



Aujourd'hui : système de fichier

- Définition de l'arborescence
- Sur le disque
- Racine UNIX
- Commandes associées



Systeme de fichiers

Ce qui fait correspondre comment les fichiers sont physiquement stockés sur les **disques** (via des séquences de 0 et de 1) et ce avec quoi l'utilisateur interagit (une **arborescence**)

Selon le contexte :

- **organisation hiérarchique des fichiers au sein d'un système d'exploitation**
- **organisation des fichiers sur un médium (ext[2-4], NTFS, FAT, FAT32, ...)**

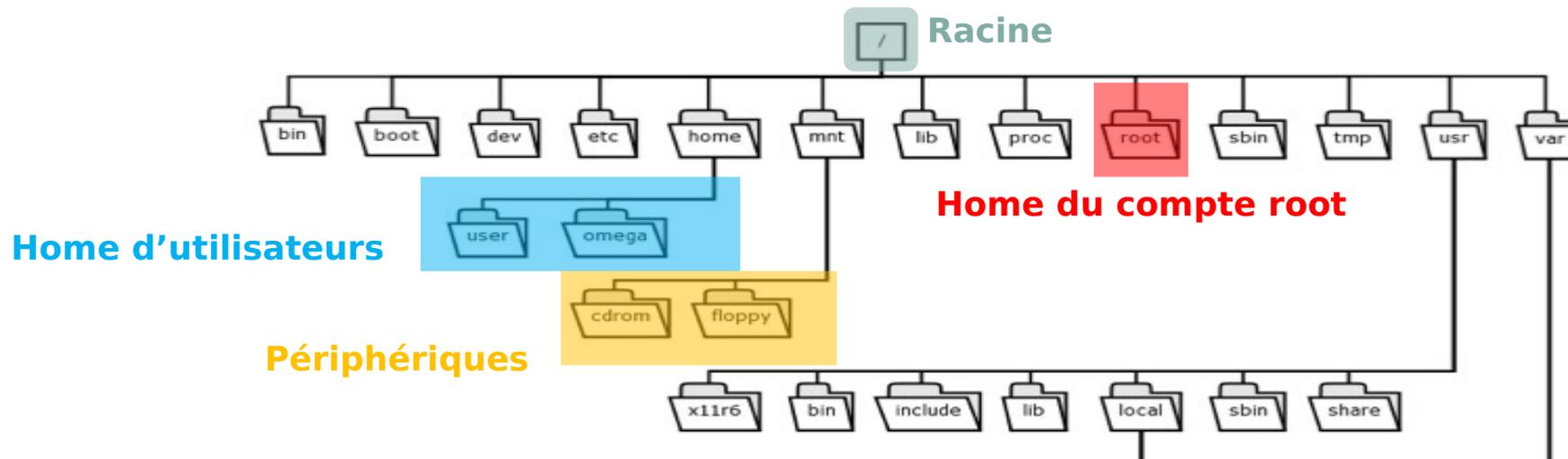


Systeme de fichiers UNIX : L'arborescence

Arborescence UNIX

Les **fichiers** (fichiers systèmes et fichiers utilisateur) sont **tous** rangés dans des **répertoires** organisés dans **une** structure hiérarchique : l'**arborescence**

- **Root** (« racine » en français, notée « / ») = le répertoire initial
/!\ à ne pas confondre avec le **compte root**, ni avec le « / » utilisé dans les **chemins**.
- **Home** (« maison » en français) = un dossier réservé à chaque compte du système.
- On peut « **monter** » des périphériques extérieurs (clés USB, DDE, ...) dans l'arborescence (/mnt, /media)
→ pas de disques différents comme sur Windows (A:, C:, D:...).



Chemins UNIX

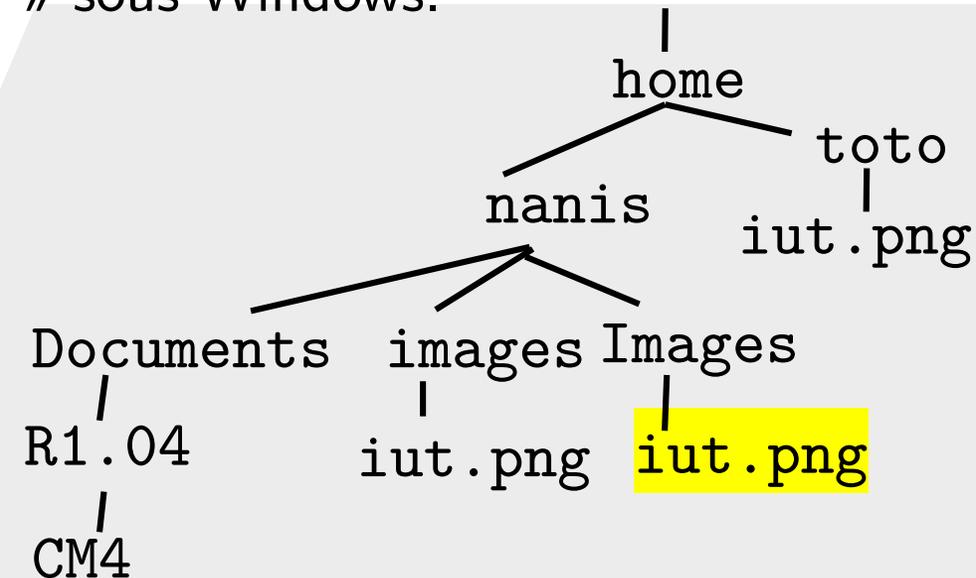
Un chemin est **chaîne de caractère qui décrit l'emplacement d'un fichier**.

- **absolu** : défini depuis la racine du système de fichiers « / » (ou home « ~ »)
- **relatif** : défini à partir de notre localisation en utilisant « . » (répertoire courant) et « .. » (répertoire parent).

/!\ le **séparateur de chemin** est « / » sous UNIX mais « \ » sous Windows.

Exemple : on est dans `/home/nanis/Documents/R104/CM4` et on veut faire référence au fichier `iut.png`.

Wooclap





Wooclap



1

Allez sur wooclap.com

2

Entrez le code
d'événement dans le
bandeau supérieur

Code d'événement
XZBBRK



Activer les réponses par SMS

Chemins UNIX

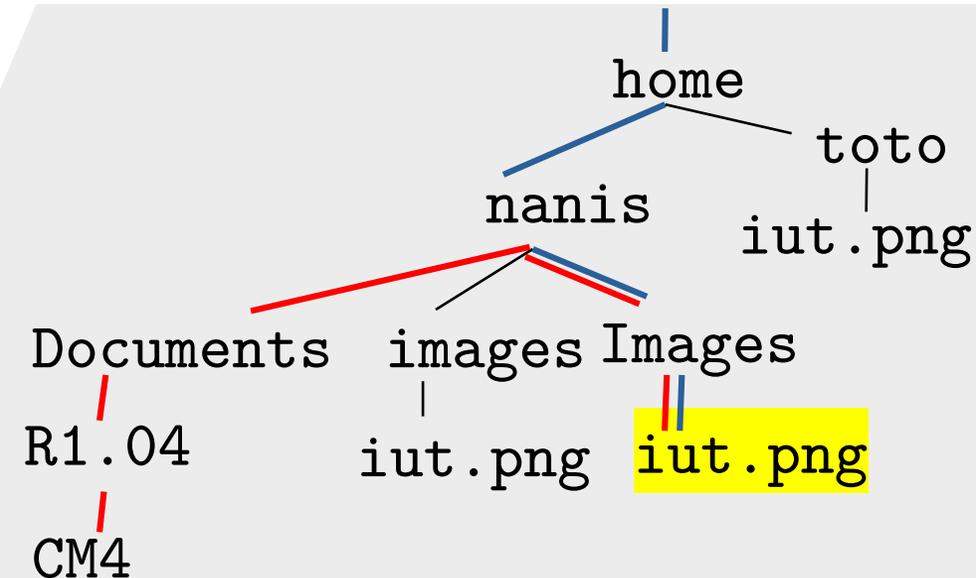
Un chemin est **chaîne de caractère qui décrit l'emplacement d'un fichier**.

- **absolu** : défini depuis la racine du système de fichiers « / » (ou home « ~ »)
- **relatif** : défini à partir de notre localisation en utilisant « . » (répertoire courant) et « .. » (répertoire parent).

/!\ le **séparateur de chemin** est « / » sous UNIX mais « \ » sous Windows.

Exemple : on est dans `/home/nanis/Documents/R104/CM4` et on veut faire référence au fichier `iut.png`.

- `/home/nanis/Images/iut.png` (absolu)
- `../../../../Images/iut.png` (relatif)



Note : Toutes les commandes UNIX qui manipulent les fichiers acceptent toutes les formes de chemins

<https://doc.ubuntu-fr.org/chemins>

Noms de fichiers

- Idéalement, composés exclusivement de caractères alphanumériques (accents possible), du tiret « - », du souligné « », et du point « . ».
- Ils sont sensibles à la casse : `toto.txt`, `TOTO.txt`, `toto.TXT`
- Certains caractères nécessitent d'être « **échappés** » quand on les utilise dans un nom de fichier en ligne de commande. On les fait précéder de « \ ». : « \\$ », « \\ »
- Les caractères interdits ou à éviter sont :
 - « / » : **Interdit**, sert à séparer les noms de répertoire dans un chemin
 - « \ » : À **échapper**, et peut poser des problèmes de compatibilité avec d'autres systèmes d'exploitation comme Windows. À éviter.
 - « - » : À **éviter** en début de nom (le shell va l'interpréter comme une option de commande...)
 - « * », « ? », « : », « ' », « " », « # », « \$ », « ; », « ! », « & » , « | », parenthèses « () », accolades « {} », chevrons « <> », crochets « [] » : **À éviter**.
 - Espace : À **échapper**.
 - « . » en début de nom indique que le fichier est caché.

Noms génériques de fichiers

Un **nom générique** est un nom qui contient un ou plusieurs caractères spéciaux (**méta-caractères** ou **joker**). Il permet de désigner un ensemble d'objets.

- « ? » signifie n'importe quel caractère

Exemple : « t?t? » correspond entre autre à « toto » qu'à « titi » mais pas à « tooto ».

- « * » signifie n'importe quelle chaîne de caractères (y compris vide).

Exemple :

« bon*.txt » peut correspondre entre autre aux noms de fichiers suivants : « bonjour.txt », « bonsoir.txt », « bon_à_rien.txt », « bon_j_ai_pas_d'autre_idée_mais_vous_avez_compris_hein.txt », « bon.txt », mais pas à « bonjour », « bon », ni « jour.txt ».

- **Exception** : Le caractère « . » ne peut pas être remplacé par un joker s'il est au début d'un nom de fichier. Conséquence directe : une chaîne avec joker porte soit sur les fichiers non-cachés, soit sur les fichiers cachés, mais pas les deux en même temps.

Noms génériques de fichiers

- `[]` signifient un caractère appartenant à un ensemble de valeurs décrites dans les crochets, si les fichiers résultants existent
`t[oi]to` → toto, tito
 - utilisé avec les crochets permet de définir un intervalle, plutôt qu'un ensemble de valeurs.
`t[a-d]to` → tato, tbto, tcto, tdto
 - ! ou ^ utilisé entre crochets en première position, signifie tout caractère excepté ceux spécifiés entre crochets.
`t[!o]to` → pas toto
- `{ }` signifient un caractère appartenant à un ensemble de valeurs décrites dans les crochets, même si les fichiers résultants n'existent pas.
`t{o,i}to` → toto, tito
 - .. utilisé avec les accolades, permet de définir un intervalle, plutôt qu'un ensemble de valeurs.
`t{a..d}to` → tato, tbto, tcto, tdto



Noms génériques de fichiers – Exemple

```
prompt> ls test*
```

```
prompt> touch test_[ab].txt
```

```
prompt> ls test*
```

```
prompt> touch test_{a..j}.txt
```

```
prompt> ls test*
```

```
test_a.txt test_c.txt test_e.txt test_g.txt test_i.txt
```

```
test_b.txt test_d.txt test_f.txt test_h.txt test_j.txt
```

```
prompt> ls test_[ab].txt
```

```
test_a.txt test_b.txt
```



Wooclap



1 Allez sur wooclap.com

2 Entrez le code
d'événement dans le
bandeau supérieur

Code d'événement
XZBBRK

 Activer les réponses par SMS

Noms génériques de fichiers – Exemples

Tous les fichiers dont le nom...

f* commence par 'f'.

f? commence par 'f', suivi d'un seul caractère quelconque.

f[12xy] commence par 'f', suivi d'un caractère à choisir parmi '1', '2', 'x' ou 'y'.

f[a-z] commence par 'f', suivi d'un caractère dont le code ASCII est compris entre le code 'a' et le code 'z', donc une lettre minuscule.

***.c** fini par .c (= dont l'extension est .c)

?c est formé d'un caractère quelconque, suivi de '.c'

?? est formé de deux caractères.

***.[A-Za-z]** se termine par un '.' suivi d'une seule lettre majuscule ou minuscule.

***.[ch0-9]** se termine par un '.' suivi d'un seul caractère à choisir parmi 'c', 'h', ou un chiffre entre '0' et '9'.

[!f]* ne commence pas par 'f'

***[!0-9]** ne se termine pas par un chiffre.



Noms génériques de fichiers – les classes

La **norme POSIX** définit des classes (groupes) de caractère suivantes :

`[:upper:]` pour les majuscules

`[:lower:]` pour les minuscules

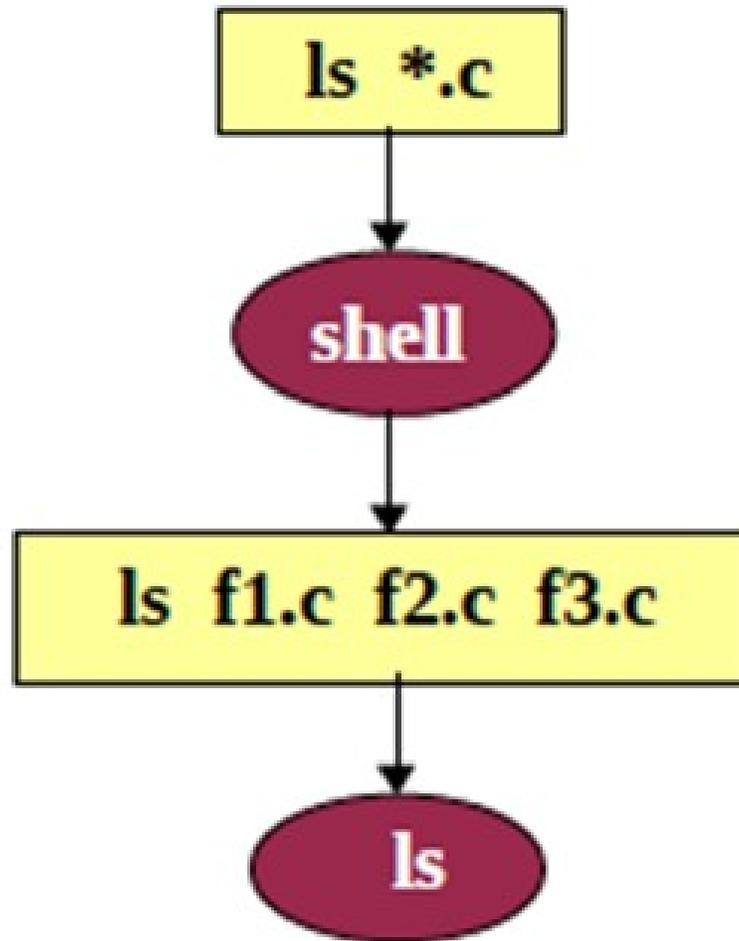
`[:digit:]` pour les chiffres de 0 à 9

`[:alnum:]` pour les caractères alphanumériques

Exemple :

`ls ./[[:upper:]]*` → tous les noms de fichiers qui commencent par une lettre majuscule.

Noms génériques de fichiers – substitution



Le traitement des méta-caractères est indépendant de la commande. Il est **effectué par le Shell, avant l'exécution de la commande.**

Chaque ligne est donc analysée deux fois :

- 1) **Par le Shell :**
si la ligne contient un joker, le shell remplace le motif par les valeurs possibles.
Exemple : le shell lit `ls f*.c`. Il enlève « `f*.c` » et le remplace par « `f1.c f2.c f3.c` ». Le résultat est donc `ls f1.c f2.c f3.c`.
- 2) **Par la commande :** lorsque la commande `ls` est exécutée, elle reçoit « `f1.c f2.c f3.c` ». la commande va donc travailler sur les 3 fichiers.



Le système de fichier : correspondance avec le disque



Différents systèmes de fichiers

Comment les fichiers sont stockés sur les disques

- Sous windows : FAT, FAT32, NTFS
- Sous Linux : ext2, ext3

Implémentations et caractéristiques différentes (tailles max d'un fichier, gestion des droits, etc).

Systeme de fichiers : generalites

Dans un flat file system (« plat ») :

- Pas de hierarchie, tous les fichiers sont à la racine, et rangés à la queueleu sur le disque (arg si changement de taille...).
- un fichier **directory file (DF)** stocke l'adresse où lire les données d'un fichier (~ sommaire d'un livre)



Dans un système de fichiers moderne :

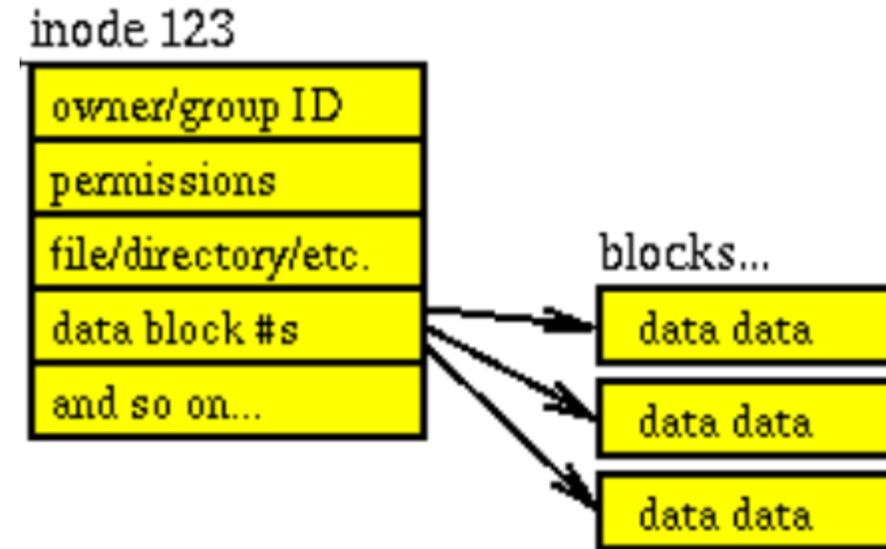
- Les fichiers sont mis dans des **blocs** de taille standardisée (ça laisse un peu de marge : **slack space**). Ils sont mis dans plusieurs block s'ils sont plus gros qu'un bloc (fragmentation).
- S'il y a niveaux de dossiers, chaque dossier a un **directory file** qui stockent le bloc (ou plutôt une liste de blocs) où est rangés chacun de ses éléments (fichier ou dossier).



Systeme de fichiers : inodes de fichiers

Structure de données qui contient les **métadatas** d'un fichier :

- un identifiant (**inode number**),
- la taille du fichier,
- l'identifiant du périphérique contenant le fichier,
- l'identifiant du propriétaire du fichier, et du groupe,
- le mode du fichier (ses droits d'accès),
- l'horodatage du fichier (date de modification de l'inode, du fichier et de dernier accès)
- l'adresse des datas du fichiers

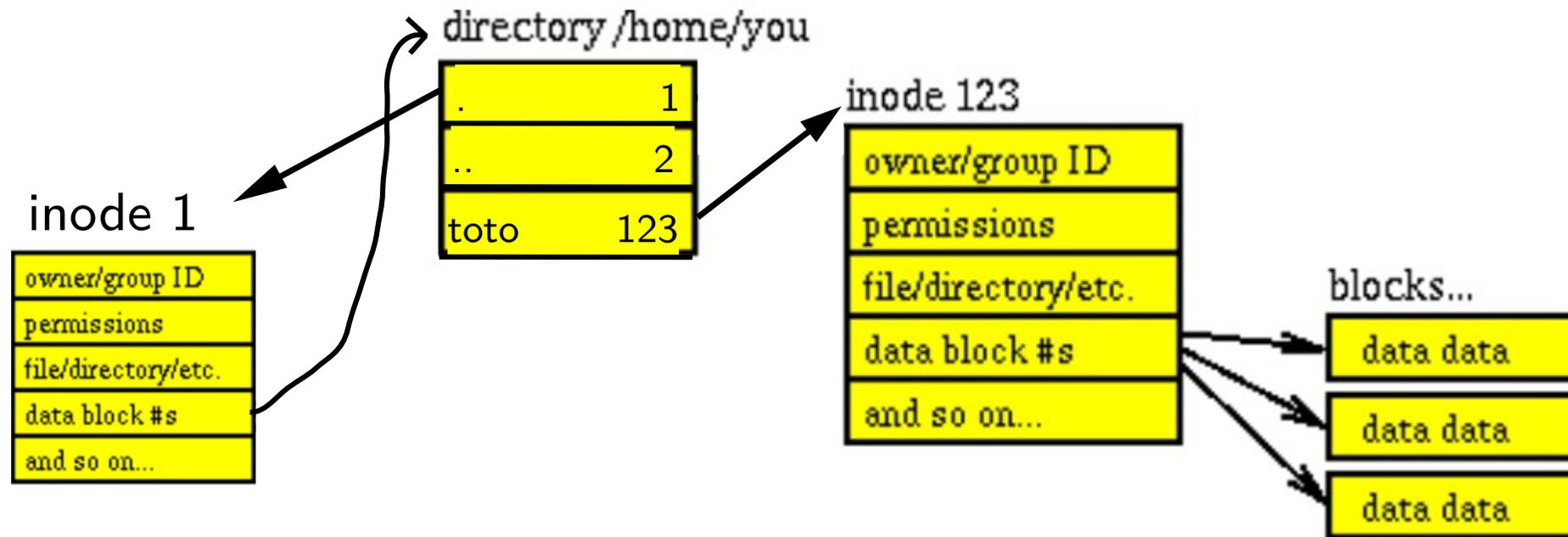


Attention ! L'inode d'un fichier ne contient **pas son nom** ! (Un fichier peut en effet être présent à plusieurs endroits dans l'arborescence de fichiers, sous différents noms ; on en reparle plus tard dans les slides sur les liens physiques et symboliques,)

Similaires aux **tables d'allocation de fichiers** dans les systèmes de type FAT (Windows)

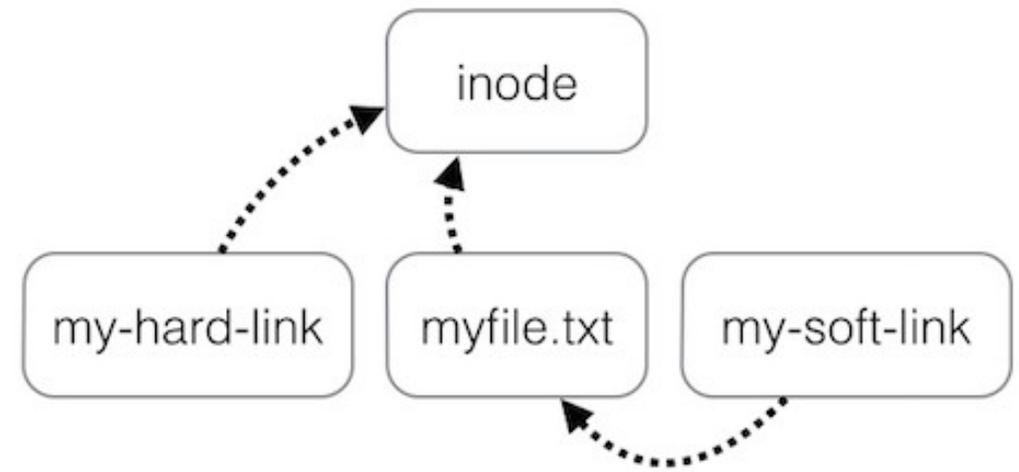
Systeme de fichiers : les repertoires

Un repertoire est comme un fichier ordinaire : il possede un inode (contenant les metadatas), et ses donnees (table de liens [chaîne de caractere, i-nombre]) sous forme de suite d'octets.



Systeme de fichiers : les liens

- Lien dur « hard link » : un fichier qui pointe **vers** le même **inode** qu'un autre fichier.
- Lien symbolique « soft link » : un fichier qui pointe **vers un chemin donné**.

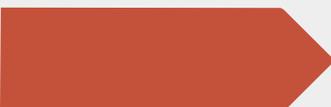




Différentes catégories de fichiers

En UNIX, « **tout est fichier** »

- fichiers **normaux** : documents odp, sources des programmes, fichier textes de configuration, images, archives zip, **exécutables** (programmes en code binaire).
- fichiers **répertoires** (les **dossiers**), qui peuvent contenir d'autres fichiers.
- fichiers **liens symboliques**
- fichiers **spéciaux**
 - dans /dev, le système d'exploitation prépare des canaux de communication avec les périphériques. **Démo** : **tty ; echo toto > /dev/pts/0**



Systeme de fichiers : La racine des UNIX



L'organisation à la racine

```
cd / ; ls
```

Standard Filesystem Hierarchy Standard – FHS maintenu par la fondation Linux.

https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

- A changé au fil des années
- Pas forcément respecté par toutes les distribs



Wooclap



1

Allez sur wooclap.com

2

Entrez le code
d'événement dans le
bandeau supérieur

Code d'événement
XZBBRK



Activer les réponses par SMS



L'organisation à la racine

- **/bin** : [binaries] les exécutable
- **/boot**: le noyau et les fichiers de démarrage
- **/dev** : [device file] le répertoire des fichiers spéciaux pour communiquer avec les périphériques (« tout est fichier », même le hardware : disque `dev/disk/sda`, webcam, clavier...)
- **/etc** : [etc -- D. Ritchie / Edit To Configure] les fichiers de config au niveau du système (« system-wide »)
 - **/etc/rc.d** scripts de démarrage du système
 - **/etc/cron** description des tâches périodiques à effectuer
- **/home** : la racine des répertoires personnels des utilisateurs
- **/lib** [library] les bibliothèques et les modules du noyau, équivalent des DLL de Windows
- **/mnt** [mount] et **/media** : la racine des points de montage des systèmes de fichiers périphériques ou extérieurs (cd, clé-usb, ...).



L'organisation à la racine

- **/opt** [optional] : lieu d'installation d'applications supplémentaires (Chrome, Signal, OwnCloud, ...)
- **/proc** [processus] : contient des infos sur l'état du système et les différents processus en fonction.
- **/root** : répertoire personnel du super-utilisateur root
- **/sbin** : [system binaries] les exécutables pour l'administration du système (commandes de démarrage et d'arrêt du système, ...)
- **/tmp** [temporary] : les fichiers temporaires
- **/usr** : programmes accessibles à tout utilisateur; sa structure reproduit celle de la racine /
- **/var** [variable] : contient des fichiers dont on s'attend qu'ils grossissent en taille. (**var/crash**, **/var/mail**, **/var/log**, file d'impression dans **/var/spool/lpd**)



Systeme de fichiers : Les commandes associées



Commandes de base sur les fichiers et répertoires

- `ls, cat, cp, mv, rm, touch`
- `pwd, cd, mkdir, rmdir, rm -r`

Wooclap



1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
XZBBRK

 Activer les réponses par SMS



Commandes de base sur les fichiers

- **ls** *fichier ...* [**list**] affiche le contenu des répertoires (à un niveau) et les noms des fichiers passés en arguments, ou s'il n'y a pas d'arguments, tous les fichiers du répertoire courant (« . ») sauf ceux commençant par un point.
- **cat** *fichier ...* [**concatenate**] affiche le contenu des fichiers donnés en arguments
- **cp** *fichier1 fichier2* [**copy**] copie *fichier1* dans *fichier2*
- **mv** *fichier1 fichier2* [**move**] renomme *fichier1* en *fichier2*
- **rm** *fichier* [**remove**] détruit le fichier *fichier*



Commandes de base sur les répertoires

- **pwd** [**p**rint **w**orking **d**irectory] affiche le chemin absolu du répertoire courant.
- **cd** *répertoire* [**c**hange **d**irectory] change de répertoire courant. Sans argument, rapatrie dans le répertoire de connexion (votre home).
- **mkdir** *répertoire* [**m**ake **d**irectory] crée un répertoire.
- **rmdir** *répertoire* [**r**emove **d**irectory] détruit le répertoire s'il est vide et si ce n'est pas votre répertoire courant.
- **rm -r** *répertoire* [**r**emove] détruit le répertoire récursivement (même non vide).
- **cp -r** *répertoire* [**c**opy recursively]
- **mv -r** *répertoire* [**m**ove recursively]

Commandes de base sur les répertoires - Exemples

- Créer un répertoire vide et le supprimer

```
prompt> mkdir rep
```

```
prompt> rmdir rep
```

```
prompt> rmdir rep
```

```
rmdir: impossible de supprimer 'rep': Aucun fichier ou dossier de ce nom
```

- Créée un répertoire non vide et le supprimer :

```
prompt> mkdir rep ; touch rep1/toto
```

```
prompt> rmdir rep1
```

```
rmdir : 'rep1' : Le répertoire n'est pas vide.
```

```
prompt> rm -r rep1
```

- Copier un répertoire ou un fichier

```
prompt> cp -r repertoire repertoire1
```

```
prompt> cp fichier fichier1
```

Commandes de base sur les répertoires - Exemples

- Copier une arborescence

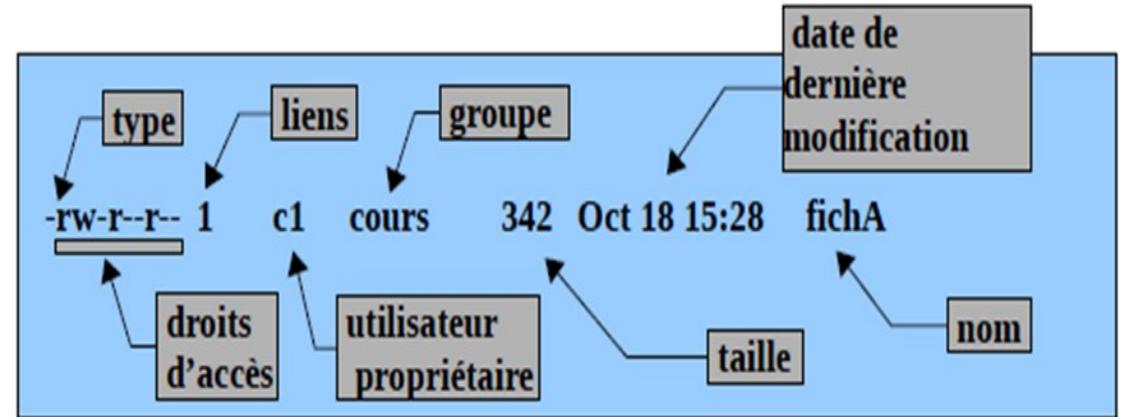
```
prompt> ls -R /tmp/r1
/tmp/r1 :
f1 f2 r11 r12
/tmp/r1/r11 :
f11
/tmp/r1/r12 :
f12
prompt> cp -r /tmp/r1 .
prompt> ls -R r1 r1 :
f1 f2 r11 r12
r1/r11 :
f11 r1/r12 :
f12
```

- Supprimer une arborescence : Attention il n'y a pas de demande de confirmation !

```
prompt> rm -rf r1
prompt> ls r1
ls : r1 : Aucun fichier ou répertoire de ce type.
```

La commande `ls -l`

`ls -l` (long) affiche de nombreuses informations sur le fichier :



- Le **type du fichier** :
 - 'd' : pour répertoire,
 - '-' pour fichier ordinaire,
 - 'b' pour périphérique bloc,
 - 'c' pour périphérique caractère,
 - 'l' lien symbolique,
 - 'p' tube nommé (IPC),
 - 's' socket locale (IPC).
- Le **nom de fichier** : Limité à 14 (ou 255) caractères parmi le jeu ASCII. Le système n'impose aucun format. On évite les caractères invisibles et les méta-caractères (*, ?, [et]).
- La **taille du fichier** : C'est son nombre d'octets. Elle sert à déterminer la fin du fichier. Il n'y a donc pas de marque de fin de fichier.
- **Droits d'accès** : Trois groupes d'autorisation :
 - l'utilisateur propriétaire,
 - les personnes appartenant au groupe propriétaire et
 - les autres.



La commande `ls -i`

Pour voir les numéros d'i-nodes par la commande :

```
prompt> ls -i
```

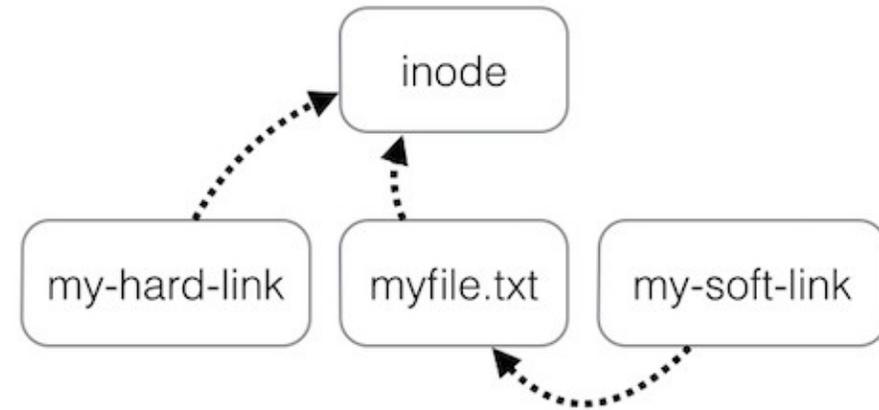
```
423    fichA
```

```
666    fichB
```

```
759    fichC
```

La commande `ln` [link]

- Lien dur « hard link » : un fichier qui pointe **vers le même inode** qu'un autre fichier.
- Lien symbolique « soft link » : un fichier qui pointe **vers un chemin** donné.



```
echo 'Hello, World!' > myfile.txt
```

```
ln myfile.txt my-hard-link % hard link
```

```
ls -i myfile.txt my-hard-link % ils ont les même inode
```

```
ln -s myfile.txt my-soft-link % lien symbolique
```

```
ls -i myfile.txt my-soft-link % ils n'ont pas les mêmes inode
```

```
rm myfile.txt % n'affecte pas le lien dur, alors que le lien soft est cassé : il renvoie vers un fichier qui n'existe plus
```



La commande `du`

Connaître la taille [disk usage] d'une arborescence et de chacun de ses sous-répertoires et fichiers

```
prompt> du .  
8 ./kde/Autostart  
8 ./kde  
4 ./rep  
56 .
```

Connaître le total (`-s`, `--summarize`) avec la taille exprimée en K, M et G (`-h`, `--human-readable`)

```
prompt> du -hs /home  
620 M /home
```



La commande `file`

`file fichier ...` affiche le type du fichier. À utiliser avant de visualiser le contenu d'un fichier pour éviter d'afficher un contenu binaire. ;-)

Exemple :

```
prompt> file /bin/ls /etc/passwd /usr/bin
```

```
bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),dynamically linked (uses shared libs),  
for GNU/Linux 2.6.15, stripped. /bin/ls: demand paged pure executable
```

```
/etc/passwd: ASCII text
```

```
/usr/bin: directory
```

La commande `grep`

`grep` recherche dans un ou plusieurs fichiers les lignes qui correspondent à un **motif** (expression régulière) :

`grep [options] motif chemin`

Avec :

options: les options possibles (voir man `grep`)

motif : le terme à rechercher, entre guillemets

chemin : le chemin du fichier (ou dossier) où faire la recherche

Rq : guillemets facultatifs, mais conseillés si le motif contient des caractères autres qu'alphanumériques.

Exemple :

```
prompt> grep c1 /etc/passwd # Afficher toutes les lignes du fichier /etc/passwd contenant la chaîne c1.  
c1:vs9Fi0TbjD6xg:208:2001:eleve 1:/usr/c1:/bin/ksh c10:vs9Fi9bjD6xg:209:2001:eleve  
10:/usr/c10:/bin/ksh  
prompt> grep "^m" /etc/passwd # Les lignes qui commencent par « m »  
prompt> grep "[0-9]\{10\}" * # recherche des suites de 10 chiffres dans tous les fichiers
```

<https://wodric.com/commande-grep/>

<https://www.geeksforgeeks.org/grep-command-in-unixlinux/>



Commandes d'affichage de fichiers

cat

head

less

more

pg : affiche le contenu du fichier passé en argument par pages de 23 lignes

pr : formate le texte du fichier passé en argument pour l'impression.

<https://www.geeksforgeeks.org/pr-command-in-linux/>



La commande `wc`

`wc` compte le nombre de lignes, de mots et de caractères contenus dans le fichier passé en argument,

Exemple:

```
prompt> wc fichier  
8 48 208 fichier
```

→ Le fichier « `fichier` » possède 8 lignes, 48 mots et 208 caractères.



La commande `sum`

`sum` calcule et affiche une **somme de contrôle** (checksum) pour un fichier (intégrité des fichiers).

Exemple :

```
prompt> sum fichier  
08860 1
```

La commande `diff`

`diff` affiche les lignes différentes devant être modifiées pour que les deux fichiers soient identiques.

Exemple :

```
prompt> diff toto toto1
3d2
< line3
5a5
> GNU is not UNIX
```

	toto	toto1
1	this is the original text	1 this is the original text
2	line2	2 line2
3	line3	3 line4
4	line4	4 happy hacking
5	happy hacking !	5 GNU is not UNIX

La commande `diff`

`diff` affiche les lignes différentes devant être modifiées pour que les deux fichiers soient identiques.

Exemple :

```
prompt> diff -u toto toto1
```

```
--- toto 2024-09-22 18:10:46.370960519 +0200
```

```
+++ toto1 2024-09-22 18:10:14.937189801 +0200
```

```
@@ -1,5 +1,5 @@
```

```
this is the original text
```

```
line2
```

```
-line3
```

```
line4
```

```
happy hacking !
```

```
+GNU is not UNIX
```

toto

```
1 this is the original text
2 line2
3 line3
4 line4
5 happy hacking !
```

toto1

```
1 this is the original text
2 line2
3 line4
4 happy hacking
5 GNU is not UNIX
```

La commande `cmp`

`cmp` compare octet par octet les deux fichiers passés en paramètres. Cette commande renvoie « 0 » si les fichiers sont identiques, « 1 » sinon.

Exemple :

```
prompt> cmp toto toto1  
toto toto1 sont différents: octet 41, ligne 3
```

```
prompt> cmp toto toto  
prompt>  
/*les fichiers sont identiques*/
```

	toto	toto1
1	this is the original text	this is the original text
2	line2	line2
3	line3	line4
4	line4	happy hacking
5	happy hacking !	GNU is not UNIX



La commande `touch`

`touch` met à jour la date de dernière modification du fichier. Si le fichier n'existe pas encore, il sera créé (et de taille nulle) sauf si l'option `-c` (no create) est utilisée.

Exemple :

```
prompt> ls -l .  
-rw-r--r-- 1 c1 cours 0 Oct 9 1991 toto  
prompt> touch toto  
prompt> ls -l toto  
-rw-r--r-- 1 c1 cours 0 Sep 23 2024 toto  
prompt> touch titi  
prompt> ls -l .  
-rw-r--r-- 1 c1 cours 0 Sep 23 2024 toto  
-rw-r--r-- 1 c1 cours 0 Sep 23 2024 titi
```



La commande `compress`

`compress` opère une compression visant à diminuer l'espace occupé par les différents fichiers référencés (algorithme de Lempel–Ziv–Welch). Chaque fichier est remplacé par un nouveau fichier dont la référence est obtenue en suffixant la référence d'origine avec l'extension `.Z`. Les caractéristiques du fichier sont conservées. Le fichier d'origine est supprimé.

`compress [options] liste_fichiers`

Exemples :

```
prompt> compress -v exemple.xls
```

```
exemple.xls : -- replaced with exemple.xls.Z Compression: 24.57%
```

```
le fichier exemple.xls est compressé et remplacé par le fichier exemple.xls.Z
```

```
prompt> compress -rv abc
```

```
comprime tous les fichiers contenus dans abc et ses sous répertoires de manière récursive
```



La commande `uncompress`

`uncompress` permet la **décompression** et la reconstruction d'une série de fichiers à partir de leurs formes compressées avec la commande `compress`.

`uncompress [options] liste_fichiers`

Exemple d'utilisation :

```
prompt> uncompress -v exemple.xls.Z
```

```
exemple.xls : 24.6% -- replaced with exemple.xls
```

le fichier `exemple.xls.Z` est décompressé et remplacé par le fichier `exemple.xls`

La commande tar

tar (tape archiver) permet de gérer des archives de fichiers. La **clé** définit les actions de la commande. Elle est constituée d'une suite de caractères définissant la **fonction** (crtux) et des **qualificatifs** de cette commande (Abfhmov).

tar [clé] liste_fichiers.tar [chemin]

Exemples d'utilisation :

prompt> **tar cvf archive_cible.tar /etc**

Place tous les fichiers du répertoire **/etc** dans le fichier **archive_cible.tar**

prompt> **tar xvf archive_cible.tar**

Extraction de **archive_cible.tar** dans le répertoire courant.

prompt> **tar tvf archive_cible.tar**

Liste les fichiers contenus dans **archive_cible.tar**



La commande **tar** – les fonctions (crtux)

- c** : création d'une nouvelle archive. Sur bande, l'écriture de l'archive a lieu en début de bande et non à la suite du dernier fichier ;
- r** : fonction de remplacement permettant d'écrire en fin d'archive les fichiers de références données ;
- t**: liste des références de fichiers dans l'archive sans restitution ;
- u** : les fichiers sont ajoutés en fin d'archive s'ils n'y figurent pas encore ou si la date de modification de la dernière version archivée est antérieure à la version du fichier sur le disque ;
- x**: fonction d'extraction de l'archive. Si la référence examinée est une référence de répertoire, son contenu est extrait de manière récursive. Si aucune référence de fichier n'est donnée, tous les fichiers de l'archive sont extraits.



La commande **tar** – les qualificatifs (Abfhmov)

- A** : les messages d'avertissement sont supprimés ;
 - f** : l'argument suivant est interprété comme une référence de fichier correspondant au nom de l'archive (au lieu d'une référence par défaut qui est en général celle d'un fichier spécial associé à un dérouleur de bande). Si cet argument est -, la commande lit sur l'entrée standard ou écrit sur la sortie standard ;
 - h** : les liens symboliques sont suivis (par défaut, ils ne le sont pas) ;
 - v** : option « verbeuse ».
- ... et les autres → **man tar**



La commande `find`

`find` parcourt récursivement l'arborescence en sélectionnant des fichiers selon des **critères de recherche**, et **exécute des actions** sur chaque fichier sélectionné.

`find répertoire_de_départ [critère_de_recherche] action_à_exécuter`

Exemple :

`$ find ~ -print`

parcoure toute l'arborescence à partir du home (~), sélectionne tous les fichiers (puisque'il n'y a aucun critère de recherche), et affiche le nom de chaque fichier trouvé.

La commande **find** – critère de recherche

-name modèle sélectionne uniquement les fichiers dont le nom correspond au modèle.

Attention ! Le modèle doit être interprété par la commande **find** et non par le shell, donc s'il contient des caractères spéciaux pour le shell (par exemple *****), ceux-ci doivent être échappés.

Mauvais exemple : `$ find /usr/c1 -name *.c -print`

Le shell remplace ***.c** par la liste des fichiers finissant par **.c** du répertoire **/usr/c1**, puis va chercher dans l'arborescence donnée ces noms de fichiers.

Cela reviendra à : `$ find /usr/c1 -name f1.c f2.c f3.c -print`

Problème : **-name** n'accepte qu'un seul argument !

Par contre dans : `$ find /usr/c1 -name '*.c -print`

C'est bien ***.c** qui sera passé en argument de l'option **-name** de la commande **find**. La recherche se fera donc bien sur les trois fichiers **f1.c**, **f2.c** et **f3.c**.



La commande **find** – critère de recherche

-perm *nombre_octal* sélectionne les fichiers dont les droits d'accès sont ceux indiqués par le nombre octal.

Exemple : Afficher tous les fichiers qui sont autorisés en lecture, écriture et exécution pour l'utilisateur propriétaire, les personnes du groupe propriétaire et tous les autres.

```
$ find /usr/c1 -perm 777 -print
```



La commande **find** – critère de recherche

-type *caractère* sélectionne les fichiers dont le type est celui indiqué par le caractère.

C'est-à-dire :

- **c** pour un fichier spécial en mode caractère
- **b** pour un fichier spécial en mode bloc
- **d** pour un répertoire
- **f** pour un fichier normal
- **l** pour un lien symbolique

Exemple : afficher tous les répertoires et sous-répertoires de */usr/c1*.

```
$ find /usr/c1 -type d -print
```



La commande **find** – critère de recherche

-links *nombre_décimal* sélectionne les fichiers dont le nombre de liens est donné par le nombre décimal. Si le nombre est précédé d'un + (d'un -) cela signifie supérieur (inférieur) à ce nombre.

Exemple : afficher tous les fichiers qui ont plus de deux liens.

```
$ find /usr/c1 -links +2 -print
```

-user *n[ou]m_utilisateur* sélectionne les fichiers dont l'utilisateur propriétaire est *nom_utilisateur* ou dont le numéro d'utilisateur (UID) est *num_utilisateur*.

Exemple : Afficher tous les fichiers spéciaux appartenant à l'utilisateur **c1**

```
$ find /dev -user c1 -print
```



La commande **find** – critère de recherche

-inum *nombre_décimal* sélectionne les fichiers ayant pour numéro d'i-noeud *nombre_décimal*.

-newer *fichier* sélectionne les fichiers qui sont plus récents que celui passé en argument.

-atime *nombre_décimal* sélectionne les fichiers qui ont été accédés dans les *nombre_décimal* derniers jours.

-mtime *nombre_décimal* sélectionne les fichiers qui ont été modifiés dans les *nombre_décimal* derniers jours.

-size *nombre_décimal*[*c*] sélectionne les fichiers dont la taille est de *nombre_décimal* blocs. Si on post-fixe le *nombre_décimal* par le caractère *c*, alors la taille sera donnée en nombre de caractères.

La commande `find` – critère de recherche

Plusieurs critères peuvent être groupés (combinés) par les opérateurs (`et`).

Attention : pour le shell, ce sont des caractères spéciaux, ils doivent être échappés.

- Le **ET** logique est implicite : on met plusieurs critères à la suite et `find` sélectionne les fichiers qui répondent à tous les critères.

Exemple : afficher les fichiers se terminant par `.c` **ET** modifiés dans les 3 derniers jours.

```
$ find /usr/c1 \( -name '*.c' -mtime -3 \) -print
```

- Le **OU** logique est représenté par l'opérateur `-o`
Exemple : affiche tous les fichiers se terminant par `.txt` **OU** `.doc`.

```
$ find /usr/c1 \( -name '*.txt' -o -name '*.doc' \) -print
```

- Le **NON** logique est l'opérateur `!`
Exemple : afficher tous les fichiers **n'**appartenant **PAS** à `c1`, mais qui se trouvent dans son arborescence.

```
$ find /usr/c1 ! -user c1 -print
```

La commande `find` – les actions possibles

- `-print` affiche le nom des fichiers sélectionnés sur la sortie standard.

Exemple : afficher toute l'arborescence de `c1`.

```
$ find /usr/c1 -print
```

- `-exec commande \;` exécute `commande` sur tous les fichiers sélectionnés. Dans la commande shell, « `{}` » sera remplacé par le nom du fichier sélectionné.

Exemple : rechercher tous les fichiers se terminant par l'extension `.o` dans l'arborescence `/usr/c1` et les détruit, puis recherche les fichiers se terminant par l'extension `.o` pour vérifier

```
$ find /usr/c1 -name '*.o' -exec rm {} \;
```

```
$ find /usr/c1 -name '*.o' -print
```

Exemple : rechercher dans les répertoire `/dev` et `/home`, tous les fichiers appartenant à `hugo`, en format long.

```
$ find /dev /home -user hugo -exec ls -l {} \;
```



TD3 :

L'arborescence du système de
fichiers d'UNIX



Recap

- En Linux, tout est **fichier**, même les commandes du terminal qui sont stockées dans **/bin**
- En Linux, l'utilisateur **root** a tous les droits.
- **cd** [change directory], **ls** [list], **cat** [concatenate], **cp** [copy], **rm** [remove]

cd /bin

ls

cat ls

cp ls machintruc

sudo cp ls machintruc

sudo rm ls

ls

machintruc