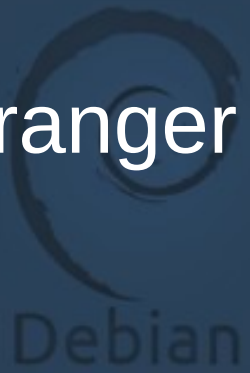
Chapitre 2 : Fichiers et dossiers

fichiers et répertoires
nommage
commandes et noms complets

2.1 - Dossiers



Pour bien ranger les fichiers

« Répertoires » de fichiers

- **Dossier = classeur, rangement de fichiers**
  - pour organiser les fichiers (documents) logiquement
  - autres noms : **répertoire**, **directory**, folder, catalogue *(termes mal choisis, correspond à la représentation interne)*

Pourquoi ce nom de *répertoire* ?

- En interne, un disque dur (ou carte mémoire) contient des *blocs*.
- Un *bloc* = espace de taille fixe pour des données, en général 4096 octets

  C'est la quantité forfaitaire de toute lecture ou écriture

  Un disque de 2 To = 2x1024x1024x1024x1024 / 4096 blocs !

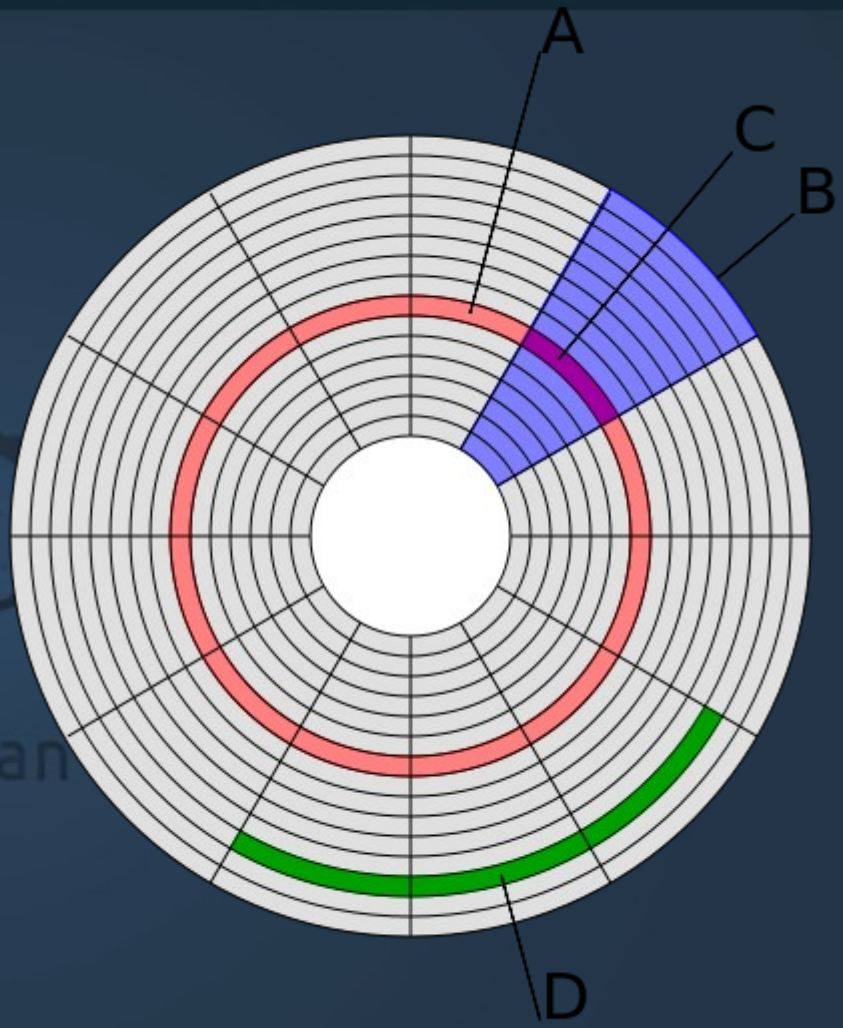
Disque dur

- Intérieur d'un disque dur (pas SSD) (img internet) :

Structure logique

- Chaque face est découpée de manière abstraite en cylindres et secteurs
- Un bloc = quelques secteurs du même cylindre

  (image Wikipédia)



Pourquoi « répertoire » ? (suite)

- Un fichier est stocké sur un ou plusieurs blocs

  => il faut mémoriser leurs numéros

- Un dossier = liste de (nom, type, liste des numéros des blocs du fichier)

  Ça ressemble à un répertoire téléphonique (nom, n^{os} de téléphone)

  d'où ce nom de *répertoire*

- Voir le cours de P3 pour davantage d'infos

Imbrication de dossiers

- Les dossiers peuvent être imbriqués :

NB : un fichier ou dossier ne peut être que dans un seul (autre) dossier

Nommage des éléments

- Dans un dossier, on ne peut avoir qu'un seul élément d'un certain nom (= couleur dans les dessins)
- Le même nom peut se trouver dans différents dossiers : cherchez l'erreur ici

En résumé

- **Un dossier peut contenir un nombre quelconque de fichiers, et aussi d'autres dossiers (sous-dossiers).**

- **Seule contrainte : dans un même dossier, les noms des éléments (extension comprise) sont uniques.**

Création de dossiers

- Création d’un dossier (directory) :
  - **`mkdir nomdudossier`**
- Destruction d’un dossier vide :
  - **`rmdir nomdudossier`**

NB: ne marche pas s'il reste des fichiers dedans, y compris des fichiers « cachés »

Fichiers cachés

- Lorsque le nom d'un fichier commence par un point, il n'est pas affiché par la commande `ls`
  - Fichiers et dossiers de configuration `.bashrc` `.config` `.gconf` etc
  - Dossiers spéciaux appelés `.` et `..`
- Pour voir ces fichiers et dossiers :
  - Dans le shell, il faut faire `ls -a` (ou `--all`)
  - Dans l'interface à fenêtre, il faut appuyer sur CTRL H ou le menu affichage, montrer les fichiers cachés

2.2 – Arbre de dossiers

Les dossiers sont imbriqués et cette structure a une forme d'arbre

Dossier de compte

- Chaque utilisateur possède un dossier personnel associé à son compte

Home dir

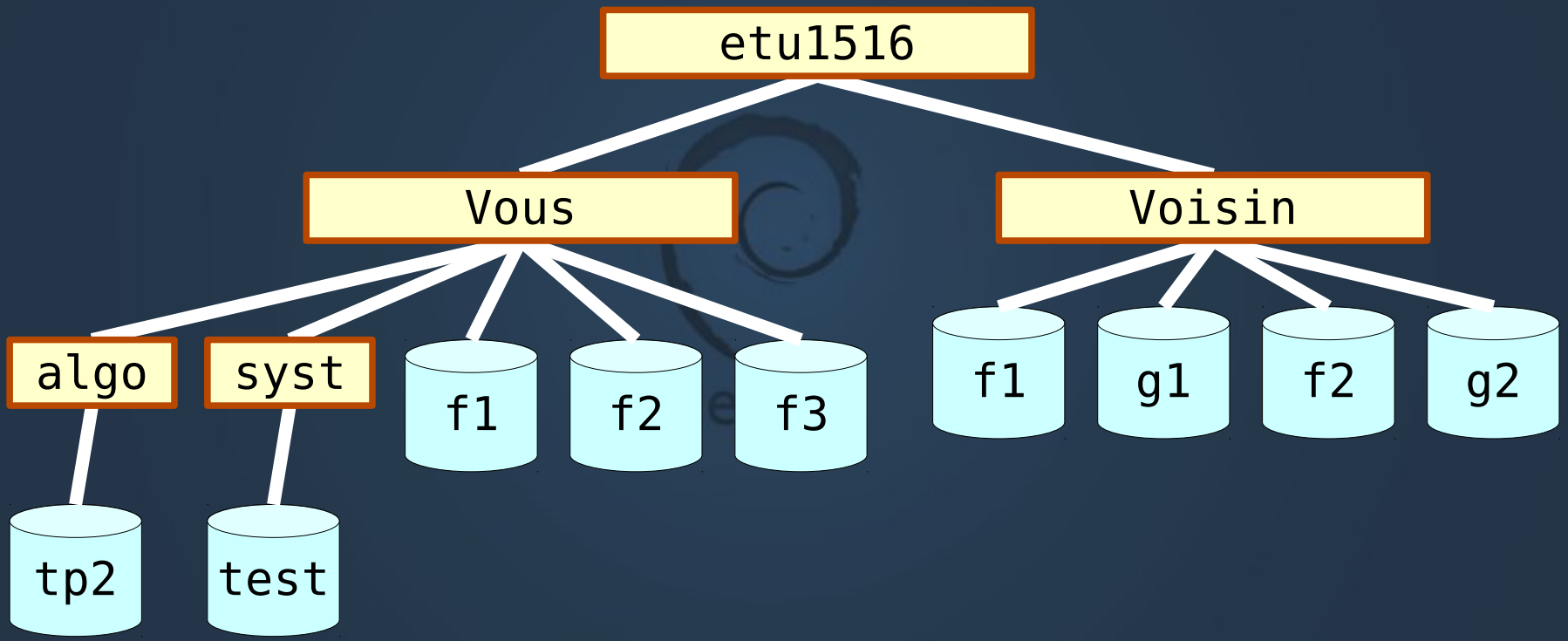
- **Le dossier du compte d'un utilisateur est le « répertoire d'origine » ou « home dir »**
  - C'est un dossier dans lequel il peut créer des fichiers et des sous-dossiers
  - Il s'appelle comme le nom de connexion
- **L'ensemble des *home dirs* est placé dans un dossier englobant.**
  - A l'IUT, il est appelé `etu1516` pour les étudiants
  - Il y a aussi un tel dossier pour les enseignants

Regroupement des comptes

- Les home dirs sont regroupés dans un autre dossier

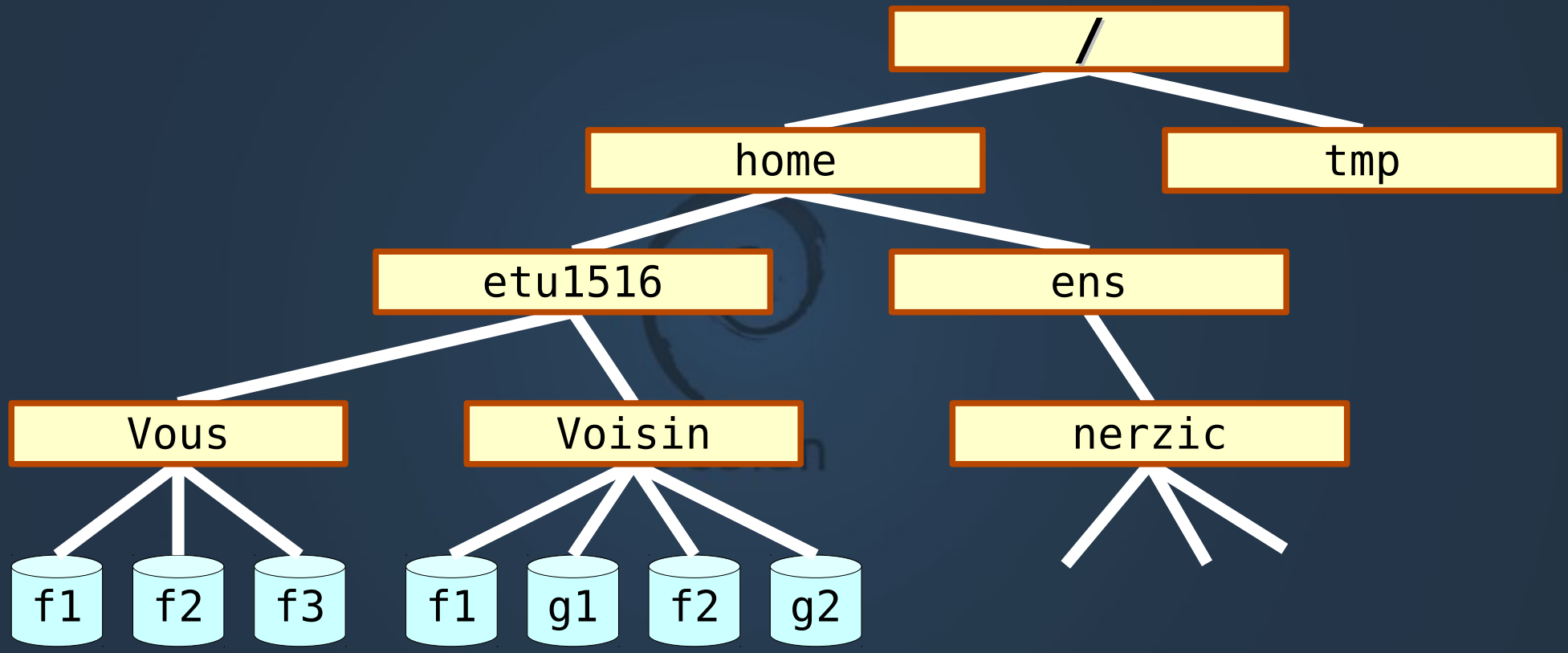
Autre représentation

- La structure se dessine autrement : un arbre



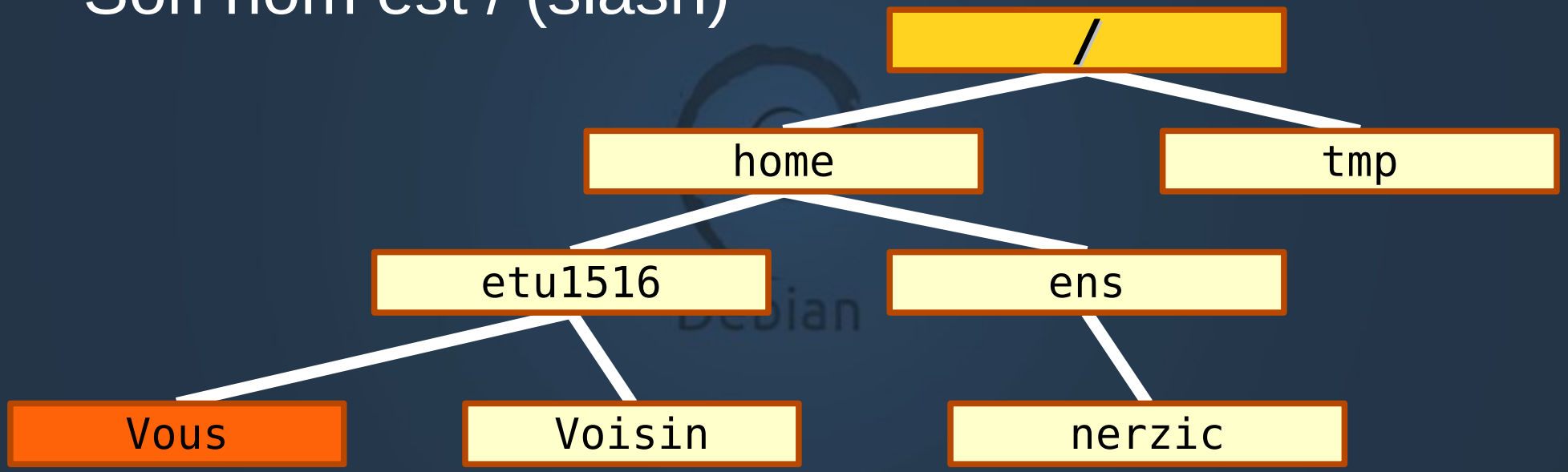
Arborescence Unix

- L'arbre complet ressemble davantage à ceci :



Arborescence (suite)

- Le sommet de l'arbre s'appelle la racine, root en anglais
- Son nom est / (slash)



Dossier parent

- Chaque fichier existant est obligatoirement quelque part dans un dossier
- Chaque dossier, excepté la racine, est obligatoirement dans un seul autre dossier : son **parent**.
  - Le dossier parent de « algo » est « vous »
  - Le dossier parent de « nerzic » est « ens »
- Il ne peut pas y avoir de boucle dans cet arbre. En remontant tout en haut, on arrive à /

2.3 – Dossier courant

Quand vous êtes connecté et que vous travaillez, là où vous êtes, c'est le « répertoire de travail » ou « dossier courant »

Home dir et Working dir

- **A la connexion, un utilisateur est placé dans le dossier de son compte (*home dir*)**
- **Travailler se fait obligatoirement dans un dossier**
- **Le répertoire actuel est appelé répertoire de travail (*working dir*, wd en abrégé)**
- **ls, more, cp, mv, rm... agissent par défaut sur les fichiers du répertoire de travail.**

Vision dans un dossier

- Dans un dossier, on ne voit que ce qu'il contient
  - pas le contenu des sous-dossiers
  - pas au delà

Autre exemple

- Le personnage ne voit que :
  - Les trois fichiers vert clair, bleu clair et lavande
  - Les deux dossiers algo et divers

Changer de dossier courant

- Pour travailler dans un autre dossier, il faut utiliser la commande cd
  - On indique dans quel sous-dossier on veut aller
  - Ce sous-dossier doit être là où on est actuellement
- Ex : cd divers

Commande cd

- **`cd`**

  sans paramètre, ramène au home dir

- **`cd` *`dossier`***

  avec un nom de dossier en paramètre, va dans ce dossier. Attention, ce dossier doit être dans le dossier courant (cf nommage plus loin).

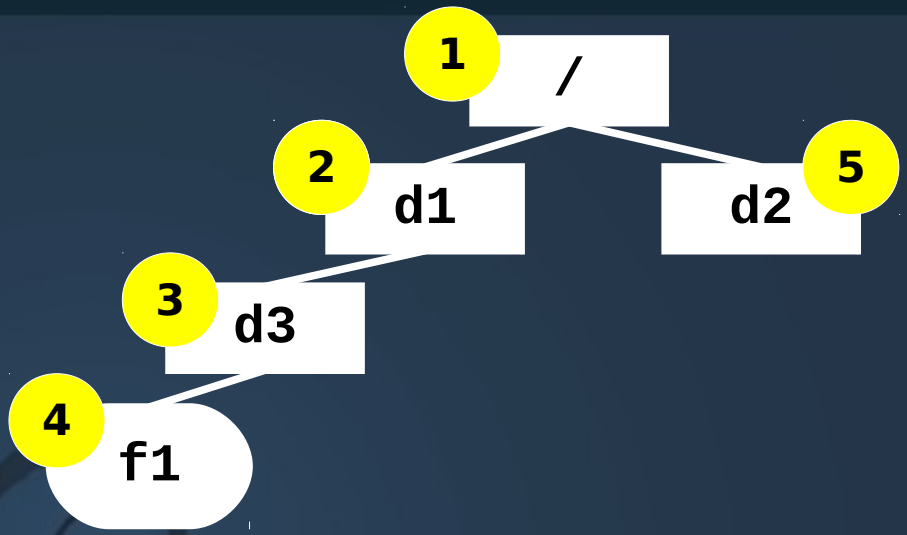
  Ça ne peut pas être le nom d'un fichier

- **`cd ..`**

  `..` désigne le dossier parent, cela fait monter d'un niveau dans l'arbre

Exemples

- Voici un arbre fictif :



- On part de 2, où amènent les séquences ?

```
cd .. puis cd d1 puis cd d3
cd .. puis cd d2 puis cd ..
cd d3 puis cd d1 puis cd /
```

2.4 – Noms complets

Désignation complète d'un fichier à travers tout un arbre de dossiers

Travail avec des fichiers éloignés

- Certaines commandes font référence à des fichiers situés dans des dossiers différents :
  - dupliquer un fichier d'un dossier à un autre
  - déplacer un fichier situé dans algo et le mettre dans sys
- **Il faut employer un nom complet :**

  la désignation du dossier et le nom du fichier concerné
- syntaxe : **`chemin/nom`**
  - `/usr/local/anonymous/SYS/1A/tp2/compliments.txt`
  - `../algo/proverbes.txt`

Deux sortes de chemins

- Pour désigner un fichier ou dossier distant, on doit indiquer les étapes qui y mènent :

  « d'abord il faut aller dans tel dossier, puis dans tel autre, etc. et pour finir, le fichier s'appelle ... »

- Le **point de départ** de cette liste d'étape peut être :
  - **Le dossier courant :** **chemin relatif**
  - **La racine du système :** **chemin absolu**
  - Le « home dir » : chemin absolu aussi

Chemins absolus

- **Chemin absolu = itinéraire partant de la racine**

  Exemples :

  - `/home/etu1516/vous/algo/proverbes.txt`
  - `/usr/lib/X11/config`
  - `/var/log/boot.log`
  - `/media/USB-767-987/readme.txt`
- **Le premier `/` est indispensable, il indique qu'on part de la racine**, s'il est absent, le chemin n'est pas absolu (donc il est relatif).

Autre exemple

- La syntaxe des URL = *Uniform Resource Locators* utilise cette notion de chemin absolu :

  http://fr.exemple.org**/docs/wiki/internet/url.html**

Chemins partant du home dir

- Il y a un cas particulier de chemins absolus, **ceux qui partent du compte** (home dir). Ils doivent commencer par **~/**

  Exemples :

  ```
  ~/algo/tp1/taxe.c
  ~/sys/tp2/partie1/stock/recit
  ```

- En fait, ~/ est rendu équivalent à **/**u/etu1213/vous, donc c'est un chemin absolu
- Attention, ~algo (sans /) est rendu équivalent à /u/algo

Chemins relatifs

- **Chemin relatif = itinéraire partant du répertoire courant**

  Exemples :

  `algo/proverbes.txt`

  `../../voisin/sys/nuages.txt`

  `sys/../algo/../sys/tp2/partie1/ok`

- **Ne surtout pas mettre de / au début**
- **Les `..` désignent le répertoire parent** :

  ex: `/../algo` est un chemin absolu, tandis que `../usr/local/anonymous` est relatif

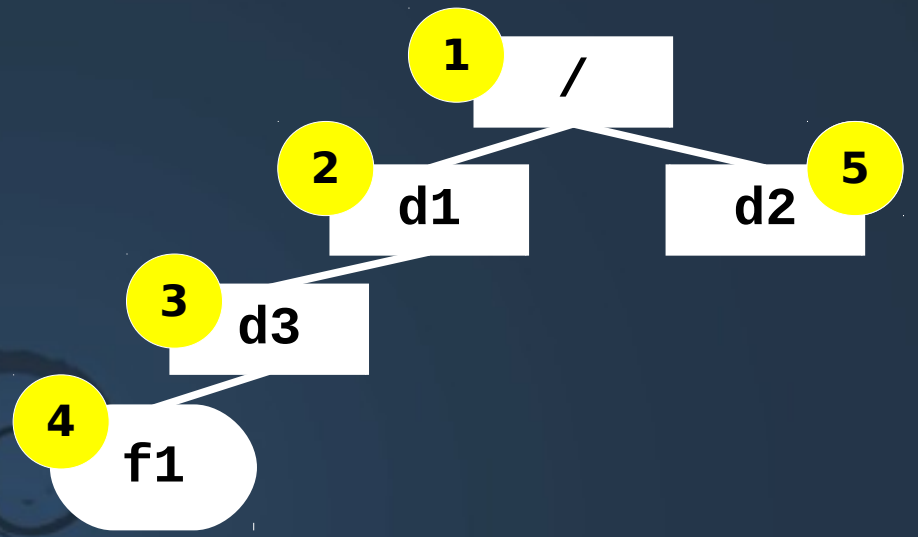
Caractères spéciaux

- Il existe des raccourcis vers des dossiers :
  - `/` la racine
  - `~` le home dir
  - `..` le répertoire parent du répertoire courant
  - `.` le répertoire courant
- Exemples :

```
~/algo/proverbes.txt
../sys/../algo
./truc.txt
```

Exercices

- Soit cet arbre fictif,
  on se situe en 5



- Quels sont les choses désignées par ces noms :

`/d1/d3/f1`	`/d2`	`/d1/d3`
`d3/f1`	`../d2`	`../d1/d3`
`../d1/..`	`/d1/../d1`	`d2`

2.5 – Paramètres noms complets

Toutes les commandes acceptent des noms complets absolus ou relatifs en paramètres

Noms et paramètres

- **Il n’y a pas de distinction entre fichiers et répertoires en ce qui concerne le nommage**
  - `mkdir /tmp/essai`
  - `vi algo/essai`
  - `rm -r ../tmp/essai`
- **C’est la commande qui vérifie que son paramètre est un fichier ou un répertoire comme ce qu’elle souhaite** (erreur si ce n'est pas le cas)

Commandes et noms complets

- **Toute commande accepte aussi bien des noms complets absolus que relatifs**
  - `vi essai.txt`
  - `vi ../algo/essai.txt`
  - `cp /tmp/essai.txt ../sys/tp1`
- **Chaque paramètre est indépendant**
  - La commande recherche chacun des fichiers ou dossiers indépendamment avant de faire le travail
  - Il faut écrire chaque nom complet en fonction du répertoire courant, pas en fonction du chemin du précédent.

Indépendance des paramètres

- Un exemple : on se situe dans le répertoire **~/sys** de votre compte (cf TP commun algo-sys)
  - Que pensez-vous de :

    `cp ~/algo/tp1/partie1/proverbes.txt ..`

    - Est-ce que ça copie proverbes.txt dans algo/tp1 ou bien au niveau de votre compte ?
  - Que pensez-vous de :

    `mkdir ../algo/tp2 tp3`

    - Est-ce que ça crée un dossier tp3 dans algo ou dans sys ?

Commande mv *(move)*

mv cheminsrc/nomsrc chemindst/nomdst

- selon ces chemins et ces noms :
  - cheminsrc ≠ chemindst ou nomdst absent : déplacement du fichier
  - nomsrc ≠ nomdst : renommage du fichier
- exemples :
  - `mv /tmp/toto /tmp/titi`
  - `mv /tmp/toto /var/toto`
  - `mv /tmp/toto /var/titi`
  - `mv /tmp/toto /var`

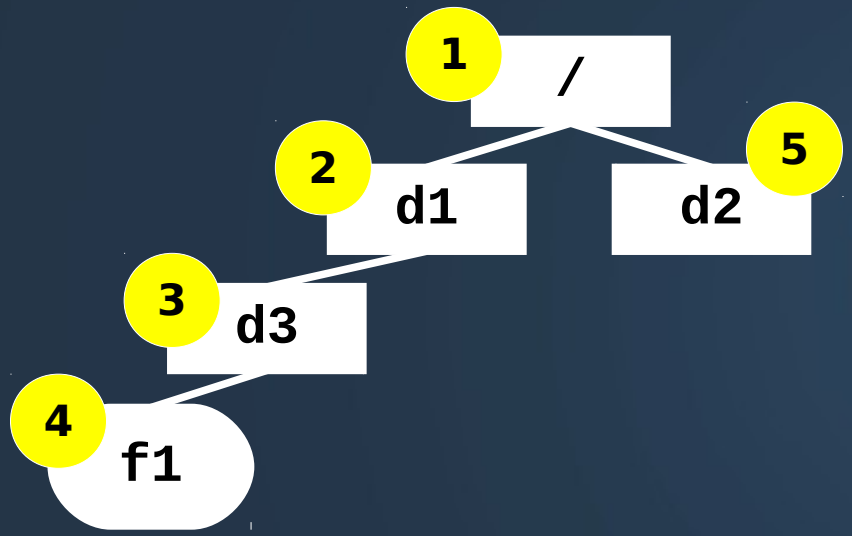
Commande cp *(copy)*

cp cheminsrc/nomsrc chemindst/nomdst

- selon ces chemins et ces noms :
  - cheminsrc ≠ chemindst ou nomdst absent : copie placée ailleurs
  - nomsrc ≠ nomdst : renommage de la copie
- exemples :
  - `cp /tmp/toto /tmp/titi`
  - `cp /tmp/toto /var/toto`
  - `cp /tmp/toto /var/titi`
  - `cp /tmp/toto /var`

Exemples de commandes

- On veut copier f1 (4) dans d2 (5) :

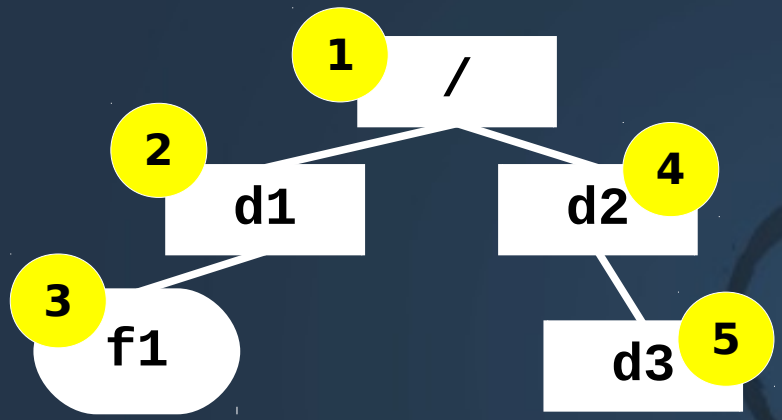


Quel est le répertoire de travail pour pouvoir faire :

```
cp  f1          /d2
cp  d3/f1       ../d2
cp  f1          ../../d2
cp  /d1/d3/f1   d2
cp  /d1/d3/f1   .
cp  /d1/d3/f1   /d2
```

Autre exemple

- On veut déplacer f1 (3) dans d2 (4)

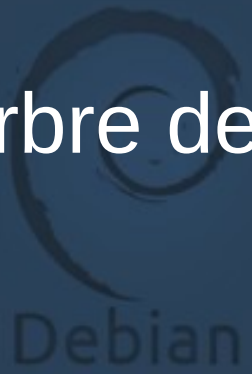


Quel est le répertoire de travail pour pouvoir faire :

- `mv f1 /d2`
- `mv /d1/f1 .`
- `mv ../../d1/f1 ..`
- `mv d1/f1 d2`
- `mv f1 ../d2`

2.6 – Divers

Que contient l'arbre des fichiers Unix ?



Arbre de fichiers Unix

- Unix a défini le rôle de certains répertoires :
  - **/u** ou /users ou **/home** : les comptes
  - **/tmp** : les fichiers temporaires
  - **/bin** : les commandes standard, ex ls, cp, rm
  - /lib : les librairies de fonctions
  - **/etc** : les fichiers de configuration (réglages)
  - **/usr** : les logiciels et leurs fichiers de configuration
  - /var : les fichiers de log (livres de bord) et autres
  - /mnt, **/media** : les répertoires des disques amovibles

Organisation Windows

- Dans Windows 7, on trouve :
  - C:\Program Files : les logiciels
  - C:\Program Data : les réglages des logiciels
  - C:\Users : les comptes
  - C:\Windows : les fichiers du système
- Remarque : Unix et Windows ont beaucoup évolué sans prendre grand soin de bien organiser les fichiers

Dernières remarques

- Il sera exigé de faire des répertoires pour :
  - chaque matière
    - chaque TP
      - éventuellement, chaque partie de TP
- Simple question de bon sens
- Exemples :

  ~/algo/tp1

  ~/sys/tp4/partie1