Droits, liens, transformations, recherche

Sergiu IVANOV

sergiu.ivanov@u-pec.fr

Les diapos disponibles en ligne :

http://lacl.fr/~sivanov/doku.php?id=fr:cours_de_systemes_et_reseaux

Plan du cours

1. Permissions d'accès aux fichiers
2. Liens symboliques et matériels
3. Redirection de flux
4. Transformations
5. Recherche

Plan du cours

Bienvenue à bord de notre train

Qu'est-ce que la ligne de commande ?

J'en ai vue une dans le métro.

Qu'est-ce qu'un fichier ?

Groupes d'utilisateurs

Chaque utilisateur appartient à un ou plusieurs groupes.

Groupes d'utilisateurs

Chaque utilisateur appartient à un ou plusieurs groupes.

Les groupes désignent les « rôles » d'un utilisateur (et donc ses droits).

- `wheel`
- `optical`
- `fcron`

Groupes d'utilisateurs

Chaque utilisateur appartient à un ou plusieurs groupes.

Les groupes désignent les « rôles » d'un utilisateur (et donc ses droits).

- `wheel`
- `optical`
- `fcron`

Pour chaque utilisateur normal (id ≥ 1000) il y a généralement un groupe du même nom.

Groupes d'utilisateurs

Chaque utilisateur appartient à un ou plusieurs groupes.

Les groupes désignent les « rôles » d'un utilisateur (et donc ses droits).

- `wheel`
- `optical`
- `fcron`

Pour chaque utilisateur normal (id ≥ 1000) il y a généralement un groupe du même nom.

La liste de groupes avec les utilisateurs associés se trouve dans `/etc/group`.

Utilisateurs et groupes associés aux fichiers

Chaque fichier a un utilisateur et un groupe associés.

Par défaut ce sont l'utilisateur qui a créé le fichier et le groupe qui lui correspond.

L'utilisateur associé à un fichier s'appelle le possesseur de ce fichier (owner).

Changer le possesseur et le groupe d'un fichier

`chgrp [groupe] [fichier]`

changer le groupe du `[fichier]` vers `[groupe]`

`chown [utilisateur] [fichier]`

changer le possesseur du `[fichier]` vers `[utilisateur]`

`chown [utilisateur]:[groupe] [fichier]`

changer le possesseur du `[fichier]` vers `[utilisateur]`,
et le groupe du `[fichier]` vers `[groupe]`

Permissions d'accès aux fichiers (droits)

Les permissions sont spécifiées en trois catégories.

```
rwx r-x r--
```

Permissions d'accès aux fichiers (droits)

Les permissions sont spécifiées en trois catégories.

`rwx r-x r--`

possesseur groupe autres

Permissions d'accès aux fichiers (droits)

Les permissions sont spécifiées en trois catégories.

`rwx r-x r--`

possesseur groupe autres

- `r` : read (lecture)
- `w` : write (écriture)
- `x` : execute (exécution)

Droits : représentation en octal

```
r = 4 = 100
w = 2 = 010
x = 1 = 001
```

Droits : représentation en octal

```
r = 4 = 100
w = 2 = 010
x = 1 = 001
```

```
rwx r-x r--
```

Droits : représentation en octal

```
r = 4 = 100
w = 2 = 010
x = 1 = 001
```

```
rwx r-x r--
```

```
111 101 100
```

Droits : représentation en octal

```
r = 4 = 100
w = 2 = 010
x = 1 = 001

rwx r-x r--

111 101 100

 7   5   4
```

Droits : représentation en octal

```
r = 4 = 100
w = 2 = 010
x = 1 = 001

rwx r-x r--

111 101 100

 7   5   4
```

Pourquoi représenter les droits en octal ?

Changer les droits

`chmod [droits] [fichier]`

changer les permissions d'accès au `[fichier]` vers `[droits]`

(en octal (un nombre de trois chiffres du coup)).

L'option `-R` permet de faire le changement récursivement.

Ajouter ou retirer des droits (notation symbolique)

Il est possible d'ajouter ou de retirer des droits sans utiliser la notation octale.

Ajouter ou retirer des droits (notation symbolique)

Il est possible d'ajouter ou de retirer des droits sans utiliser la notation octale.

```
rwx r-x r--

u   g   o
```

Ajouter ou retirer des droits (notation symbolique)

Il est possible d'ajouter ou de retirer des droits sans utiliser la notation octale.

```
rwx r-x r--

u   g   o
```

`chmod u-x fichier` : `rw- r-x r--`

- retirer le droit à l'exécution au possesseur

Ajouter ou retirer des droits (notation symbolique)

Il est possible d'ajouter ou de retirer des droits sans utiliser la notation octale.

```
rwx r-x r--

u   g   o
```

`chmod u-x fichier` : `rw- r-x r--`

- retirer le droit à l'exécution au possesseur

`chmod gu-x fichier` : `rw- r-- r--`

- retirer le droit à l'exécution au possesseur et au groupe

Ajouter ou retirer des droits (notation symbolique)

Il est possible d'ajouter ou de retirer des droits sans utiliser la notation octale.

```
rwx r-x r--

u   g   o
```

`chmod u-x fichier` : `rw- r-x r--`

- retirer le droit à l'exécution au possesseur

`chmod gu-x fichier` : `rw- r-- r--`

- retirer le droit à l'exécution au possesseur et au groupe

`chmod o+wx fichier` : `rwx r-x rwx`

- ajouter le droit à l'écriture et à l'exécution aux autres

Ajouter ou retirer des droits (notation symbolique)

Il est possible d'ajouter ou de retirer des droits sans utiliser la notation octale.

```
rwx r-x r--

u   g   o
```

`chmod u-x fichier` : `rw- r-x r--`

- retirer le droit à l'exécution au possesseur

`chmod gu-x fichier` : `rw- r-- r--`

- retirer le droit à l'exécution au possesseur et au groupe

`chmod o+wx fichier` : `rwx r-x rwx`

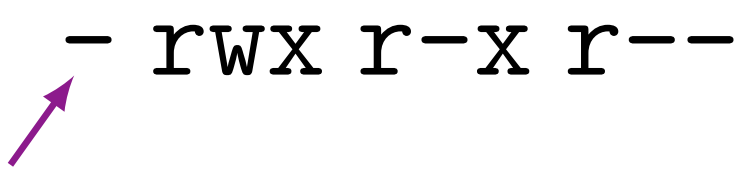
- ajouter le droit à l'écriture et à l'exécution aux autres

`chmod +x fichier` : `rwx r-x r-x`

- ajouter le droit à l'exécution à tout le monde

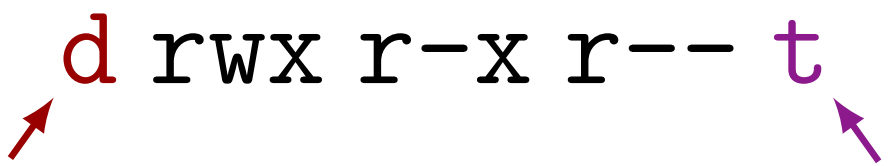
Permissions spéciales : types de fichiers

Permissions spéciales : types de fichiers



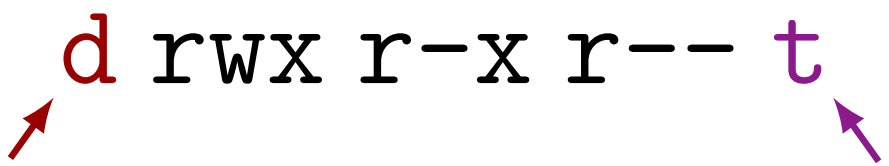
- `-` : fichier normal
- `d` : répertoire (directory)
- `l` : lien symbolique

Permissions spéciales : sticky bit (bit collant)



Le sticky bit est généralement attribué aux répertoires.

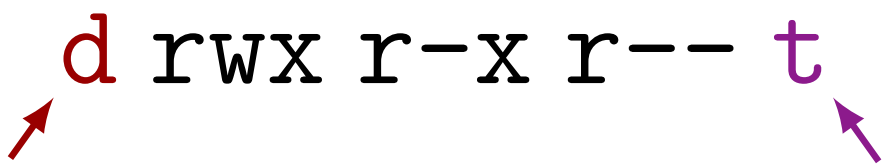
Permissions spéciales : sticky bit (bit collant)



Le sticky bit est généralement attribué aux répertoires.

Tout fichier créé dans un tel répertoire ne peut être renommé ou supprimé que par son possesseur.

Permissions spéciales : sticky bit (bit collant)



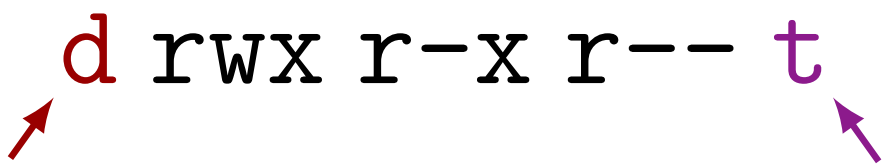
Le sticky bit est généralement attribué aux répertoires.

Tout fichier créé dans un tel répertoire ne peut être renommé ou supprimé que par son possesseur.

Typiquement attribué aux dossiers temporaires.

- `/tmp`, `/var/tmp`

Permissions spéciales : sticky bit (bit collant)



Le sticky bit est généralement attribué aux répertoires.

Tout fichier créé dans un tel répertoire ne peut être renommé ou supprimé que par son possesseur.

Typiquement attribué aux dossiers temporaires.

- `/tmp`, `/var/tmp`

```
chmod +t [dossier]
```

Permission spéciales : setuid

Permission setuid : `rws r-x r--`

- le fichier sera exécuté en tant que son possesseur

Permission spéciales : setuid

Permission setuid : `rws r-x r--`

- le fichier sera exécuté en tant que son possesseur
- `sudo` est exécuté en tant que `root`, même s'il est lancé par un utilisateur normal, parce que le possesseur de `sudo` est `root`.

Permission spéciales : setuid

Permission setuid : `rws r-x r--`

- le fichier sera exécuté en tant que son possesseur
- `sudo` est exécuté en tant que `root`, même s'il est lancé par un utilisateur normal, parce que le possesseur de `sudo` est `root`.

```
chmod u+s [fichier]
```

Permission spéciales : setgid

Permission setgid : `rws r-s r--`

- le fichier sera exécuté dans le groupe qui lui est attribué

Permission spéciales : setgid

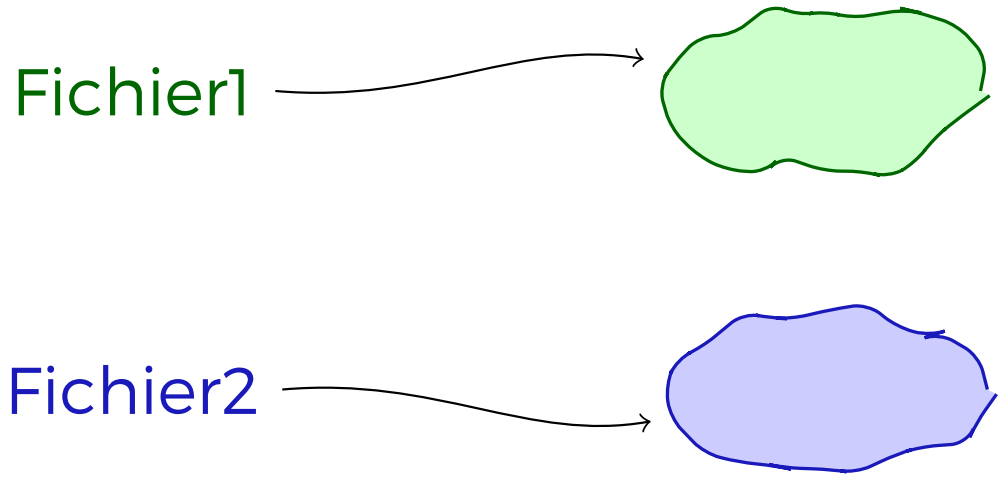
Permission setgid : `rws r-s r--`

- le fichier sera exécuté dans le groupe qui lui est attribué

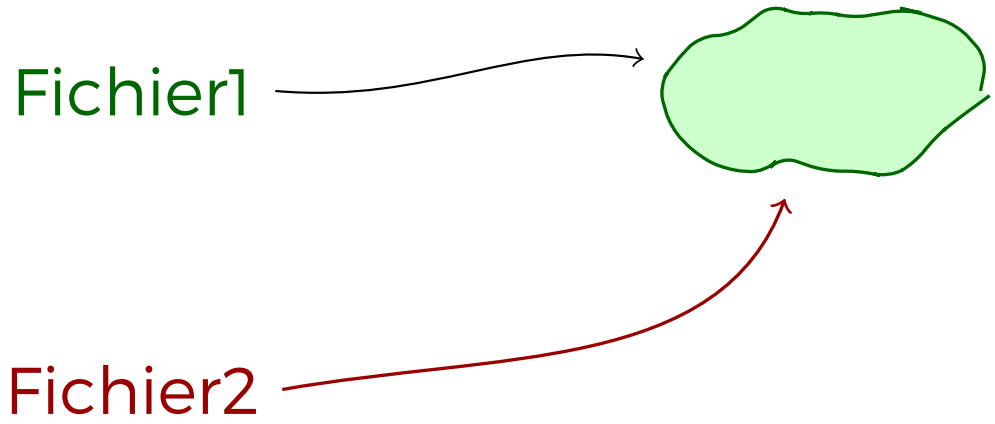
```
chmod g+s [fichier]
```

Plan du cours

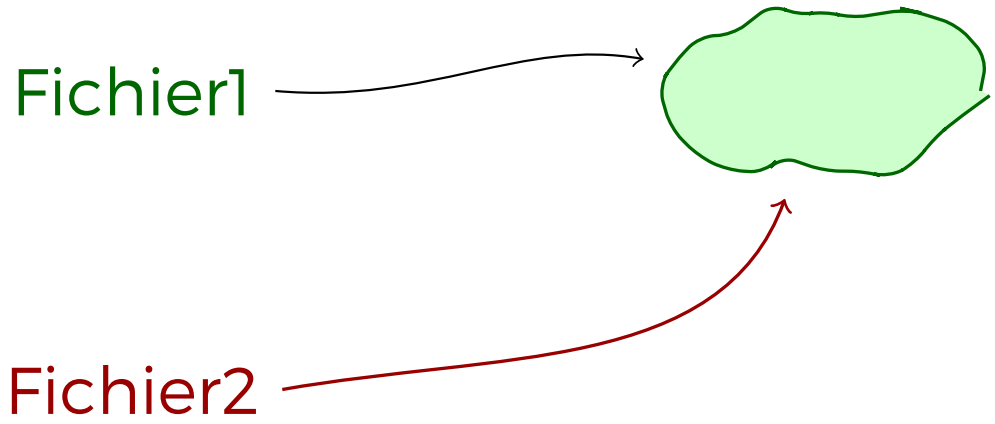
Liens matériels (hard links)



Liens matériels (hard links)

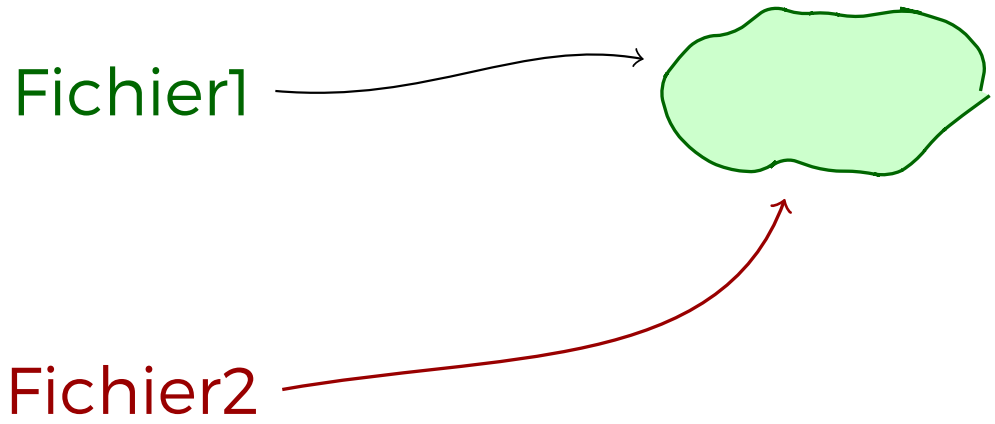


Liens matériels (hard links)



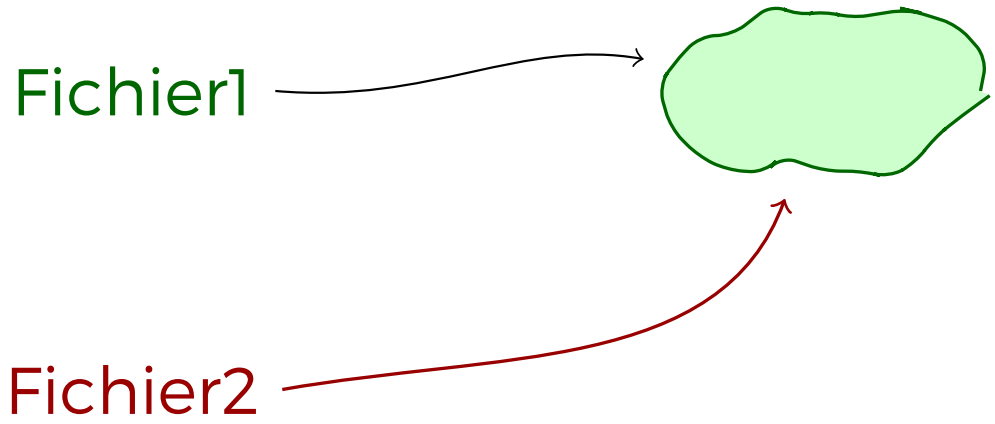
Fichier2 est habituellement appelé un lien matériel vers Fichier1.

Liens matériels (hard links)



Fichier2 est habituellement appelé un lien matériel vers Fichier1. L'affirmation inverse est vraie aussi.

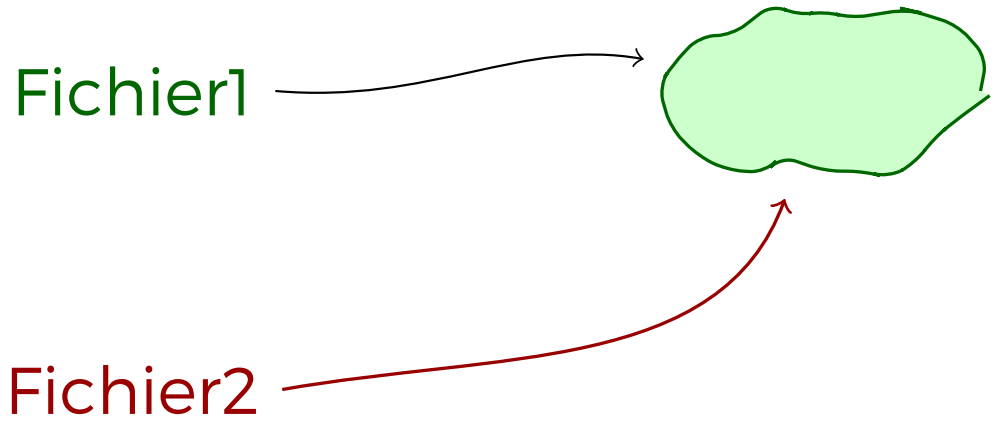
Liens matériels (hard links)



Fichier2 est habituellement appelé un lien matériel vers Fichier1. L'affirmation inverse est vraie aussi.

Tout fichier est un lien matériel.

Liens matériels (hard links)

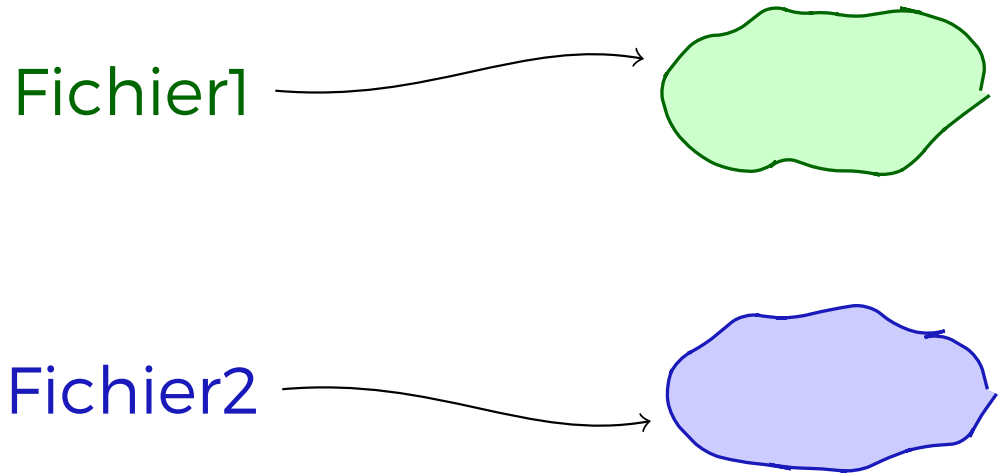


Fichier2 est habituellement appelé un lien matériel vers Fichier1. L'affirmation inverse est vraie aussi.

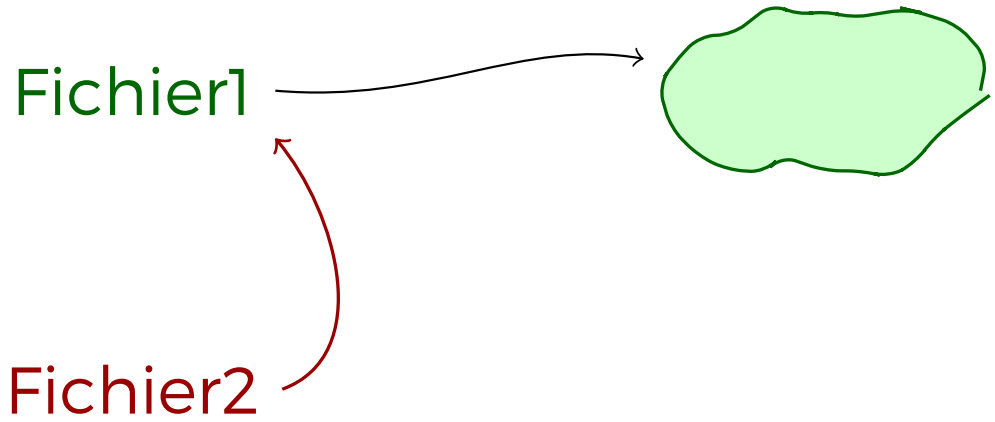
Tout fichier est un lien matériel.

Un lien matériel pointe vers le contenu dans le même système de fichiers (dans la même partition).

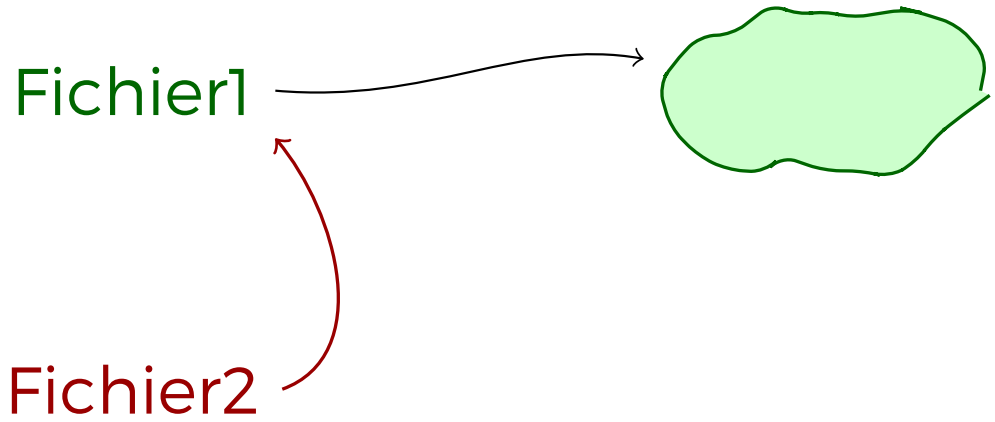
Liens symboliques (symlinks)



Liens symboliques (symlinks)

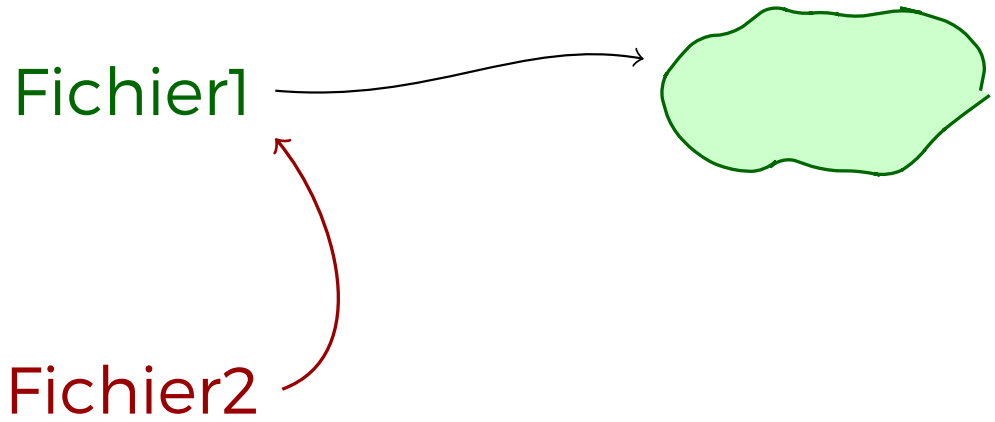


Liens symboliques (symlinks)



Fichier2 est un lien symbolique vers Fichier1.

Liens symboliques (symlinks)



Fichier2 est un lien symbolique vers Fichier1.

Un lien symbolique peut pointer vers de fichiers dans d'autres systèmes de fichiers (d'autres partitions).

Création de liens

`ln [cible] [nom]`

créer le lien matériel `[nom]` vers la `[cible]`

`ln -s [cible] [nom]`

créer le lien symbolique `[nom]` vers la `[cible]`

`ls -l [lien]`

afficher la cible du `[lien]`

Liens symboliques vs. liens matériels : taille

Quel type de lien prend plus de place ?

Liens symboliques vs. liens matériels : taille

Quel type de lien prend plus de place ?

Les liens symboliques prennent plus de place, parce que ce sont des fichiers qui contiennent le chemin vers la cible.

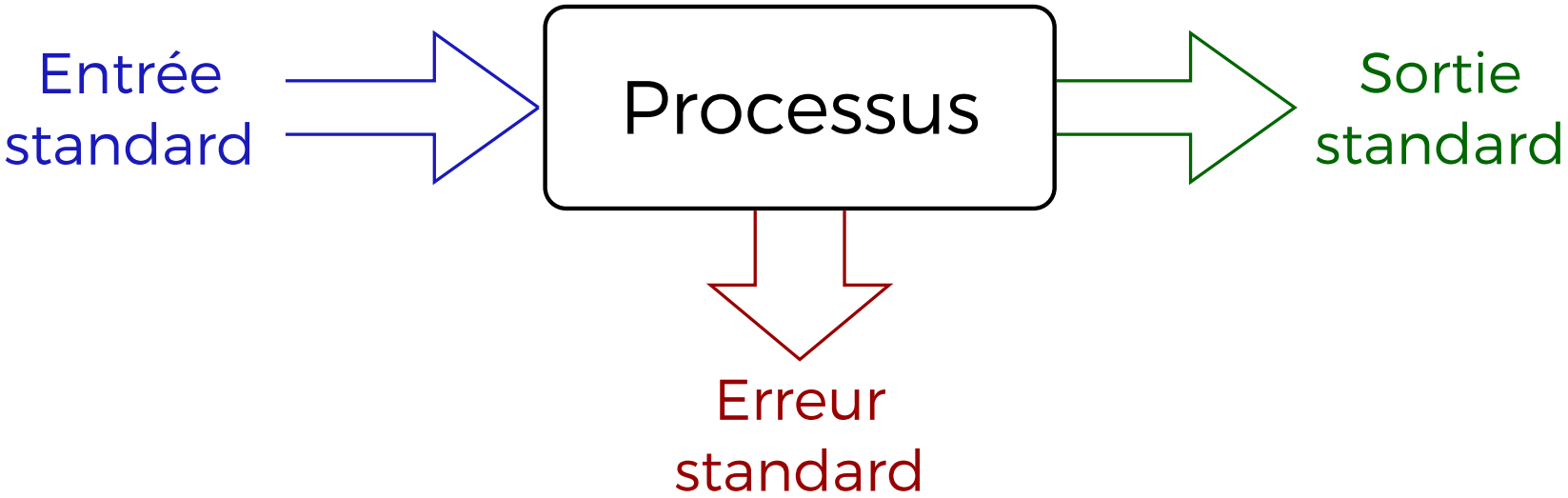
(grosso modo)

Plan du cours

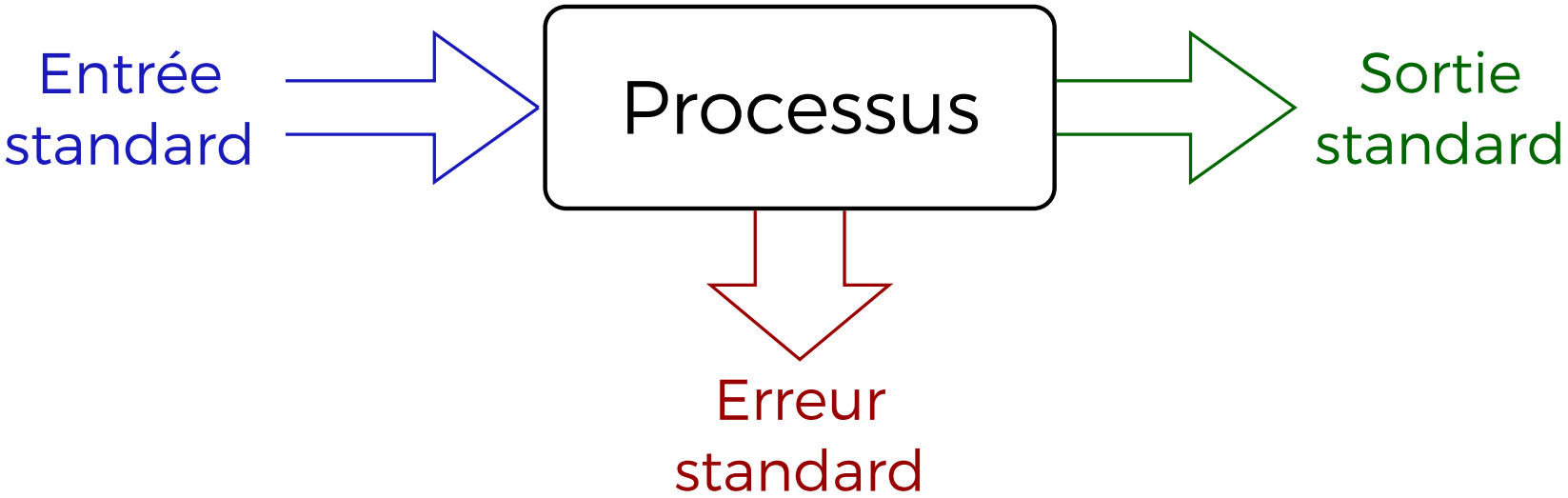
La menace fantôme

Qu'est-ce qu'un processus ?

Les flux standards



Les flux standards



Les numéros des flux (descripteurs fichier)

- 0 : entrée standard
- 1 : sortie standard
- 2 : erreur standard

Redirection de flux

`commande < [source]`

- mettre `[source]` en entrée standard
- `cat < myfichier.txt`

`commande > [destination]`

- rediriger la sortie standard vers `[destination]`
- `date > myfichier.txt`

`commande 2> [destination]`

- rediriger l'erreur standard vers `[destination]`
- `date wat 2> myfichier.txt`

Redirection maline de l'erreur standard

`commande 2>&1`

rediriger l'erreur standard vers la sortie standard

Redirection maline de l'erreur standard

`commande 2>&1`

rediriger l'erreur standard vers la sortie standard

`commande >[cible] 2>&1`

rediriger l'erreur standard vers la sortie standard, puis

rediriger la sortie standard (donc les deux flux) vers `[cible]`

Redirection maline de l'erreur standard

`commande 2>&1`

rediriger l'erreur standard vers la sortie standard

`commande >[cible] 2>&1`

rediriger l'erreur standard vers la sortie standard, puis

rediriger la sortie standard (donc les deux flux) vers `[cible]`

L'ordre compte !

`commande 2>&1 >[cible]` ne fait pas la même chose !

/dev/null : le trou noire

Toute information envoyée vers `/dev/null` est jetée à la poubelle (supprimée).

/dev/null : le trou noire

Toute information envoyée vers `/dev/null` est jetée à la poubelle (supprimée).

`commande 2>/dev/null`

supprimer tous les messages d'erreur

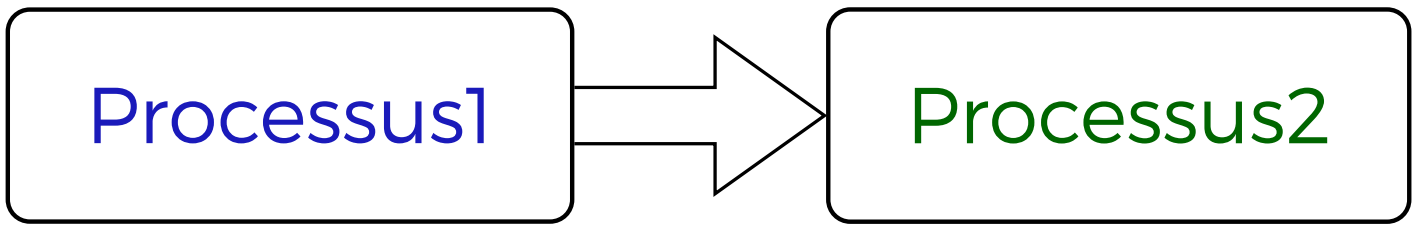
`commande 2>/dev/null 2>&1`

supprimer tous les messages

Pipes

https://openclipart.org/

Rediriger la sortie standard de `Processus1` vers l'entrée de `Processus2` :

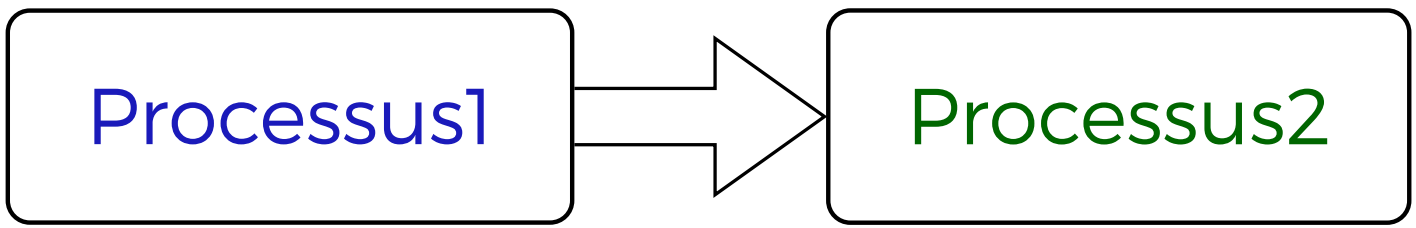


`Commande1 | Commande2`

Pipes

https://openclipart.org/

Rediriger la sortie standard de `Processus1` vers l'entrée de `Processus2` :




```
Commande1 | Commande2
```

On peut enchaîner plusieurs commandes :

```
Commande1 | Commande2 | Commande3
```

Plan du cours

Le concept de transformation

Transformation = commande bien adaptée à l'utilisation dans des chaînes de commandes reliées par des pipes

(gros clin d'œil à la programmation fonctionnelle)

echo : afficher une chaîne de caractères

`echo [chaîne]`

afficher la `[chaîne]`

Utile dans scripts et chaînage de commandes par pipes.

head : afficher quelques premières lignes

`head [fichier]`

afficher les 10 premières lignes du `[fichier]`

`head -n k [fichier]`

afficher les `k` premières lignes du `[fichier]`

head : afficher quelques premières lignes

`head [fichier]`

afficher les 10 premières lignes du `[fichier]`

`head -n k [fichier]`

afficher les `k` premières lignes du `[fichier]`

`commande | head`

afficher les 10 premières lignes de la sortie de `[commande]`

`commande | head -n k`

afficher les `k` premières lignes de la sortie de `[commande]`

tail : afficher quelques dernières lignes

`tail [fichier]`

afficher les 10 dernières lignes du `[fichier]`

`tail -n k [fichier]`

afficher les `k` dernières lignes du `[fichier]`

`commande | tail`

afficher les 10 dernières lignes de la sortie de `[commande]`

`commande | tail -n k`

afficher les `k` dernières lignes de la sortie de `[commande]`

Fichier entrée vs. entrée standard

`head`, `tail`, et la plupart d'autres commandes vont lire l'entrée standard si aucun fichier n'est leur spécifié en argument.

Fichier entrée vs. entrée standard

`head`, `tail`, et la plupart d'autres commandes vont lire l'entrée standard si aucun fichier n'est leur spécifié en argument.

Ces deux usages sont donc équivalents :

- `cat [fichier] | head`
- `head [fichier]`

WC : compter le nombre de lignes ou de mots

`wc -l [fichier]` (ou bien `cat [fichier] | wc -l`)

compter le nombre de lignes

`wc -w [fichier]`

compter le nombre de mots

sort : trier

`sort [fichier]`

trier en ordre ascendant

`sort -r [fichier]`

trier en ordre descendant

`sort -k c [fichier]`

trier d'après la colonne numéro `c`

cut : découper et extraire des colonnes

```
cut  -d [chaîne]  -f c  [fichier]
```

utiliser la `[chaîne]` comme séparateur de colonnes et afficher uniquement la colonne `c`.

cut : découper et extraire des colonnes

```
cut  -d [chaîne]  -f c  [fichier]
```

utiliser la `[chaîne]` comme séparateur de colonnes et afficher uniquement la colonne `c`.

```
cut -d' ' -f3,5
```

utiliser l'espace comme séparateur et afficher les colonnes 3 et 5.

tr : remplacer des caractères

`commande | tr [chaîne1] [chaîne2]`

remplacer les caractères de `[chaîne1]` par les caractères de `[chaîne2]` en positions correspondantes

tr : remplacer des caractères

`commande | tr [chaîne1] [chaîne2]`

remplacer les caractères de `[chaîne1]` par les caractères de `[chaîne2]` en positions correspondantes

`tr ab cd`

remplacer tous les `a` par `c` et tous les `b` par `d`

tr : remplacer des caractères

`commande | tr [chaîne1] [chaîne2]`

remplacer les caractères de `[chaîne1]` par les caractères de `[chaîne2]` en positions correspondantes

`tr ab cd`

remplacer tous les `a` par `c` et tous les `b` par `d`

Ne prend pas de fichier en argument.

xargs : lancer une commande pour toute ligne

```
commande | xargs -L1 [commande]
```

lancer `[commande]` `[ligne]` pour toute `[ligne]` dans l'entrée standard

```
cat liste | xargs -L1 rm
```

supprimer tous les fichiers les chemins vers lesquels sont donnés dans le fichier `liste`

Plan du cours

grep : trouver et afficher des motifs (1)

`grep [motif] [fichier]`

trouver et afficher toutes les lignes du fichier `[fichier]`

qui contiennent le motif `[motif]`

grep : trouver et afficher des motifs (1)

`grep [motif] [fichier]`

trouver et afficher toutes les lignes du fichier `[fichier]`

qui contiennent le motif `[motif]`

`grep -i [motif] [fichier]`

faire une recherche insensible au majuscules

grep : trouver et afficher des motifs (1)

`grep [motif] [fichier]`

trouver et afficher toutes les lignes du fichier `[fichier]`

qui contiennent le motif `[motif]`

`grep -i [motif] [fichier]`

faire une recherche insensible au majuscules

`grep -n [motif] [fichier]`

afficher les numéros des lignes trouvées

grep : trouver et afficher des motifs (1)

`grep [motif] [fichier]`

trouver et afficher toutes les lignes du fichier `[fichier]`

qui contiennent le motif `[motif]`

`grep -i [motif] [fichier]`

faire une recherche insensible au majuscules

`grep -n [motif] [fichier]`

afficher les numéros des lignes trouvées

`grep -c [motif] [fichier]`

compter le nombre de lignes trouvées uniquement

grep : trouver et afficher des motifs (2)

`grep -v [motif] [fichier]`

afficher les lignes qui ne contiennent pas le `[motif]`

grep : trouver et afficher des motifs (2)

`grep -v [motif] [fichier]`

afficher les lignes qui ne contiennent pas le `[motif]`

`grep -r [motif] [dossier]`

chercher le `[motif]` récursivement dans tous les fichiers dans `[dossier]`

grep : expressions rationnelles (regexp)

`grep -E [motif] [fichier]`

le `[motif]` est une expression rationnelle

Le terme « regexp » vient de « regular expression ».

Regexp : caractères

`[ad]` soit `a`, soit `d` (une fois)

`[^ad]` un caractère qui n'est ni `a`, ni `d`

`[a-d]` l'un des caractères entre `a` et `d` (une fois)

Regexp : répétitions

`[ad]?` soit `a`, soit `d` au plus une fois

`[ad]+` soit `a`, soit `d` au moins une fois

`[ad]*` soit `a`, soit `d` un nombre arbitraire de fois

Regexp : choix

`[a-z]+ | [0-9]+` soit une suite de lettres,
soit une suite de chiffres

Regexp : classes de caractères

```
[[:alpha:]]  =  [a-zA-Z]

[[:digit:]]  =  [0-9]

[[:alnum:]]  =  [[:alpha:] | [:digit:]]
```

find : trouver des fichiers

`find [dossier] -name "*.txt"`

trouver tous les fichiers dont le nom se termine en `.txt`

dans `[dossier]`

`find [dossier] -iname "*.TxT"`

faire une recherche insensible aux majuscules

http://www.tecmint.com/35-practical-examples-of-linux-find-command/

find : recherche par type

```
find [dossier] -type d -name "*machin*"
```

trouver tous les dossiers dont le nom contient « `machin` »

```
find [dossier] -type f -name "*machin*"
```

trouver tous les fichiers qui ne sont pas des dossiers

find : recherche par droits

```
find [dossier] -perm 754 -name "*machin*"
```

trouver les fichiers dont les permissions d'accès sont 754

```
find [dossier] ! -perm 754 -name "*machin*"
```

les fichiers dont les permissions d'accès ne sont pas 754

find : recherche par temps de modification

```
find [dossier] -mtime 50
```

trouver les fichiers qui ont été modifiés il y a 50 jours

```
find [dossier] -mtime +50 -mtime -100
```

trouver les fichiers qui ont été modifiés il y a plus de 50 jours, mais moins de 100 jours

find : exécution de commandes

```
find [dossier] -name "*.mp3" -exec rm {} \;
```

- supprimer tous les fichiers MP3
- `-exec` exécute une commande pour chaque fichier trouvé
- `{}` fait référence au nom du fichier trouvé
- `\;` marque la fin de la commande à exécuter