lequel l'utilisateur se trouve à un instant T.

Peu importe ce qu'il fait, un utilisateur est toujours dans un répertoire courant, son répertoire de travail du moment.

Moi et mon père

On a évoqué précédemment la présence systématique de 2 dossiers spéciaux : . et .., présents dans n'importe quel dossier.

- . représente le répertoire courant, le répertoire de travail.
- .. représente le répertoire parent.

Quand on est dans un dossier, on ne connaît pas forcément le nom de ce dossier et pas plus celui du parent.

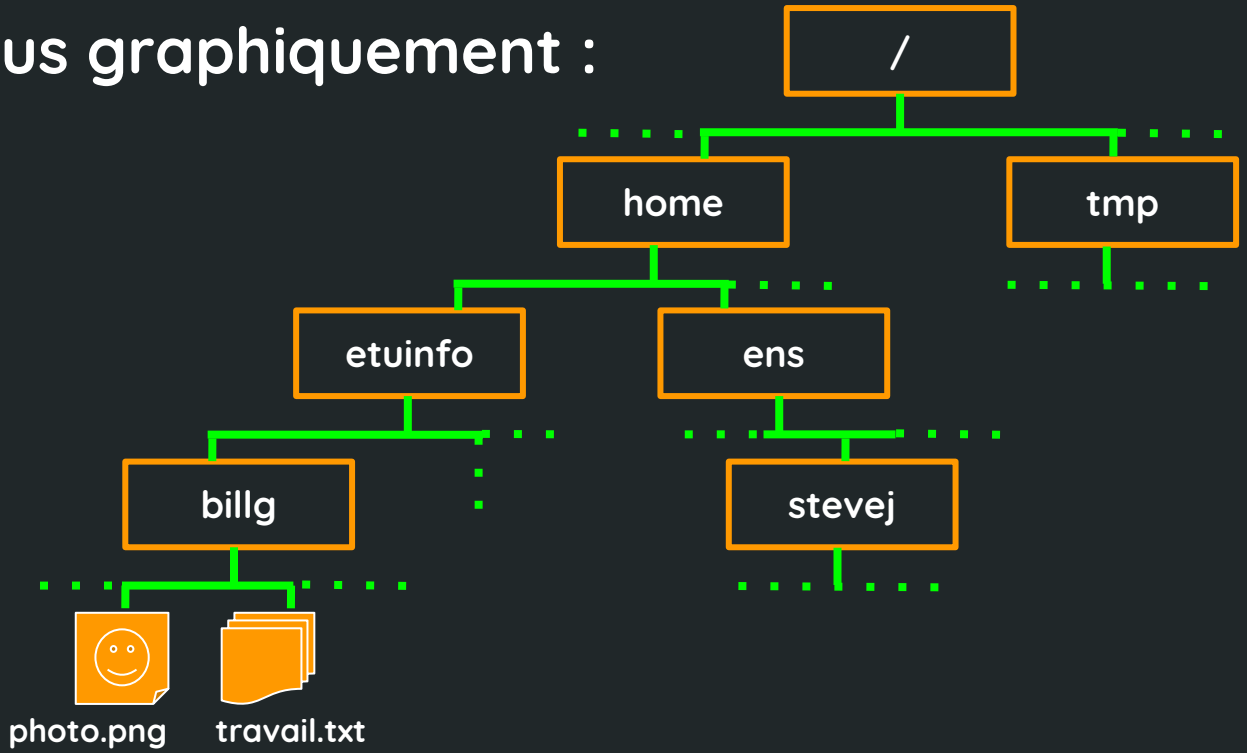
Il est ainsi facile d'y faire référence sans utiliser leur nom en utilisant "." pour dire "ici" (dans le dossier courant), et ".." pour dire "chez mon père" (dans le dossier parent).

Représentation d'arborescence

Exemple (extrait) :

```
/ (root)
+-- home/
|   +-- etuinfo/
|   |   +-- billg/
|   |       +-- photo.png
|   |       +-- travail.txt
|   +-- ens/
|       +-- stevej/
+-- tmp/
```

Plus graphiquement :



7 - Manipulation d'arborescence

rm - Supprime aussi un dossier

On peut aussi utiliser rm avec la syntaxe :
rm -r [option] objet [objet...]

rm -r dossier permet ainsi de supprimer un dossier et tous les objets contenus dans ce dossier (-r pour récursif), y compris les objets dans les dossiers dans les dossiers... Ca revient à scier une branche complète !

C'est une option très dangereuse.

Il faut réfléchir et vérifier plusieurs fois sur quelle branche on est assis avant de la scier avec l'option -r.

Si on coupe la mauvaise branche, on peut perdre gros !

cd - Change de répertoire courant

Origine du nom : change directory.
Syntaxe : cd dossier

Quelques exemples :

- cd img
- cd .. : remonte dans le dossier parent.
- cd : retourne dans le home.
- cd - : retourne dans le précédent répertoire courant.

8 - Chemins

Quand on fait un ls *.c par exemple, on travaille dans le répertoire courant : les fichiers avec l'extension .c, dans le répertoire courant.

Mais les fichiers ciblés ne sont pas toujours dans le répertoire courant.

Dans ce cas, il y a 2 solutions :

- Se déplacer dans le répertoire où sont les fichiers.
- Cibler les fichiers en indiquant en plus le chemin qui mène à eux.

La solution 2) est en général la meilleure.

Il y a 2 types de chemins :

- Chemins absolus.
- Chemins relatifs.

Chemin absolu

Un chemin absolu est un chemin complet qui part de la racine, “/”.

Exemples :

- /home/etuinfo/billg/algo/exo1.c
- ~/algo/exo2.c

~ signifiant le home de l'utilisateur, il est remplacé par /home/etuinfo/billg (exemple pour l'utilisateur loggué en tant que billg).

On obtient donc un chemin commençant par "/", il s'agit donc bien d'un chemin absolu.

Un chemin absolu cible toujours le même objet, peu importe le répertoire courant.

Chemin relatif

Un chemin relatif est un chemin complet qui part du répertoire courant “.”.

Exemples :

- algo/exo1.c
- ../sys/script.bash
- ../../ens/stevej/

Un chemin relatif ne cible pas forcément toujours le même objet, ça dépend du répertoire courant.

Les commandes acceptent indifféremment des objets spécifiés en chemins relatifs ou absolus.

9 - Indépendance des paramètres

Les paramètres d'une commande sont indépendants. Exemple, cat ../textes/fic1 fic2 affiche 2 fichiers qui ne sont pas au même endroit :

- fic1 se trouve dans le dossier textes qui est lui-même dans le dossier parent.
- fic2 est dans le dossier courant.

Quand cat affiche le 1er fichier, elle reste quand même dans le dossier courant actuel. Une commande ne change jamais le répertoire courant, sauf... laquelle ?

10 - Retour sur les commandes de base

Retour sur les commandes de base vues dans le CM #1.

Dans les exemples donnés, on travaillait toujours dans le répertoire courant. Les chemins nous apportent maintenant une autre dimension. Exemples :

- ls ..
- rm ../old/fic1
- cat ~/.bashrc
- less /etc/passwd

cp - Copie de fichiers et dossiers

Syntaxe : cp [option] source destination

- source peut être un fichier ou un dossier. Si c'est un dossier, il faut obligatoirement utiliser l'option -r (récursif, comme pour rm vu précédemment).
- destination peut être :
  - un fichier si source est un fichier.
  - un dossier, peu importe la nature de source.

Si destination n'existe pas déjà :

- Si source est un fichier, destination sera créé en fichier.
- Si source est un dossier (option -r obligatoire), destination sera créé en dossier.

Si destination existe déjà en tant que fichier :

- Si source est un fichier, destination est écrasé.
- Si source est un dossier (option -r obligatoire) : Erreur et destination est préservé.

Si destination existe déjà en tant que dossier :

- Si source est un fichier, destination va contenir une copie de source avec le même nom (sauf s'il contient un dossier portant le même nom que source ! Ca peut arriver… dans ce cas : Erreur)
- Si source est un dossier (option -r obligatoire) : Copie récursive du contenu de source vers le dossier destination en appliquant ces mêmes règles décrites ici, pour chaque objet copié.

Attention, on peut parfois penser que la destination existe en tant que dossier. Faire cp fic.txt old, en pensant que le dossier old existe : Quel est le résultat ?

Quand on veut que la destination soit un dossier, il faut TOUJOURS ajouter un “/” derrière : cp fic.txt old/, on obtient alors une erreur si old n'est pas un dossier.

Ca ne coûte rien, et c'est sans ambigüité. Mieux vaut une erreur qu'une action imprévue !

Quelques exemples :

- cp exo1.c sauvegarde : Attention, pas évident
- cp exo1.c sauvegarde/ : Sans ambigüité
- cp -r photos ../images/

mv - Renomme ou déplace des objets

Syntaxe : mv [option] source destination

On peut déplacer + renommer, en 1 même opération :

- Pour renommer source, il faut que destination ne soit pas un dossier existant.
- Pour ne pas renommer source, il faut que destination soit un dossier existant.

11 - Répertoires Unix

Voici pour finir quelques répertoires classiques Unix :

- / : la racine du FS.
- /users ou /home : les home des utilisateurs.
- /tmp : fichiers temporaires.
- /bin : commandes (binaires) système.
- /lib : librairies communes.
- /usr : logiciels et applis (Unix System Resources).
- /var : fichiers divers (logs, données appli. variables).
- /etc : configurations et réglages (divers, et cætera).
- /mnt : média amovibles (DVD, USB, dé/montables).

12 - Votre espace de travail

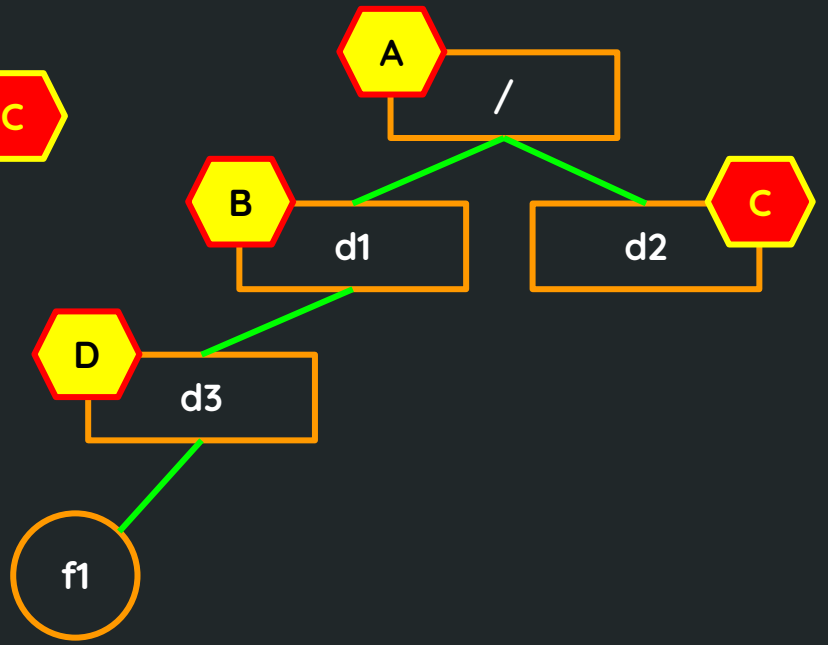
Vous devez organiser proprement votre espace de travail de la manière suivante :

- Exemple pour les TD de Système :
  ~/sys/td/td1/<ce_que_vous_voulez>

- Exemple pour les TP d'Algo :
  ~/algo/tp/tp1/<ce_que_vous_voulez>

13 - Jouons dans l'arbre

Soit cet arbre fictif.
Le répertoire courant est en C

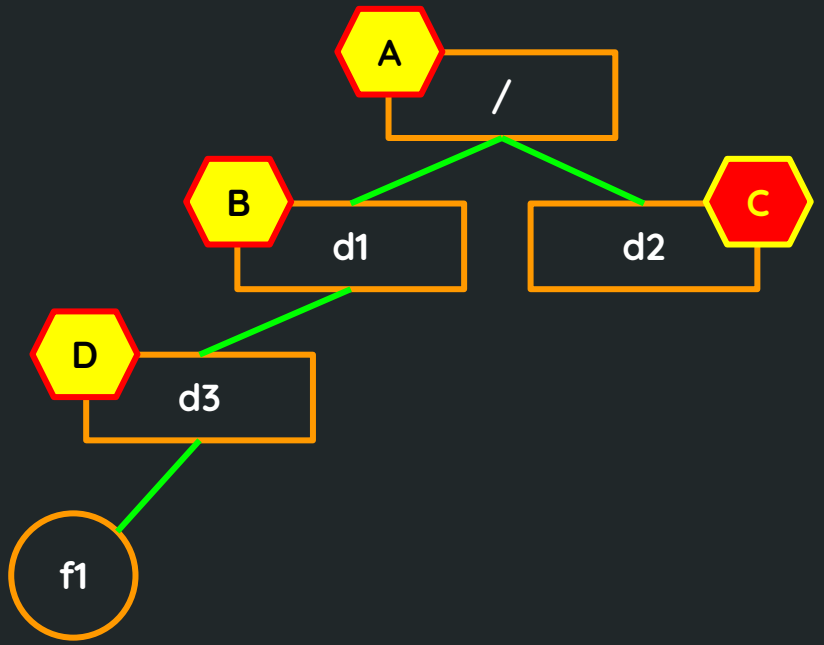


Quels sont les objets désignés par :

- /d1/d3/f1
- /d2
- /d1/d3
- d3/f1

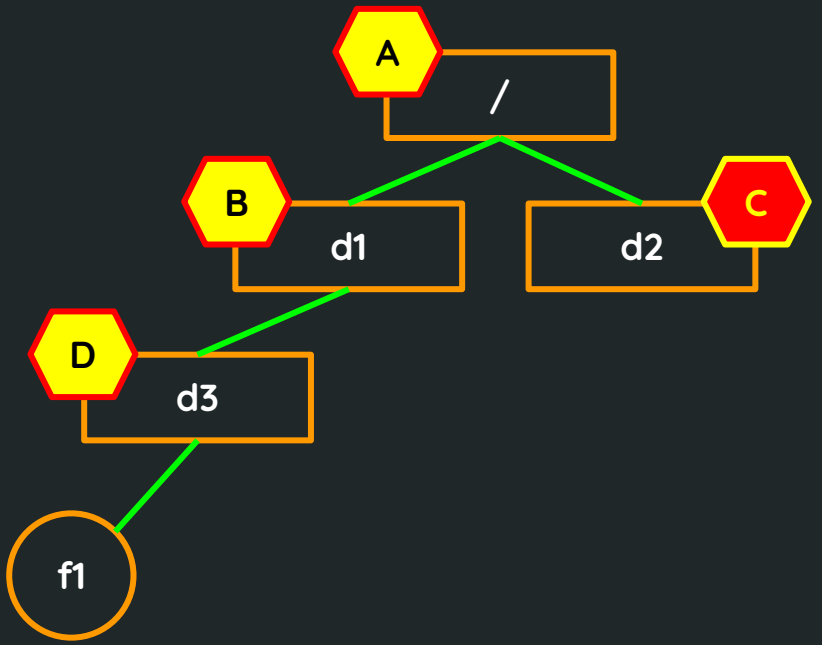
(suite) Quels sont les objets désignés par :

- ../d2
- ../d1/d3
- ../d1/..
- /d1/../d1
- d2



Où peut-on être pour que ces chemins désignent un objet existant ?

- d2
- f1
- ../d3
- ../d1
- /



https://commons.wikimedia.org/wiki/File:Thats_all_folks.svg