Linux
Le système de fichiers

JFA - 110

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

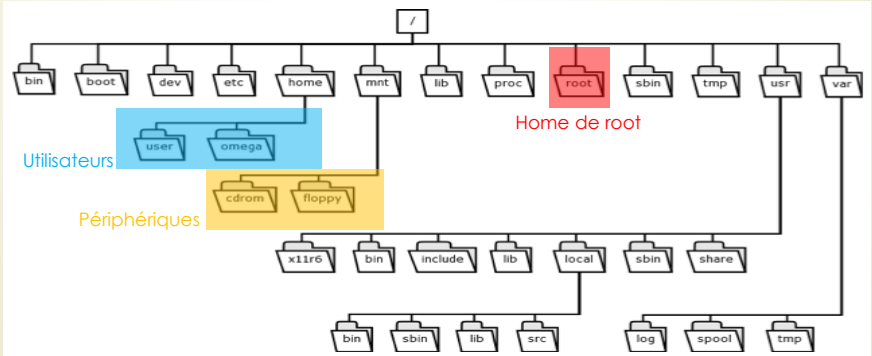
Le système de fichiers, ses commandes associées:

Un système multiutilisateur se doit de ranger ses fichiers correctement pour éviter un désastre.

Sous LINUX, ils sont organisés en une structure hiérarchique appelée arborescence, et rangés dans des répertoires.

Les différents répertoires qui contiennent les fichiers du système et vos propres fichiers, sont organisés en une structure arborescente qui part de la racine root (racine en français) et se déploie sous forme de ramifications multiples. Un répertoire particulier home (maison en français) est réservé par le système pour recevoir les fichiers des différents utilisateurs.

JFA - 111



http://sil.lmdsio.fr/lessons/linux

L'organisation des fichiers arborescente :

JFA - 112

- **/etc** : les fichiers de configuration,
  - **/etc/rc.d** : le répertoire des scripts associés aux différents niveaux de fonctionnement,
  - **/etc/rc.d/rc3.d** : scripts exécutés au démarrage du niveau 3 de fonctionnement du système,
- **/mnt** : répertoire pour le montage de CD-ROM, disques, partages, …,
- **/proc** : répertoire pour la mémorisation des processus,
- **/root** : le répertoire de connexion de l'administrateur,
- **/bin** : les binaires nécessaires en mode mono-utilisateur,
- **/sbin** : les commandes de démarrage et d'arrêt du système,
- **/tmp** : le répertoire temporaire pour la création de fichiers temporaires,
- **/dev** : le répertoire des fichiers spéciaux,
- **/var** : le répertoire des fichiers des services,
  - **/var/mail** : les boîtes aux lettres des utilisateurs,
  - **/var/spool/cron/** : le répertoire des fichiers de données du service *cron* pour la commande *crontab*,

L'organisation des fichiers arborescente est une organisation récente :

JFA - 113

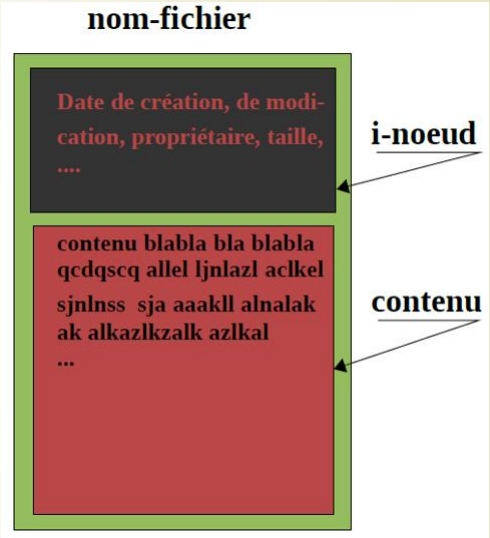
- **/var/spool/lpd** : fichiers de données du service impression,
  - **/var/run/** : fichiers contenant les PID des services actifs et des utilisateurs connectés,
  - **/var/log** : les fichiers de « *log* »,
  - **/var/crash** : les fichiers image mémoire en cas de crash du système,
- **/usr/** : répertoire des commandes du système,
  - **/usr/bin** : commandes de base du système,
  - **/usr/lib** : les bibliothèques de sous-programmes,
  - **/usr/man** : les manuels de référence,
  - **/usr/src** : les fichiers source du noyau,
  - **/usr/src/linux** : répertoire pour recompiler le noyau,
  - **/usr/doc** : la documentation des paquetages,
- **/home :** répertoire des répertoires de connexion des utilisateurs.

02/10/2023

Les fichiers ordinaires :

*Dans Linux et Unix, **tout est fichier**.*

- *Les répertoires sont des fichiers,*
- *les fichiers sont des fichiers,*
- *les périphériques sont des fichiers. Même si les périphériques sont souvent appelés nœuds, ce sont quand même des fichiers.*



JFA - 114

Pour le système, les données d'un fichier forment une suite non structurée d'octets (byte stream). Un fichier texte, par exemple, est une suite de codes ASCII, les lignes étant séparées par le code ASCII 'nouvelle ligne' (012 octal). Il n'existe aucune notion d'organisation telle que séquentiel indexé, etc...

Un certain nombre de caractéristiques sont associées à un fichier : la date de sa création, celle de la dernière modification, le propriétaire, la taille, etc... Ces caractéristiques sont regroupées dans un descripteur de fichier, appelé nœud d'index (i-node ou index-node).

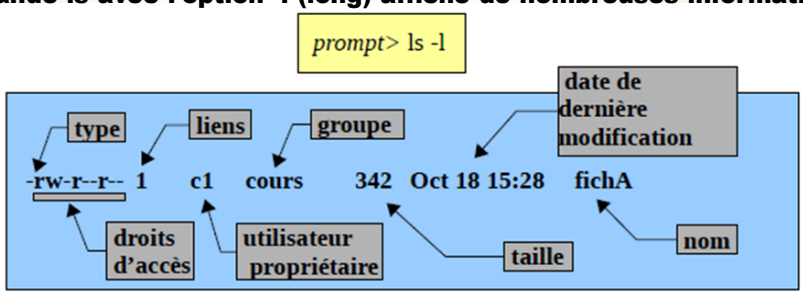
Les fichiers ordinaires : commandes associées

- **ls** *fichier* ... [**l**i**s**t] affiche le contenu des répertoires (à un niveau) et les noms des fichiers passés en argument, c'est_à-dire *fichier* ..., ou s'il n'y a pas d'argument, tous les fichiers du répertoire courant sauf ceux commençant par un point.
- **cat** *fichier* ... [con**cat**enate] affiche le contenu des fichiers *fichier* ...
- **cp** *fichier1 fichier2* [**c**o**p**y] copie *fichier1* dans *fichier2*
- **mv** *fichier1 fichier2* [**m**o**v**e] renomme *fichier1* en *fichier2*
- **rm** *fichier* [**r**e**m**ove] détruit le fichier *fichier* ; irréversible.

JFA - 115

Note : **Toutes les commandes UNIX qui manipulent les fichiers acceptent toutes les formes (absolu, relatif ...) de chemins de noms de fichier.**

La commande ls avec l'option -l (long) affiche de nombreuses informations sur le fichier :



Les fichiers ordinaires : commandes associées

JFA - 116

- ***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.

Les fichiers ordinaires : droits d'accès

Pour chaque groupe 3 caractères indiquent les autorisations :

- ***r***, ***w*** et ***x***.

JFA - 117

- **R** ***lecture*** *:* ***Autorisation de lire le contenu*** (cat, pg ...) ; ce droit est nécessaire pour faire une copie du fichier. ***ls*** permet de lister le contenu du fichier (attention pour *ls -l* il faut aussi le droit *x*).
- **w** ***écriture*** *:* ***Autorisation de modifier le contenu*** ; On peut écrire dans un fichier avec un éditeur ou une commande (cat). Il faut ce droit pour le fichier destination d'une copie, d'un déplacement ou d'un lien, si le fichier existe déjà. Pour un répertoire on peut créer et détruire des fichiers et des répertoires. Lors d'une destruction, seule cette permission prévaut. Elle permettra de détruire un fichier, même si on n'a aucun droit dessus. La destruction d'un fichier revient à modifier le répertoire, pas le fichier !
- **x** ***exécution/traverse*** ***(répertoire)***
- **-** à la place de ***r***, ***w*** ou ***x*** signifie ***non-autorisé***.

Commandes associées

- **ls -F :** affiche les noms de fichiers suivis du caractère '/' s'il s'agit d'un répertoire, et du caractère '*' s'il s'agit d'un fichier ayant la permission d'exécution.
- **ls -C :** affiche les noms de fichiers par colonnes.
- **ls -l :** affiche les noms de fichiers par ligne avec toutes les informations.

Changer les droits d'un fichier ou d'un répertoire

- **chmod : Modifie les droits d'accès sur un fichier ou un répertoire**

  chmod code fichier

  chmod code repertoire

Exemple d'utilisation :

prompt> chmod 777 fichier

JFA - 118

Personne concernée	
propriétaire	u
groupe	g
autres	o
tous	a
Action	
ajouter	+
enlever	-
initialiser	=
Accès autorisés en	
lecture	r
écriture	w
exécution/traverse	x

Changer les droits d'un fichier ou d'un répertoire

Exemple :

```
prompt> ls -l fichA
-rw-rw-rw- 1 c1 cours 342 Oct 18 15:28 fichA
 P  G  A
prompt> chmod go-w fichA; ls -l fichA
-rw-r--r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod u+x fichA; ls -l fichA
-rwxr--r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod g-r fichA; ls -l fichA
-rwx---r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod ug+rw fichA; ls -l fichA
-rwxrw-r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod g-rw fichA; ls -l fichA
-rwx---r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
prompt> chmod ug=rw fichA; ls -l fichA
-rw-rw-r-- 1 c1 cours 342 Oct 18 15:28 fichA
prompt>
```

JFA - 119

02/10/2023

Changer les droits d'un fichier ou d'un répertoire

Le code peut également être donné en mode octal.

Donc

775 (111 111 101) représente rwx rwx r-x

640 (110 100 000) représente rw- r-- ---

lettres	binaire	octal
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5
rw-	110	6
rwx	111	7

JFA -120

Exemple :

```
prompt> ls -l toto
-rw-r--r-- 1 c1 cours 342 Oct 18 15:28 toto
prompt>
prompt> chmod 760 toto; ls -l toto
-rwxrw---- 1 c1 cours 342 Oct 18 15:28 toto
prompt>
```


Changer les droits d'un fichier ou d'un répertoire

- **umask : Modifie le masque des droits de création de fichier**

  umask [masque]

Lorsqu'un programme crée un fichier, il spécifie les droits d'accès. Parmi ceux-ci, certains sont accordés, d'autres refusés, en fonction du masque. Sans argument, donne la valeur actuelle du masque. C'est le même pour les fichiers et les répertoires !

Exemple d'utilisation : **prompt> umask 022**

JFA - 121

Pour les répertoires :

droits demandés :	rwx	rwx	rwx	ou encore 777
- masque :	---	-w-	rwx	ou encore 027
droits accordés :	rwx	r-x	---	ou encore 750

Pour les fichiers :

droits demandés :	rw-	rw-	rw-	ou encore 666
- masque :	---	-w-	-w-	ou encore 022
droits accordés :	rw-	r--	r--	ou encore 644

Changer le masque pour un fichier ou un répertoire

Exemple :

```
prompt> umask
000
prompt> mkdir foo
prompt> ls -ld foo
drwxrwxrwx 2 c1 cours 512 Jul 3 16:39 foo
prompt> rmdir foo
prompt> umask 022
prompt> mkdir foo
prompt> ls -ld foo
drwxr-xr-x 2 c1 cours 512 Jul 3 16:41 foo
prompt> rmdir foo
prompt> umask 077
prompt> mkdir foo
prompt> ls -ld foo
drwx------ 2 c1 cours 512 Jul 3 16:42 foo
prompt>
```

JFA - 122

Recherche dans les fichiers

- **grep : recherche dans un ou plusieurs fichiers les lignes qui correspondent à un motif (chaine de caractères)**

grep [options] motif chemin

Avec :

options: les options possibles (voir man grep)

recherche : le terme à rechercher entre guillemets

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

JFA - 123

Les guillemets autour du terme à rechercher ne sont pas obligatoire. Néanmoins je les conseils dès lors qu'il y a des caractères autres qu'alphanumériques dans votre recherche..

Exemple d'utilisation :

prompt> grep "^m" /etc/passwd

prompt> grep "192" /etc/hosts

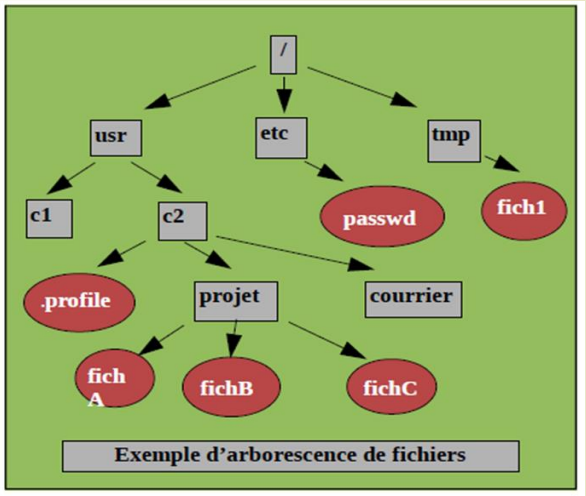
grep "[0-9]\{10\}" * # recherche des suites de 10 chiffres dans tous les fichiers

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

Les répertoires :

Un ***répertoire*** ou ***catalogue*** (***directory***) est un **fichier** qui contient une liste de noms de fichiers, parmi lesquels on peut trouver des sous-répertoires, et ainsi de suite (***arborescence logique***).

Dans l'exemple ci-contre, les répertoires sont représentés par des rectangles et les fichiers par des cercles.



Exemple d'arborescence de fichiers

JFA - 124

- ***Commandes associées***
- **mkdir** *répertoire :* [**mak**e **dir**ectory] : crée un répertoire.
- **rmdir** *répertoire :* [**rem**ove **dir**ectory] : détruit le répertoire s'il est vide et si ce n'est pas votre répertoire courant.
- **cd** *répertoire :* [**c**hange **d**irectory] : change de répertoire courant. Sans argument rapatrie dans le répertoire de connexion.
- **pwd** **:** [**p**rint **w**orking **d**irectory] : affiche le chemin absolu du répertoire courant.

02/10/2023

Les répertoires :

Un répertoire est un fichier ordinaire dans le sens où il possède un index et des données sous forme de suite d'octets. Seul le système peut écrire dans un répertoire.

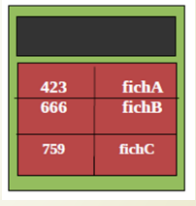
répertoire = table de liens (i-nombre, chaîne de caractère).

i-nombre = numéro d'index (i-node) de la structure décrivant le fichier.

JFA - 125

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

```
prompt> ls -i
423    fichA
666    fichB
759    fichC
prompt>
```



Exemple de Fichier-répertoire "projet"

02/10/2023

Manipulations de répertoires :

- **mkdir : Créer un répertoire**

Exemple d'utilisation :

```
prompt> mkdir rep
```

- **rmdir : Supprime un répertoire**

Exemple d'utilisation :

JFA - 126

```
prompt> rmdir rep
```

on ne peut pas supprimer un répertoire vide

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

- **cp : Copier un répertoire, un fichier**

Exemple d'utilisation :

```
prompt> cp rep rep1
prompt> cp fichier fichier1
```


Manipulations de répertoires :

- **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
```

JFA - 127

- **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.
```

Manipulations de répertoires :

- **Connaître la taille d'une arborescence et de chacun de ses sous-répertoires**

prompt> du .

8 ./.kde/Autostart

8 ./.kde

4 ./rep

56 .

JFA - 128

- **Connaître le total (-s) avec la taille exprimée en K, M et G (-h)**

prompt> du -hs /home

620 M /home

Les chemins :

- ***Chemin absolu*** : Un chemin absolu se base sur la racine de l'arborescence et commence par / ou ~,

  ex. : `/home/utilisateur/<dossier>/<fichier>`.

  On accède au fichier à partir de la racine et en indiquant tous les sous-répertoires rencontrés jusqu'au fichier.

- ***Répertoire courant*** : A chaque processus est associé un répertoire, appelé répertoire courant ou de travail (défaut).

JFA - 129

- ***Raccourcis Shell*** : Tilde **~,** utilisé en premier nom de répertoire, remplace le chemin absolu par son répertoire personnel, soit /home/utilisateur, mais cette fonctionnalité est propre au shell, et pas au système de fichier.

- ***Commandes associées***

  ln *fichier1 fichier2 :* [**l**i**n**k] :établit un nouveau lien sur le fichier *fichier1*.

Gestion des utilisateurs :

- **groupadd nom_groupe : Ajoute un groupe d'utilisateurs.**

Exemple d'utilisation :

```
prompt> groupadd but1a
```

- **groupdel nom_groupe : Détruit un groupe d'utilisateurs.**

Exemple d'utilisation :

```
prompt> groupdel but1a
```

JFA - 130

- **useradd nom_utilisateur : Ajoute un utilisateur**

Exemple d'utilisation :

```
prompt> useradd jfa
```

- **usermod nom_utilisateur : Modifie les paramètres d'un compte utilisateur**

Exemple d'utilisation :

```
prompt> usermod -c "This is the best user" jfa
prompt> usermod -d /home/jfahome jfa
prompt> usermod -e 2020-05-29 jfa
```

02/10/2023

https://www.geeksforgeeks.org/usermod-command-in-linux-with-examples/

Gestion des utilisateurs :

- **id nom_utilisateur : Affiche les informations d'identité d'un utilisateur.**

Exemple d'utilisation :

```
prompt> id jfa
```

- **groupdel nom_groupe : Détruit un groupe d'utilisateurs.**

Exemple d'utilisation :

```
prompt> groupdel but1a
```

JFA - 131

- **useradd nom_utilisateur : Ajoute un utilisateur**

Exemple d'utilisation :

```
prompt> useradd jfa
```

- **usermod nom_utilisateur : Modifie les paramètres d'un compte utilisateur**

Exemple d'utilisation :

```
prompt> usermod -c "This is the best user" jfa
prompt> usermod -d /home/jfahome jfa
prompt> usermod -e 2020-05-29 jfa
```

02/10/2023

https://www.geeksforgeeks.org/usermod-command-in-linux-with-examples/

Les chemins :

- ***Chemin relatif*** : c'est relatif au répertoire courant où se trouve l'utilisateur. Un chemin qui commence par autre chose que / ou ~ est un chemin relatif. Sous Linux on se trouve par défaut dans son répertoire personnel qui est /home/<nom d'utilisateur>. Dans un terminal on peut naviguer d'un répertoire à l'autre avec la commande **cd**.
- On peut aussi utiliser ce type de chemin pour indiquer où se trouvent les ressources les unes par rapport aux autres, indépendamment de la racine du système, par ex. pour que les fichiers d'un site web puissent se retrouver les uns les autres.
- On peut ne spécifier qu'une partie du chemin d'accès, pour que ce chemin soit interprété à partir du répertoire courant.

JFA - I32

« . » : répertoire courant

« .. » : répertoire parent

Donc on peut désigner un fichier de 2 façons :

nom relatif ou **nom absolu**.

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

- ***cd .*** : avec la commande **cd** on se déplace dans le répertoire courant, donc on reste où on est !
- ***cd ..*** : avec la commande **cd** on se déplace dans le répertoire parent, donc le répertoire juste au dessus de là où on est !
- ***cd ~*** : avec la commande **cd** on se déplace dans le répertoire Home, donc le répertoire de login de l'utilisateur !

02/10/2023

Gestion des liens entre fichiers

Un fichier (entête plus contenu) peut avoir plusieurs noms ; dans tous les cas, le système identifie un fichier non pas par un nom, mais par son numéro de i-noeud.

Un répertoire est un fichier dont le contenu contient un ensemble de couples **(nom, i-noeud)** ; **chaque nom est un lien vers un i-noeud donné**.

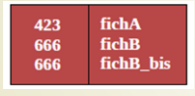
❑ ***Les liens "hard"***

Création de deux ou plusieurs noms vers un **i-noeud unique** au moyen de la commande :

JFA - I33

ln fichB fichBbis

Cette commande n'est utilisable qu'à l'intérieur d'un même système de fichiers (arborescence).



prompt> **ls -i**
423 fichA
666 fichB
666 fichBbis
prompt>

prompt> **ls -ali**
389 drwxr-xr-x 2 c1 cours 342 Oct 08 15:27 .
125 drwxr-xr-x 5 c1 cours 342 Oct 08 15:27 ..
423 -rw-r--r-- 1 c1 cours 342 Oct 18 15:28 fichA
666 -rwxr-xr-x 2 c1 cours 106 Oct 10 10:53 fichB
666 -rwxr-xr-x 2 c1 cours 106 Oct 10 10:53 fichBbis
prompt>

02/10/2023

Les liens "Symboliques"

Ces liens, différents des précédents, permettent de lier plusieurs noms à un même fichier sans rattacher ces derniers au i-noeud correspondant.

Pour cela on utilise la commande **:**

```
ln avec l'option -s
```

Cette commande est utilisable dans un système de fichiers (arborescence) ou entre des systèmes de fichiers différents.

La commande : `prompt> ln -s foo bar`

JFA - 134

Crée le lien symbolique bar qui pointe sur le fichier foo.

```
prompt> ls -i bar foo
22195 bar
22192 foo
```

On a des i-noeuds différents. Par contre si l'on utilise la commande "ls -l" on voit apparaître le lien :

```
prompt> ls -l bar foo
lrwxrwxrwx 1 root root 3 Aug 5 16:51 bar->foo
-rw-r--r-- 1 root root 12 Aug 5 16:50 foo
prompt>
```

02/10/2023

Les droits d'accès de la cible ("foo") via "bar" sont ceux de la cible.

Génération de noms de fichiers

Certains caractères spéciaux sont interprétés par le shell, et permettent de décrire les noms de fichiers. Ce sont *des méta-caractères* (c'est-à-dire des caractères utilisés pour décrire d'autres caractères) :

- **Le caractère '*'** signifie n'importe quelle chaîne de caractères.
- **Le caractère '?'** signifie n'importe quel caractère.
- **Les crochets '[ ]'** signifient un caractère appartenant à un ensemble de valeurs décrites dans les crochets.
- **Le caractère '-'** utilisé avec les crochets permet de définir un intervalle, plutôt qu'un ensemble de valeurs.
- **Le caractère '!' ou '^'** utilisé entre crochets en première position, signifie tout caractère excepté ceux spécifiés entre crochets.

JFA - 135

Un ***nom générique*** est un mot qui contient un ou plusieurs caractères spéciaux. Il permet de désigner un ensemble d'objets.

02/10/2023

JFA - 136

Exemples avec

- **f*** Tous les fichiers dont le nom commence par 'f'.
- **f?** Tous les fichiers dont le nom commence par 'f', suivi d'un seul caractère quelconque.
- **f[12xy]** Tous les fichiers dont le nom commence par 'f', suivi d'un caractère à choisir parmi '1, '2', 'x' ou 'y'.
- **f[a-z]** Tous les fichiers dont le nom 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** Tous les fichiers dont l'extension est .c
- **?.c** Tous les fichiers dont le nom est formé d'un caractère quelconque, suivi de '.c'
- **??** Tous les fichiers dont le nom est formé de deux caractères.
- ***.[A-Za-z]** Tous les fichiers dont le nom se termine par un '.' suivi d'une seule lettre majuscule ou minuscule.
- ***.[ch0-9]** Tous les fichiers dont le nom se termine par un '.' suivi d'un seul caractère à choisir parmi 'c', 'h', ou un chiffre entre '0' et '9'.
- **[!f]*** Tous les fichiers dont le nom ne commence pas par 'f'
- ***[!0-9]** Tous les fichiers dont le nom ne se termine pas par un chiffre.

JFA - 137

Exemples avec les classes :

Pour définir les ensembles il est préférable d'utiliser la norme POSIX avec les noms de classe suivant :

- [:upper:] pour les majuscules
- [:lower:] pour les minuscules
- [:digit:] pour les chiffres de 0 à 9
- [:alnum:] pour les caractères alphanumériques

Ces noms s'utilisent de la façon suivante :

```
ls ./[[:upper:]]*
```

pour, par exemple, tous les noms de fichiers qui commencent par une lettre majuscule.

Substitution par le shell des méta-caractères :

Lorsque le Shell trouve des métacaractères sur une ligne, il substitue l'expression entière, par exemple :

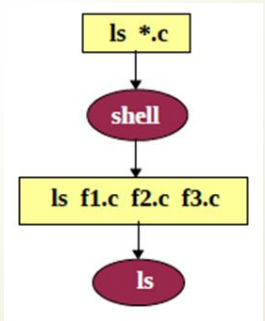
```
f* -> f1 f36 foo fz
```

Chaque ligne est donc analysée deux fois :

1. **Par le Shell** : celui-ci lit la ligne « ls -l f*.c » . Il enlève de cette ligne « f*.c » et le remplace par « f1.c f2.c f15.c ». Le résultat est donc la ligne « ls -l f1.c f2.c f15.c ».
2. **Par la commande** : lorsque celle-ci est exécutée, elle reçoit la ligne « ls -l f1.c f2.c f15.c », elle va donc travailler sur les 3 fichiers.

JFA - 138

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



JFA -139

DUT Informatique – Semestre 1

Ressource R1.04

Responsable : Jean-François ANNE

Les redirections d'entrées - sorties :

Tout processus effectue ses entrées/sorties via des « descripteurs de fichiers ». Ils sont numérotés à partir de 0.

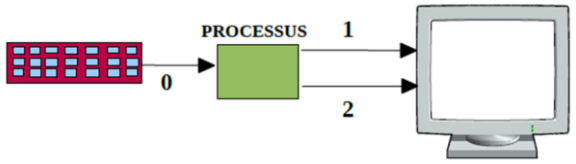
Par convention une commande ou un programme utilise :

- **le descripteur 0 (entrée standard) pour lire des données,**
- **le descripteur 1 (sortie standard) pour envoyer ses données en sortie,**
- **le descripteur 2 (sortie d'erreur) pour envoyer ses messages d'erreurs.**

JFA - 140

Chaque descripteur peut être associé :

- à un fichier,
- à un périphérique (terminal par exemple),
- au descripteur d'un autre processus.



La configuration de ces descripteurs, c'est-à-dire leur association à un fichier ou à un périphérique, fait partie de l'environnement d'un processus.

Note : La commande elle-même ne connaît pas la destination finale des données qu'elle écrit. Elle se contente d'écrire sur un descripteur. C'est l'interpréteur de commandes qui met en place un environnement spécifique avant d'appeler la commande. Les demandes de redirection sont traitées par l'interpréteur de commandes avant même le lancement des commandes.

La redirection de la sortie standard :

Pour rediriger la sortie standard vers un fichier, vous devez utiliser la syntaxe suivante :

```
commande > fichier
```

Si le fichier n'existe pas, il est créé par le Shell. S'il existe déjà, le Shell détruit son contenu pour le remplacer par la sortie de la commande.

Exemple :

JFA - 141

```
prompt> who > who.t
prompt>
```

Cette ligne ne produit aucune sortie sur le terminal. Toutes les informations de la commande ***who*** sont écrites dans le fichier ***who.t***

```
prompt> cat who.t
c1 tty1 Apr 25 02:44
c2 tty3 Apr 25 02:55
c3 tty6 Apr 25 02:33
prompt>
```

Le fichier ***who.t*** contient la sortie de la commande précédente ***who***, c'est-à-dire la liste des personnes connectées au système à l'instant où cette commande a été exécutée.

La redirection de la sortie standard :

Il est possible de rediriger plus d'une commande dans un fichier, il suffit de délimiter ces commandes par des parenthèses et séparées par des points virgules,

```
prompt> (date; who) > who.x
prompt>
```

Les sorties ***date*** et ***who*** seront redirigées dans le même fichier ***who.x***

Si on omet les parenthèses, le Shell exécute ***date***, imprime le résultat sur le terminal et ensuite lance ***who***, en redirigeant le résultat sur le fichier ***who.x.***

JFA - 142

La redirection de la sortie standard :

Pour rediriger la sortie standard dans un fichier, vous pouvez également utiliser la syntaxe suivante :

```
commande >> fichier
```

Dans ce cas, la sortie de la commande est rajoutée à la fin de fichier. Comme précédemment, si fichier n'existe pas, le Shell le crée.

JFA - 143

```
prompt> date > date.t ; cat date.t
Sun Jul 31 10:45:21 MET 1996
prompt> date >> date.t ; cat date.t
Sun Jul 31 10:45:21 MET 1996
Sun Jul 31 10:45:22 MET 1996
prompt>
```

La redirection de l' entrée standard :

Le Shell permet aussi de rediriger l'entrée standard d'un processus (clavier), en utilisant la syntaxe suivante :

```
commande < fichier
```

Dans ce cas, la commande lit ses données dans un *fichier.*

```
prompt> cat < who.x
```

JFA - 144

Cette commande affiche le contenu du fichier *who.x*.

Syntaxe générale :

- **n < fichier** redirige en lecture le descripteur *n* sur ***fichier***. Si ***n*** n'est pas précisé, ce sera 0 (l'entrée standard). Si ***fichier*** n'existe pas le shell renvoie une erreur.
- **n > fichier** redirige en écriture le descripteur ***n*** sur ***fichier***. Si ***n*** n'est pas précisé, ce sera 1 (la sortie standard). Si ***fichier*** existe il sera écrasé, sinon créé.
- **n >> fichier** redirige en écriture le descripteur ***n*** à la fin de ***fichier***, sans détruire les données préalablement contenues par ce fichier. Si ***n*** n'est pas précisé, ce sera 1 (la sortie standard). Si ***fichier*** n'existe pas il sera créé.

JFA - 145

- **n < &m** duplique le descripteur ***n*** sur le descripteur ***m*** en lecture, ainsi ***n*** et ***m*** seront dirigés vers le même fichier, ou le même périphérique.
- **n > &m** duplique le descripteur ***n*** sur le descripteur ***m*** en écriture.
- **n < &-** ferme le descripteur ***n*** en lecture.
- **n > &-** ferme le descripteur ***n*** en écriture.

Syntaxe générale :

Exemple :

```
prompt> ls > f1 2> f2
```

Le listing du répertoire actuel est envoyé vers le fichier f1, qui si il n'existe pas est créé. Si il a une erreur à l'exécution du ls, cette erreur est envoyée dans le fichier f2, qui s'il n'existe pas est créé.

```
prompt> ls fich-inexistant >> f1 2>&1
```

Les caractéristiques du fichier-inexistant (si il existe) sont ajoutées à la fin du fichier f1. les erreurs sont dirigées vers le descripteur 1 qui est redirigé vers le fichier, donc les erreurs sont écrites à la fin du fichier f1,

JFA - 146

```
prompt> ls fich-inexistant 1>> f1 2>>&1
```

Les caractéristiques du fichier-inexistant 1 (si il existe) sont affichées sur le descripteur 1 qui lui est redirigé vers le fichier f1, ajoutées à la fin. les erreurs sont dirigées vers le descripteur 1 qui est redirigé vers le fichier, donc les erreurs sont écrites à la fin du fichier f1,

```
echo 1234567890 > File     # Write string to "File".
exec 3<> File              # Open "File" and assign file desc. 3 to it.
read -n 4 <&3              # Read only 4 characters.
echo -n . >&3              # Write a decimal point there.
exec 3>&-                  # Close file desc. 3.
cat File                   # ==> 1234.67890
1234.67890
```


Double redirection :

Il est possible de rediriger à la fois l'entrée et la sortie.

Exemple :

```
prompt> wc < /etc/passwd > tmp
prompt> cat tmp
20 21 752
prompt>
```

- ✓ **Les redirections d'entrées/sorties standards sont effectuées par le Shell d'une façon totalement indépendante du contexte. Les caractères spéciaux '>' et '<' peuvent être situés n'importe où sur une ligne de commande (précéder ou suivre la commande).**

JFA - 147

Les deux lignes suivantes produisent le même résultat :

```
prompt> who > tmp; grep 'c[12]' < tmp
c1 tty4 Jul 31 09:46
c2 tty2 Jul 31 09:17

prompt> > tmp who; < tmp grep 'c[12]'
c1 tty4 Jul 31 09:46
c2 tty2 Jul 31 09:17
prompt>
```

Double redirection :

- ✓ **Les indications de redirections des entrées/sorties sont traitées par le Shell et ne sont pas passées à la commande. Le Shell appelle la commande après ces modifications.**
- ✓ **Un processus ne possède qu'une seule entrée, qu'une seule sortie et une seule sortie d'erreur. Par conséquent chaque descripteur ne peut être redirigé qu'une seule fois par commande !**

Exemple :

JFA - 148

```
prompt> cmd > fichier 1 > fichier 2
```

Cette ligne de commande n'a pas de sens, et *cmd verra sa sortie standard redirigée* uniquement vers *fichier2, après avoir* détruit le contenu de *fichier1*.

Attention :

Le Shell traite les séparateurs avant les redirections ; par conséquent il y a une différence importante entre les deux commandes suivantes :

Exemple :

```
prompt> cmd  2>  fichier

prompt> cmd  2  >  fichier
```

JFA - 149

- ✓ **Dans le premier cas, on redirige la sortie d'erreur de la commande cmd vers fichier.**
- ✓ **Alors que dans le second cas, on redirige la sortie standard de la commande cmd vers fichier et 2 sera considéré comme un argument de cmd (dû à l'espace entre le 2 et le >) !**

Poubelle :

On peut rediriger la sortie d'erreur vers la « poubelle » (/dev/null); dans ce cas les messages d'erreur seront perdus :

Exemple :

```
prompt> cat fichier-inexistant 2>/dev/null
prompt>
```

JFA - 150

Communication entre processus :

Les utilisateurs disposent d'un mécanisme leur permettant de lancer un certain nombre de processus de façon concurrente (en quelque sorte parallèlement) et communiquant entre eux par l'intermédiaire de tubes (pipes), le système assurant la synchronisation de l'ensemble des processus ainsi lancés.

Principe de fonctionnement : la sortie standard d'un des processus est redirigée sur l'entrée dans un tube et l'entrée standard de l'autre processus est redirigée sur la sortie de ce tube.

Ainsi les deux processus peuvent échanger des données sans passer par un fichier intermédiaire de l'arborescence.

JFA - 151

Exemple :

```
prompt> cmd1 | cmd2
```

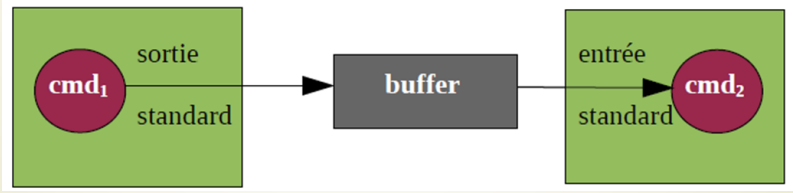


illustration de "cmd1 | cmd2"

Communication entre processus :

Le lancement concurrent de processus communiquant deux par deux par l'intermédiaire d'un tube sera réalisé par une commande de la forme :

```
commande_1 | commande_2 | ... | commande_n
```

Exemple :

```
prompt> who > tmp ; grep cours < tmp
cours     ttya4  Jul 31 10:50
cours     ttyc6  Jul 31 09:34
cours     ttya2  Jul 31 09:02
prompt> rm tmp
prompt>
```

JFA - 152

L'utilisation de la commande ***rm*** évite de conserver le fichier intermédiaire.

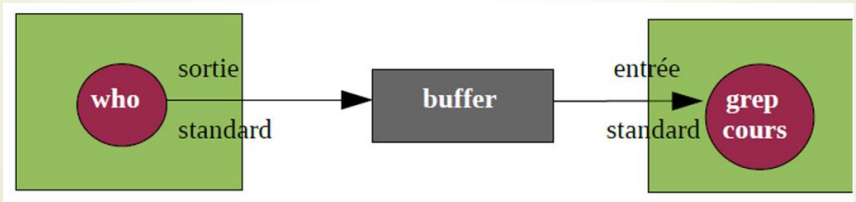
Communication entre processus :

L'utilisation d'un pipe (|) permet de résumer ces trois lignes de commandes en une seule :

```
prompt> who | grep cours
cours     ttya4     Jul 31 10:50
cours     ttyc6     Jul 31 09:34
cours     ttya2     Jul 31 09:02
prompt>
```

JFA - 153

La sortie produite par la commande ***who*** est associée à l'entrée de la commande ***grep***. ***who*** donne la liste des personnes connectées au système à un moment donné ; ***grep*** cherche si la chaîne ***cours*** est présente dans le flot de données qu'elle reçoit. On peut donc considérer que la commande ***grep*** joue le rôle de filtre.



prompt> who | grep cours

Communication entre processus :

Examinons l'exemple suivant :

```
prompt> ps -a | wc -l
9
prompt>
```

La commande "**ps -a | wc -l**" entraîne la création de deux processus concurrents (allocation du processeur). Un tube (fichier particulier, appelé *buffer*) est créé dans lequel les résultats du premier processus ("**ps -a**") sont écrits. Le second processus lit dans le tube.

JFA - 154

Lorsque le processus écrivain se termine et que le processus lecteur dans le tube a fini d'y lire (le tube est donc vide et sans lecteur), ce processus détecte une fin de fichier sur son entrée standard et se termine.

Le système assure la synchronisation de l'ensemble dans le sens où :

- il bloque le processus lecteur du tube lorsque le tube est vide en attendant qu'il se remplisse (s'il y a encore des processus écrivains);
- il bloque (éventuellement) le processus écrivain lorsque le tube est plein (si le lecteur est plus lent que l'écrivain et que le volume des résultats à écrire dans le tube est important).

Communication entre processus :

Le système assure l'implémentation des tubes. Il est chargé de leur création et de leur destruction.

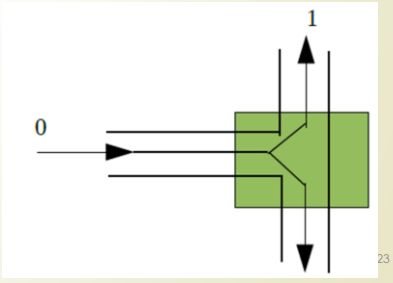
Un tube de communication (|) permet de mémoriser des informations. Il se comporte comme une file FIFO, d'où son aspect unidirectionnel.

Pour vérifier que plusieurs processus existent simultanément, il suffit de lancer la séquence suivante :

JFA - 155

```
prompt> ps -l | tee /dev/tty | wc -l
F S UID PID PPID PRI ... CMD
1 S 102 241 234 158 -bash
1 R 102 294 241 179 ps
1 S 102 295 241 154 tee
1 S 102 296 241 154 wc
5
prompt>
```

La commande "**tee fichier**" utilisée ici correspond au schéma suivant :
Elle permet d'envoyer à la fois sur sa sortie standard et dans le fichier de référence donné en paramètre ce qu'elle lit sur son entrée standard (ici la référence utilisée est ***/dev/tty*** qui permet de visualiser à l'écran les résultats de la commande **ps**).



fichier

Communication entre processus :

JFA - 156

PS : Les commandes ayant la propriété à la fois de lire sur leur entrée standard et d'écrire sur leur sortie standard sont appelées des **filtres**. Les commandes **cat**, **wc**, **sort**, **grep**, **sed**, **sh** ou **awk** sont des filtres ; par contre **echo**, **ls** ou **ps** n'en sont pas.

Autre exemple d'utilisation de la commande **tee** :

```
prompt> ls -l /tmp | grep c1 | tee out
-rw-------1   c1 cours     37888      ...Ex1
-rw-------1   c1 cours     5120       ...Ex2
prompt> cat out
-rw-------1   c1 cours     37888      ...Ex1
-rw-------1   c1 cours     5120       ...Ex2
prompt>
```

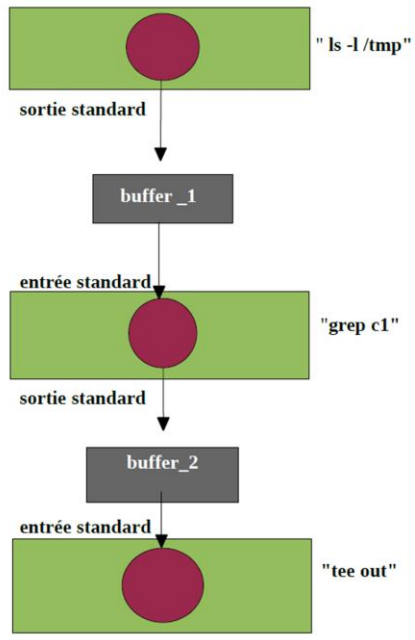
- **ls** envoie la liste des fichiers contenus dans le répertoire */tmp* sur sa sortie standard
- cette sortie est associée à l'entrée standard de **grep**
- **grep** lit les données qui arrivent sur son entrée standard, sélectionne les lignes contenant *c1*, et les envoie sur sa sortie standard
- cette sortie est associée à l'entrée standard de **tee**
- **tee** se contente de dédoubler ses sorties, c'est-à-dire de lire les données qui arrivent sur son entrée standard, et d'envoyer ces données sur sa sortie standard et sur le fichier dont le nom est donné en argument, ici « *out* ».

02/10/2023

Communication entre processus :

JFA - 157

Note : vous ne verrez aucun des résultats intermédiaires de cette suite de pipes.



Communication entre processus :

Un exemple de suite de tubes pour le filtrage des données consiste à afficher l'espace libre (en kilo-octet) sur la partition qui contient le répertoire de travail.

```
prompt> df -k . | tail -1 | sed "s/ */ /g" | cut -d " " -f 4
60833156
prompt>
```

Nous allons détailler cette ligne de commande :

Affichage des statistiques en kilo-octet (option -k) sur le répertoire courant (.) :

JFA - 158

```
prompt> df -k .
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda7 98123404 32281532 60833156 35% /
prompt>
```

On ne garde qu'une seule ligne en partant de la fin :

```
prompt> df -k . | tail -1
/dev/sda7 98123404 32281532 60833156 35% /
prompt>
```


Communication entre processus :

On ne garde qu'un seul espace entre chaque mot :

```
prompt> df -k . | tail -1 | sed "s/ */ /g"
/dev/sda7 98123404 32281532 60833156 35% /
prompt>
```

On ne garde que le quatrième champ de la ligne, l'option « d » précise le séparateur à prendre en compte :

JFA - 159

```
prompt> df -k . | tail -1 | sed "s/ * / /g" | cut -d " " -f 4
60833156
prompt>
```