Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

15 février 2012

Partie #4

Plan

Métacaractère et Chemins ciblés

Le caractère *

- Le cataractère * est utilisé comme un *jocker* pour remplacer une chaîne de caractères,
- Il est utilisé pour pointer plusieurs fichiers ou répertoires dont le nom partage un motif commun.
- Le caractère * peut être placé en début, en fin ou au milieu d'une chaîne de caractères,
- Le caractère * peut être répété.

Exemple de manipulation avec la commande mv

```
[ login@localhost ~ ] mv *.jpg Images/
```

Ici, le chemin `*.jpg` pointe tous les fichiers du répertoire courant dont le nom se fini par l'extension `.jpg`. Il pointe donc les fichiers `etacentauri.jpg` et `aldebaran.jpg` et exclue les autres fichiers (ici le fichier `alphacentauri.gif`).

```
chez_moi/.............  Répertoire Courant
├── aldebaran.jpg ..........  Fichier ciblé
├── alphacentauri.gif
├── etacentauri.jpg ........  Fichier ciblé
└── Images/ ..............  Répertoire final
```

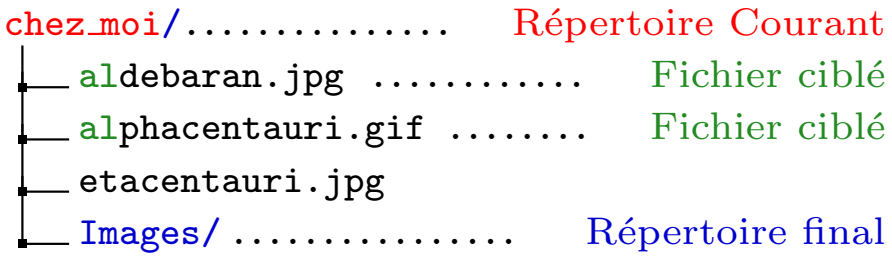
```
chez_moi/.............  Répertoire Courant
├── alphacentauri.gif
└── Images/ ..............  Répertoire final
    ├── aldebaran.jpg .....  Fichier déplacé
    └── etacentauri.jpg ...  Fichier déplacé
```

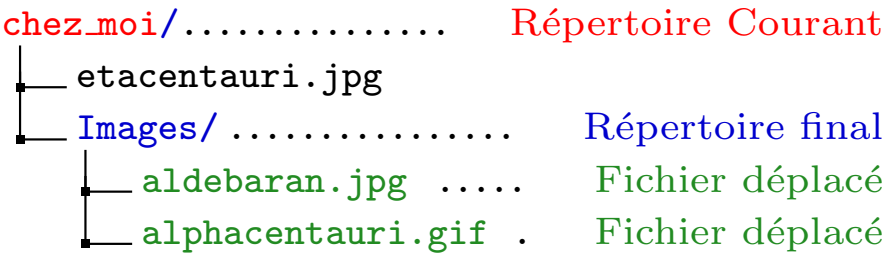
Métacaractère et Chemins ciblés

Exemple de manipulation avec la commande mv

```
[ login@localhost ~ ] mv al* Images/
```

Ici, le chemin `al*` pointe tous les fichiers du répertoire courant dont le nom commence par les caractères `al`. Il pointe donc les fichiers `aldebaran.jpg` et `alphacentauri.gif` et exclue les autres fichiers (ici le fichier `etacentauri.jpg`).

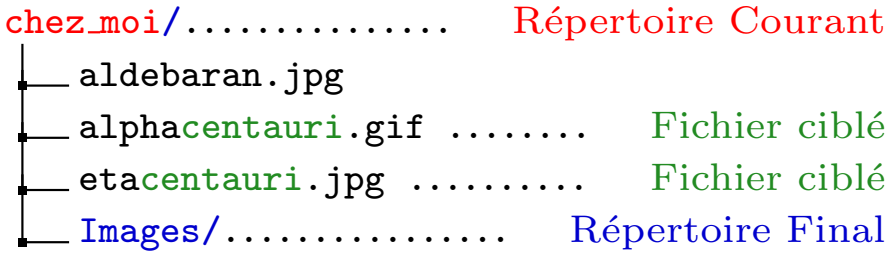


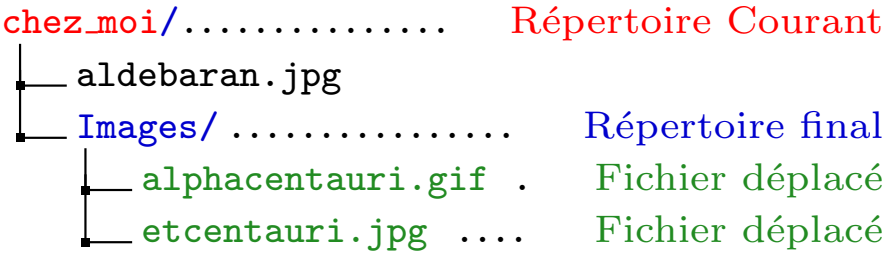


Exemple de manipulation avec la commande mv

```
[ login@localhost ~ ] mv *centauri* JPG/
```

Ici, le chemin `*centauri*` pointe tous les fichiers du répertoire courant dont le nom contient la chaîne de caractères `centauri`. Il pointe donc les fichiers `alphacentauri.gif` et `etacentauri.jpg` et exclue les autres fichiers (ici le fichier `aldebaran.jpg`).

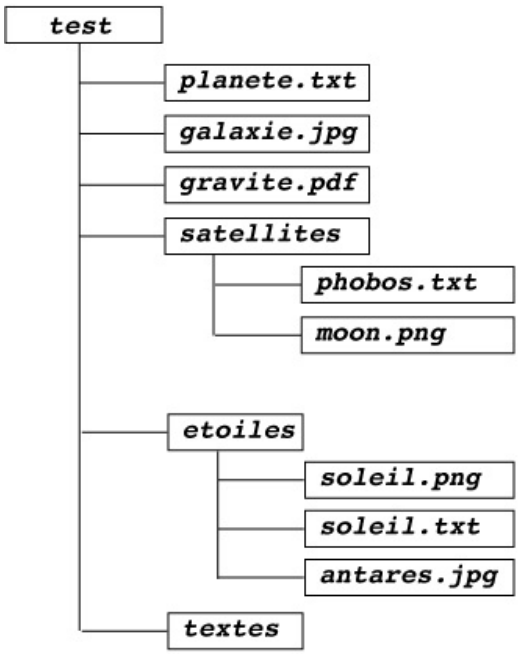


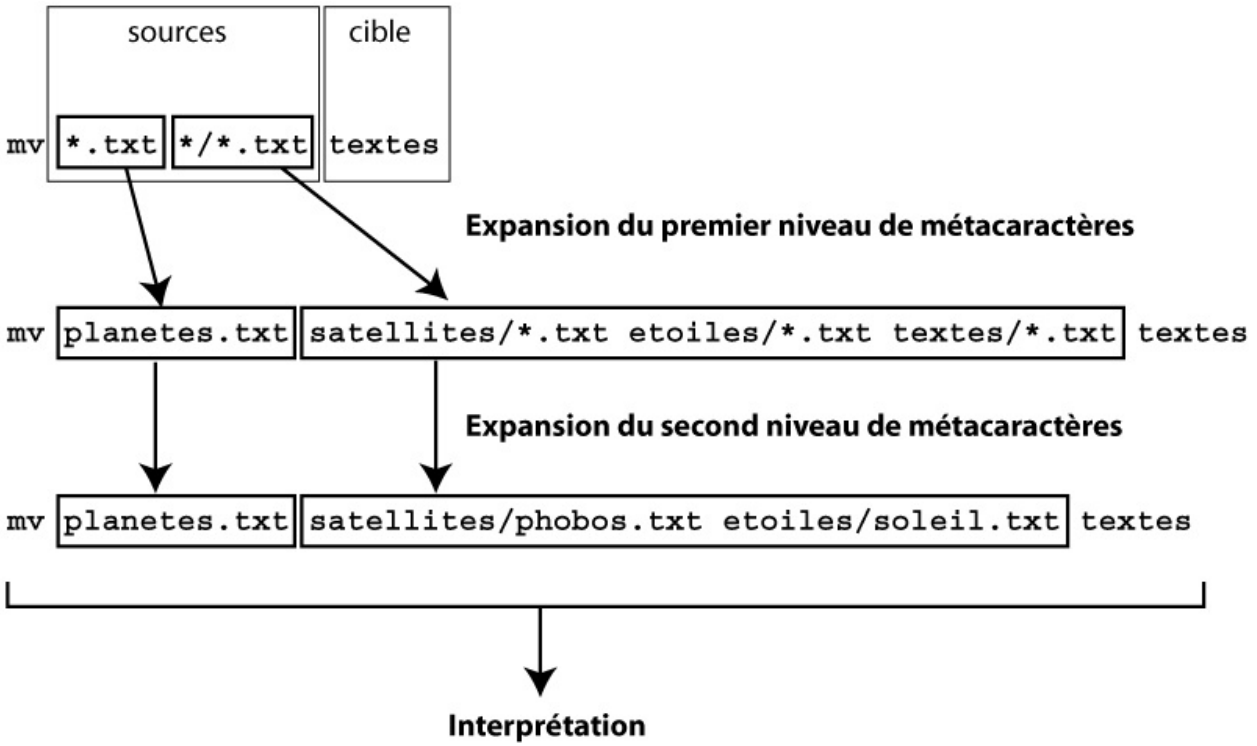


Métacaractère et Chemins ciblés

Exemple plus complexe et Détails de l'interprétation

- Le cararctère * est développé lors de l'interprétation.





find

Syntaxe

```
find  depart -iname "motif"
```

Description

- Recherche dans les répertoires et sous-répertoires les fichiers dont le nom correspond au motif en partant du point de l'arborescence spécifié par le depart.
- L'option -iname indique que le motif sera recherché sans tenir compte des majuscules et minuscules.

Exemple d'utilisation:

```
chez_moi/ . Répertoire courant
├── Mes_Images/
│   ├── JPG/
│   │   ├── soleil.jpg
│   │   └── lune.jpg
│   └── GIF/
│       ├── alphacentauri.gif
│       └── terre.gif
└── Mes_Photos/
    ├── etacentauri.jpg
    └── phobos.gif
```

```
[ login@localhost ~ ] find . -iname *.gif
./Mes_Images/GIF/alphacentauri.gif
./Mes_Images/GIF/terre.gif
./Mes_Photos/phobos.gif

[ login@localhost ~ ] find . -iname *centauri*
./Mes_Images/GIF/alphacentauri.gif
./Mes_Photos/etacentauri.jpg

[ login@localhost ~ ] find Mes_Images/ -iname *e.*
Mes_Images/GIF/terre.gif
Mes_Images/JPG/lune.jpg

[ login@localhost ~ ] ■
```

find(bis)

Syntaxe

```
find  depart -iname "motif" -exec commande \;
```

Description

- Exécute la commande sur la liste des fichiers identifiés par `find`,
- Dans la rédaction de la commande, la liste des fichiers est symbolisée par les caractères {}.

Exemple d'utilisation:

```
[ login@localhost ~ ] find ./ -iname *centauri* -exec mv {} Mes_Photos \;
```

chez_moi/.....État Initial Répertoire courant

```
Mes_Images/
    JPG/
        etacentauri.jpg...Trouvé par find
        soleil.jpg
    GIF/
        alphacentauri.gif.Trouvé par find
        terre.gif
Mes_Photos/
```

chez_moi/.....État Final Répertoire courant

```
Mes_Images/
    JPG/
        soleil.jpg
    GIF/
        terre.gif
Mes_Photos/
    alphacentauri.jpg.....Déplacé par mv
    etacentauri.jpg.......Déplacé par mv
```

Liens symboliques

Définition

- C'est une entrée spéciale contenue dans la liste des références (fichiers ou répertoires) d'un répertoire qui pointe vers une autre référence (fichier ou répertoire) déjà existante dans l'arborescence du système de fichiers.
- Autrement dit, c'est un *alias* placé dans un répertoire qui fait référence à un autre fichier ou répertoire de l'arborescence (où qu'il soit).
- C'est un *"raccourci"* placé dans l'arborescence.

Manipulation

- Lors de la création d'un lien symbolique, seule la référence est copiée. Les données pointées par la référence n'existent toujours qu'en un seul exemplaire.
- Le même fichier sera accessible par son chemin d'origine ou par le chemin de l'*alias* (*i.e.* le chemin du lien symbolique).
- La destruction du lien symbolique n'entraine pas la destruction du fichier original.
- Plusieurs liens symboliques peuvent pointer le même fichier ou le même répertoire.

ln

Syntaxe

```
ln -s source cible
```

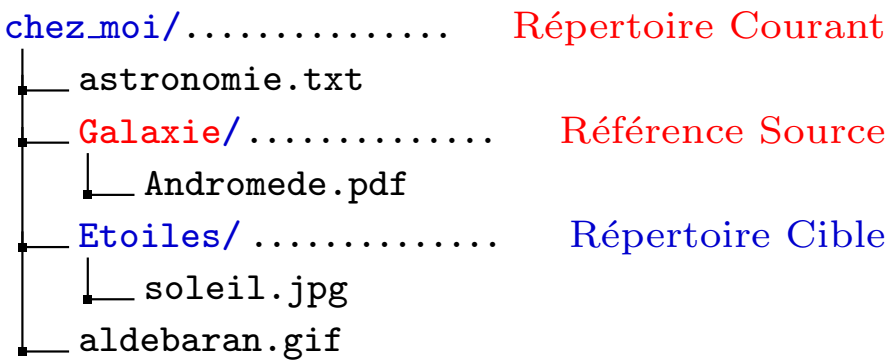
Description

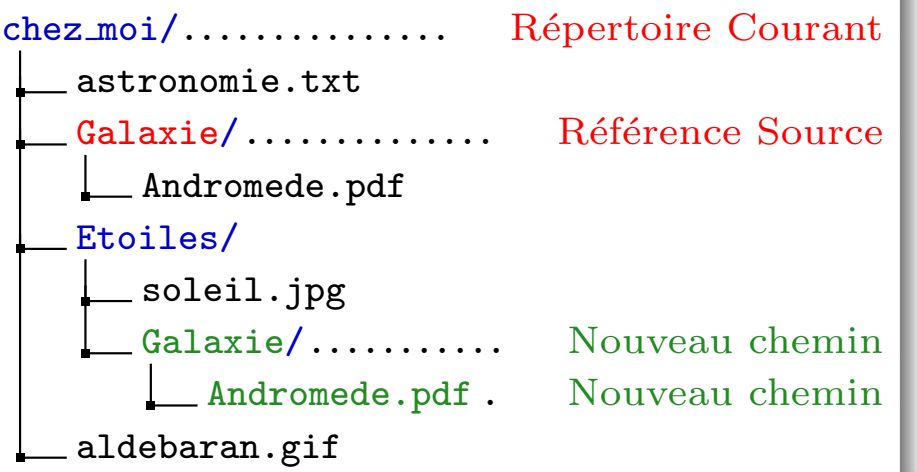
- Crée un lien symbolique entre la référence source et le chemin cible..

Exemple d'utilisation:

```
[ login@localhost ~ ] ln -s Galaxies Etoiles/Galaxies
```

Le lien symbolique sur un répertoire donne également accès à toutes les références contenues dans le répertoire pointé par le lien. Ainsi, le fichier `~/Galaxie/Adromede.pdf` est aussi accessible par le chemin `~/Etoiles/Galaxie/Andromede.pdf`.



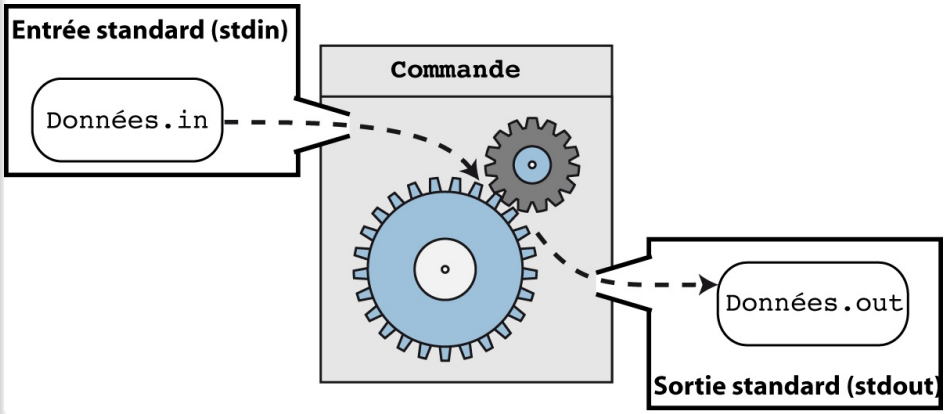


Plan

Entrée et sortie standard

Rappel : Les programmes informatiques

- Un programme prend des données en entrée. Ces données peuvent être lues dans un fichier ou fournies par un flux du système.
- Le programme manipule ces données.
- Le programme fournit un résultat en sortie (des données). Ces données peuvent être écrites dans un fichier ou exportées comme un flux vers le système.



Les flux de données

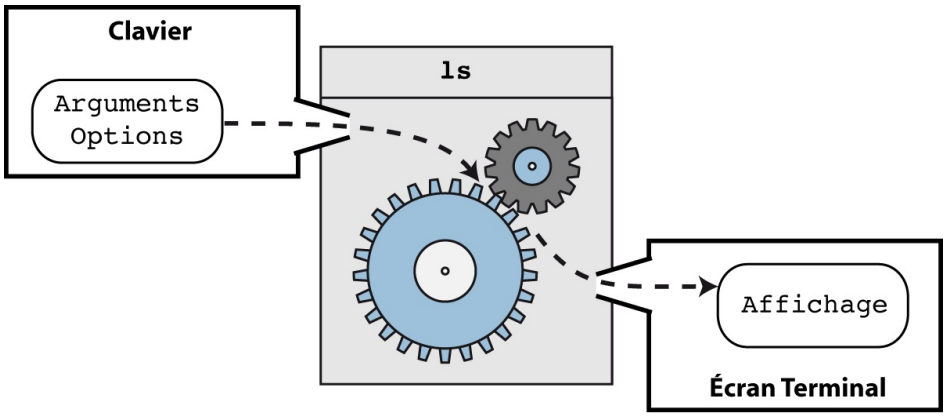
Pour fonctionner, un programme a donc besoin de lire des données (flux d'entrée : input) et d'écrire les résultats de ses évaluations (flux de sortie : output). On distingue 3 types de flux de données :

- **STDIN** : entrée standard (là où sont lues les données),
- **STDOUT** : sortie standard (là où sont écrits les résultats),
- **STDERR** : sortie erreur (là où sont écrit les messages d'erreur).

Entrée et sortie standard

Les commandes qui lisent sur l'entrée standard

- Certaines commandes Linux qui traitent les données d'un fichier (dont le chemin est passé en paramètre) peuvent alternativement, si aucun chemin fichier n'est spécifié, travailler directement avec les données lues sur l'entrée standard.
- Parmi les commandes déjà vue : `cat`, `head`, `tail`, `cut`, `sed`, `grep`.
- **Par défaut, l'entrée standard est le clavier**.



Les commandes qui écrivent sur la sortie standard

- Les affichages produits par les commandes Linux sont le résultat de leur évaluation. Ce résultat est écrit sur la sortie standard.
- **Par défaut, la sortie standard est l'écran**.

Redirection des Entrée/Sorties

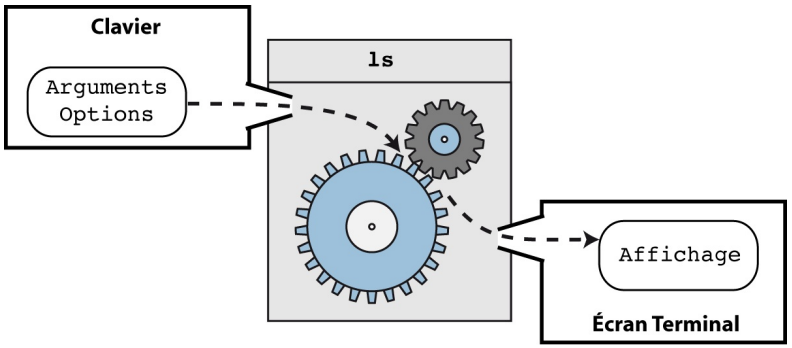
Commandes de Redirection

Il est possible de modifier le comportement par défaut des commandes et de donner une entrée et/ou une sortie standard différente des entrées/sorties standards.

- `command > fichier.out`
  - **Redirige la sortie standard** de la commande `command` vers le fichier `fichier.out`.
  - Si le fichier `fichier.out` n'existe pas, il est créé avec comme contenu les affichages produits par la commande `command`.
  - **Si le fichier `fichier.out` existe, son contenu est écrasé** et remplacé par les affichages produits par la commande `command`.
- `command >> fichier.out`
  - **Redirige la sortie standard** de la commande `command` vers le fichier `fichier.out`.
  - Si le fichier `fichier.out` n'existe pas, il est créé avec comme contenu les affichages produits par la commande `command`.
  - Si le fichier `fichier.out` existe, les affichages produits par la commande `command` sont **ajoutés à la fin du contenu du fichier**.
- `command 2> fichier.err`
  - **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec écrasement du contenu** si le fichier de sortie existe déjà.
- `command 2>> fichier.err`
  - **Redirige la sortie erreur** de la commande `command` vers le fichier `fichier.err` **avec préservation du contenu** si le fichier de sortie existe déjà.

Exemple de redirection

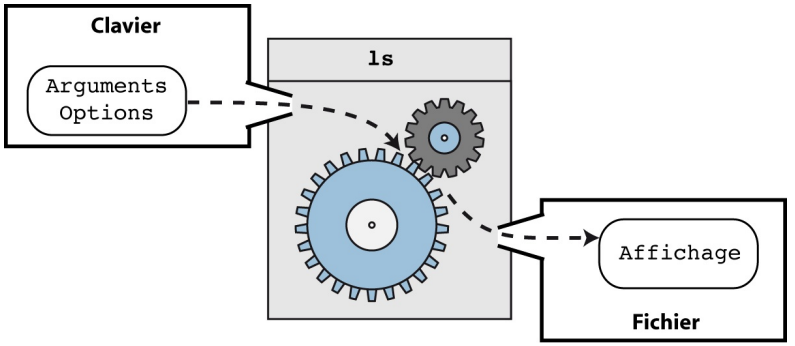
Comportement par défaut de la commande `ls`




```
[ login@localhost ~ ] ls
aldenaran.jpg alphacentauri.gif etacentauri.jpg

[ login@localhost ~ ] ls
aldenaran.jpg alphacentauri.gif etacentauri.jpg

[ login@localhost ~ ] ■
```

La sortie standard de la première commande `ls` est l'écran. La liste du contenu du répertoire courant est affichée à l'écran.

Redirection de la sortie de la commande `ls`



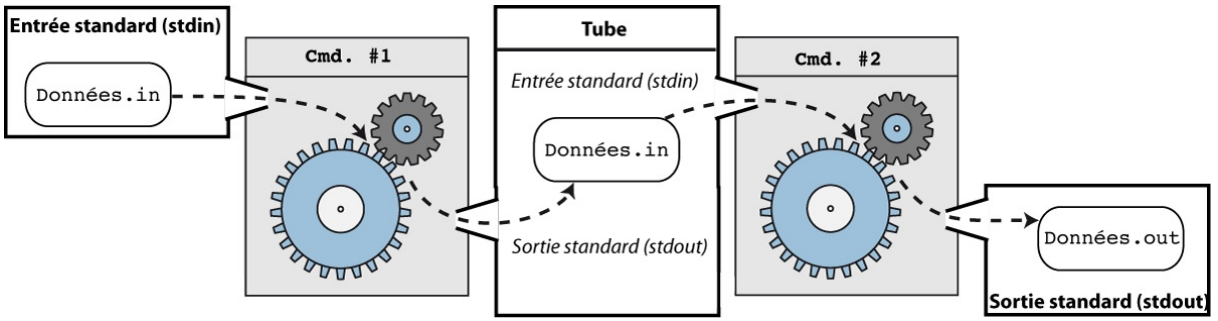

```
[ login@localhost ~ ] ls > 1.out


[ login@localhost ~ ] ls
1.out aldenaran.jpg alphacentauri.gif etacentauri.jpg

[ login@localhost ~ ] ■
```

La sortie standard de la première commande `ls` est redirigée vers le fichier `1.out`. La liste du contenu du répertoire courant est écrite dans le fichier `1.out`.
La deuxième commande `ls`, montre qu'un fichier portant le nom `1.out` a été créé.

Tubes

Principes de fonctionnement des Tubes (Pipe en anglais)

- A la différence des redirections simples qui permettent de rediriger la sortie standard d'une commande vers un fichier,
- Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.



Syntaxe

- Le tube est symbolisé par le caractère |.
- `cmd1 | cmd2`
  - La sortie standard de la première commande (`cmd1`) est redirigée vers l'entrée standard de la deuxième commande (`cmd2`).
  - L'entrée standard de la commande `cmd1` et la sortie standard de la commande `cmd2` ne sont pas modifiées.

Exemple de Tubes avec les commande `ls` et `more`

Rappel des commandes :

- `ls` affiche à l'écran (stdout) la liste des fichiers contenus dans un répertoire.
- `more` affiche page par page le contenu des données passée sur son entrée standard.

Exemple #1

- Si de très nombreux fichiers sont contenus dans un répertoire, la commande `ls` peut produire un affichage qui ne tient pas dans l'écran, rendant impossible le parcours de la liste des fichiers (seuls les derniers sont visibles).

```
[ login@localhost ~ ] ls
```

Défilement de tous les fichiers

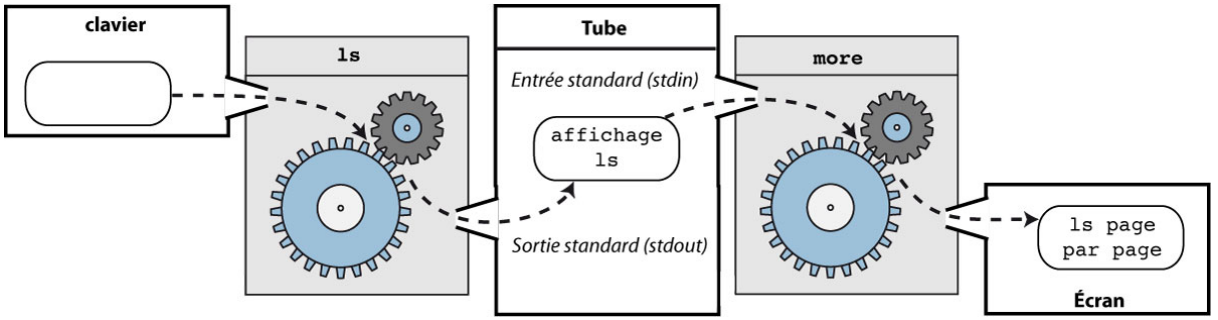
```
betelgeuse.jpg    etacentauri.jpg
soleil.jpg        syrius.gif
vega.png
 [ login@localhost ~ ] ■
```

```
Images/ ..... Répertoire courant
├── aldebaran.jpg ..... Hors de la fenetre
├── alphacentauri.gif ..... Hors de la fenetre
├── betelgeuse.jpg ..... Dans la fenetre
├── etacentauri.jpg ..... Dans la fenetre
├── soleil.jpg ..... Dans la fenetre
├── syrius.gif ..... Dans la fenetre
└── vega.png ..... Dans la fenetre
```

Exemple de Tubes avec les commande `ls` et `more`

Exemple #1 (suite) :

- La redirection de la sortie standard de la commande `ls` vers l'entrée standard de la commande `more` permet de passer en revue l'affichage de la commande `ls` page par page.




```
[ login@localhost ~ ] ls | more
aldebaran.jpg      alphacentauri.gif
betelgeuse.jpg     etacentauri.jpg
soleil.jpg         syrius.gif
```

Affichage d'une première page puis
Presser la touche Enter pour la page suivante

```
soleil.jpg         syrius.gif
vega.png
 [ login@localhost ~ ] ■
```

```
Images/ ..... Répertoire courant
├── aldebaran.jpg ..... Page 1
├── alphacentauri.gif ..... Page 1
├── betelgeuse.jpg ..... Page 1
├── etacentauri.jpg ..... Page 1
├── soleil.jpg ..... Page 1&2
├── syrius.gif ..... Page 1&2
└── vega.png ..... Page 2
```

Exemple de Tubes avec les commande `ls` et `grep`

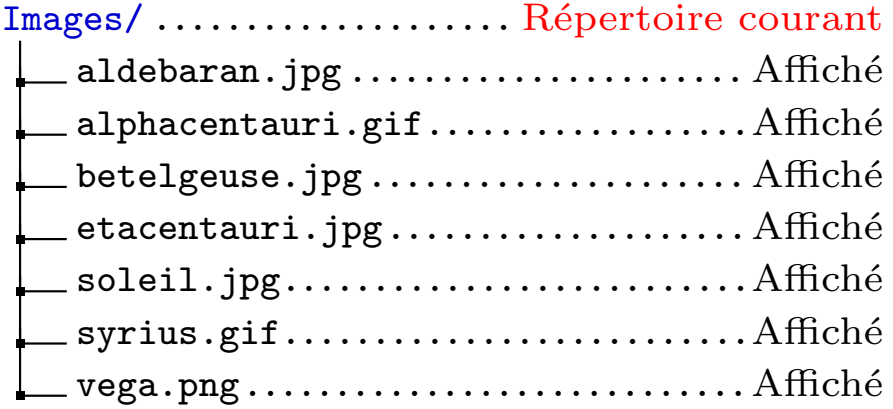
Rappel des commandes :

- `ls` affiche à l'écran (stdout) la liste des fichiers contenus dans un répertoire.
- `grep` affiche les lignes d'un texte qui comportent un certain motif.

Exemple #2 :

- Si de très nombreux fichiers sont contenus dans un répertoire, la commande `ls` peut produire un affichage qui ne tient pas dans l'écran, rendant compliqué l'identification de certain type de fichier (fichiers au format `gif` par exemple).

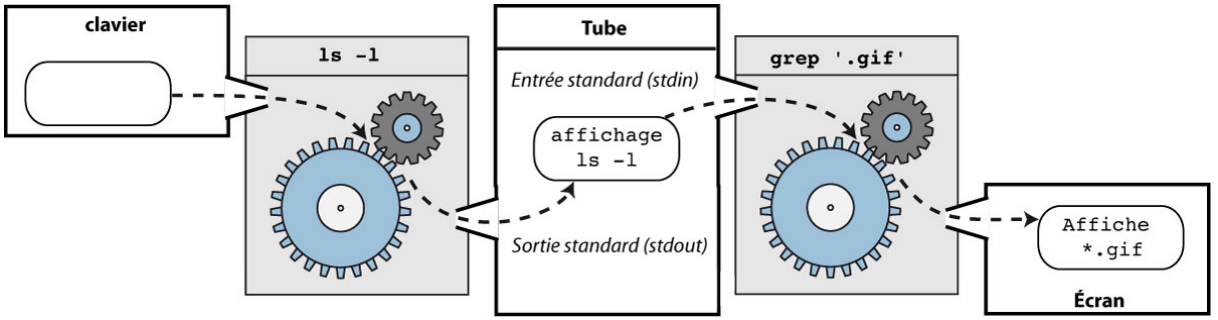
```
[ login@localhost ~ ] ls
aldebaran.jpg
alphacentauri.gif
betelgeuse.jpg
etacentauri.jpg
soleil.jpg
syrius.gif
vega.png

[ login@localhost ~ ] ■
```



Exemple de Tubes avec les commande `ls` et `more`

Exemple #2 (suite) :

- La redirection de la sortie standard de la commande `ls` vers l'entrée standard de la commande `grep` permet d'effectuer un filtrage des fichiers présents dans le répertoire sur la base d'un motif présent dans leur nom (par exemple l'extension `.gif`).




```
[ login@localhost ~ ] ls | grep '.gif'
alphacentauri.gif
syrius.gif

[ login@localhost ~ ] ■
```

```
Images/ ..... Répertoire courant
├── aldebaran.jpg ..... Retenu par le filtre
├── alphacentauri.gif ..... Affiché
├── betelgeuse.jpg ..... Retenu par le filtre
├── etacentauri.jpg ..... Retenu par le filtre
├── soleil.jpg ..... Retenu par le filtre
├── syrius.gif ..... Affiché
└── vega.png ..... Retenu par le filtre
```

Exemple de Tubes avec la commande cut

Rappel des commandes :

- cut permet d'afficher une colonne donnée d'un texte. Pour cela on spécifie le numéro de la colonne et le séparateur de colonnes.

Exemple #3 :

- Extraction d'une colonne d'un fichier formaté dans lequel les séparateurs de colonne ne sont pas homogènes.

Par exemple soit le fichier suivant dont on souhait extraire la liste des planètes :

comparatif.txt

```
Planetes<Nom>Masse(10E+18t)
Tellurique<Mercure>330
Tellurique<Venus>4871
Jovienne<Uranus>86760
Jovienne<Neptune>103000
```

Premier essai :
choisir le séparateur '>' et sélectionner la première colonne :

```
[ login@localhost ~ ] cut -d '>' -f 1 comparatif.txt
Planetes<Nom
Tellurique<Mercure
Tellurique<Venus
Jovienne<Uranus
Jovienne<Neptune

[ login@localhost ~ ] ■
```

C'est un echec ! ! !

Exemple de Tubes avec la commande cut

Rappel des commandes :

- cut permet d'afficher une colonne donnée d'un texte. Pour cela on spécifie le numéro de la colonne et le séparateur de colonnes.

Exemple #3 :

- Extraction d'une colonne d'un fichier formaté dans lequel les séparateurs de colonne ne sont pas homogènes.

Par exemple soit le fichier suivant dont on souhait extraire la liste des planètes :

comparatif.txt

```
Planetes<Nom>Masse(10E+18t)
Tellurique<Mercure>330
Tellurique<Venus>4871
Jovienne<Uranus>86760
Jovienne<Neptune>103000
```

Deuxième essai :
choisir le séparateur '<' et sélectionner la deuxième colonne :

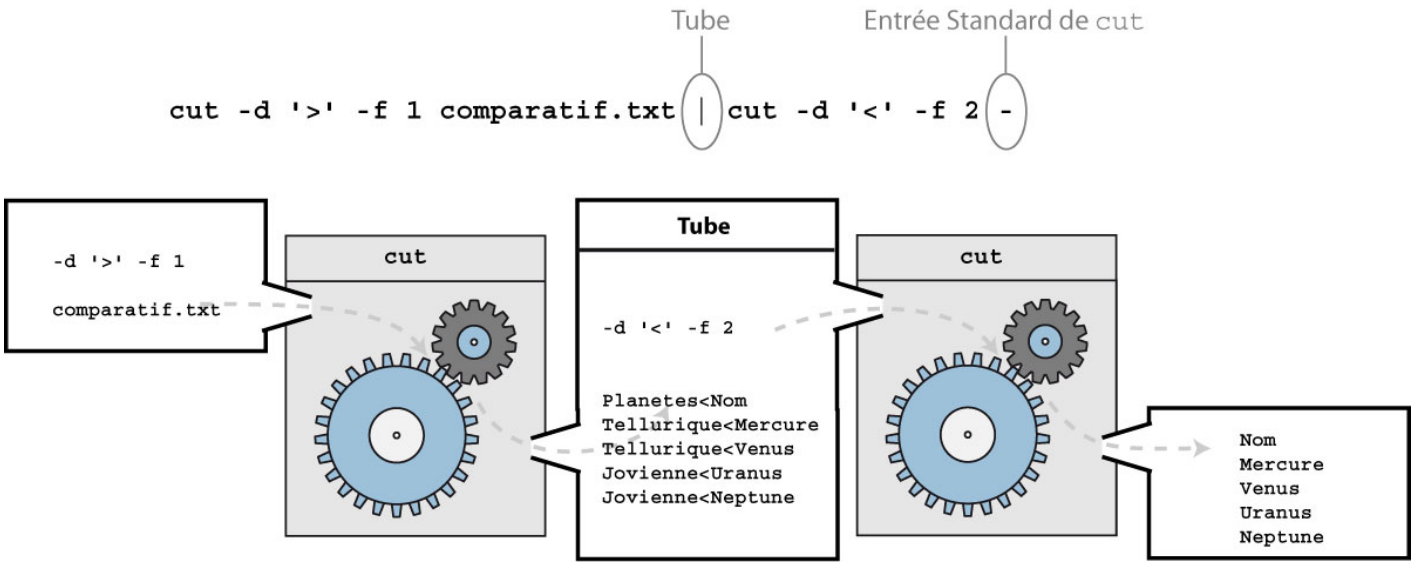
```
[ login@localhost ~ ] cut -d '<' -f 2 comparatif.txt
Nom>Masse(10E+18t)
Mercure>330
Venus>4871
Uranus>86760
Neptune>103000

[ login@localhost ~ ] ■
```

Encore un echec ! ! !

Exemple de Tubes avec la commande cut

La solution : Décomposer en étapes simples et utiliser un tube

- Lors d'une première passe, prendre la première colonne en utilisant le séparateur >, puis
- Lors de la deuxième passe, prendre la deuxième colonne en utilisant le séparateur <.




```
[ login@localhost ~ ] cut -d '>' -f 1 comparatif.txt | cut -d '<' -f 2 -
Nom
Mercure
Vebus
Uranus
Neptune

[ login@localhost ~ ] ■
```

Plan

Introduction

Rappels sur le langage bash

- Langage interprété permettant d'adresser des commandes au système.
- Comme tout langage il est caractérisé par un vocabulaire (le nom des commandes), une grammaire (enchaînement des commandes, des options et des paramètres) et d'une syntaxe (caractères de séparation).
- Il permet à l'utilisateur de définir des programmes pour les besoins spécifiques de certains utilisateurs.

La programmation

Écrire un programme c'est définir un ensemble d'instructions pour réaliser une tâche complexe qu'une seule commande ne peut faire seule. Pour cela il faut pouvoir :

- Regrouper ensemble plusieurs instructions (script),
- Séparer les différentes instructions (séparateurs),
- Définir des variables pour définir des instructions génériques dont le résultat de l'interprétation sera paramétré par les différentes valeurs prises par les variables,
- Exécuter l'ensemble de ces instructions ...

Les variables

Définition

- Une variable est une chaîne de caractères à laquelle est associée une valeur.
- La chaîne de caractères est le nom de la variable.
- Pour faire appel à la valeur il suffit de taper le nom de la variable.
- Lorsque l'on donne un nom de variable comme paramètre à une instruction, celle-ci est remplacée lors de l'interprétation de la commande par sa valeur.
- On appelle cela *une variable*, car la valeur associée à un nom de variable peut changer ...
- On appel *affectation* le fait d'associer une valeur à un nom de variable.

Utilisation

- Les variables sont utilisées pour stocker des valeurs qui doivent être utilisée par la suite, souvent plusieurs fois.
- Elles permettent de définir des instructions dont le comportement sera différent selon la valeur de la variable à l'instant de l'évaluation.

Exemple d'utilisation de variables

Définition et appel d'une variable

- Le signe = permet de réaliser l'affectation selon la syntaxe suivante (Attention : il ne faut pas mettre de caractère espace autour du signe d'affectation '=') :

```
nom_de_la_variable=valeur
```

- Pour accéder au contenu d'un variable, on fait précéder son nom du caractère $.

```
$nom_de_la_variable
```

Se placer dans un répertoire souvent utilisé

Pour sélectionner comme répertoire courant un répertoire dans lequel on se place souvent et dont le nom est long a taper, on peut définir une variable contenant le nom du répertoire, et utiliser le nom de la variable en lieu et place du nom complet du répertoire :

```
[ login@localhost ~ ] DirTP1=/home/chez_moi/TP_Intro_Systeme/TP_1

[ login@localhost ~ ] cd $DirTP

[ login@localhost ~/TP_Intro_Systeme/TP_1 ] ■
```

echo

Syntaxe

```
echo  expression
```

Description

- Affiche sur la sortie standard l'expression après interprétation.

Exemple d'utilisation:

Affiche 'Bonjour' :

```
[ login@localhost ~ ] echo Bonjour
Bonjour
[ login@localhost ~ ] ■
```

Définie une variable puis affiche sa valeur :

```
[ login@localhost ~ ] Astre=Terre
[ login@localhost ~ ] echo $Astre
Terre
[ login@localhost ~ ] echo La planete $Astre
La planete Terre
[ login@localhost ~ ] ■
```

Exemple d'utilisation de variables

Modification de la valeur d'une variable

- Pour modifier la valeur d'une variable il faut lui affecter une nouvelle valeur.
- La syntaxe est la même que celle de la définition de la valeur d'une variable (on utilise le signe d'affectation '=').

Afficher des messages personnalisés

Cet exemple utilise deux variables `Cible` et `Entete`. La même instruction `echo $Entete $Cible` affiche des messages différents car la valeur de la variable `Entete` a été modifiée entre les deux évaluations. Cet exemple montre :

- comment on modifie la valeur d'une variable et
- comment une même instruction paramétrée par des variables peut conduire à des résultats différents selon la valeur des variables.

```
[ login@localhost ~ ] Cible='A tous'

[ login@localhost ~ ] Entete='Bonjour'

[ login@localhost ~ ] echo $Entete $Cible
Bonjour A tous
```

```
[ login@localhost ~ ] Entete='Salut'

[ login@localhost ~ ] echo $Entete $Cible
Salut A tous
[ login@localhost ~ ] ■
```

; Le séparateur de commandes

Écrire plusieurs instructions sur la même ligne de commande

- Il est possible d'écrire plusieurs instructions sur la même ligne de commande.
- Pour séparer les différentes instructions on utilise le caractère de ponctuation ;
- Les instructions seront évaluées dans l'ordre d'écriture, et une instruction ne sera évaluée qu'après que la précédente ait terminée sont évaluation.

Exemple d'instruction multiples sur la même ligne de commande

```
[ login@localhost ~ ] Dir='Mes_Images' ; cd $Dir
~/Mes_Images

[ login@localhost ~/Mes_Images ] echo $Dir ; Dir='~/Mes_Photos'
~/Mes_Images

[ login@localhost ~/Mes_Images ] cd $Dir ; echo $Dir
~/Mes_Photos

[ login@localhost ~/Mes_Photos ] ■
```

Les scripts bash : Présentation

Les scripts bash

- Ce sont des fichiers textes qui regroupent un ensemble d'instructions.
- Ce sont des programmes rédigés et interprétables par le langage bash.
- Ils regroupent dans un même fichier (réutilisable) un ensemble d'instructions qui pourraient être tapées sur la ligne de commande.
- Ces programmes peuvent admettre des paramètres.
- Ils peuvent être lancés depuis la ligne de commande.

mon_premier_script.sh

```
#!/bin/bash

instruction_1
instruction_2
...
...
instruction_n
```

Le nom des scripts

- Le nom donné au fichier texte contenant le script doit être *expressif*.
- Traditionnellement, l'extension donnée à un fichier bash est `.sh`

Alternatives possibles

Le bash est l'interpréteur de commande développé dans le cadre du projet GNU. Il existe d'autres interpréteurs historiques qui sont également disponibles dans la plupart des distributions Linux.

- csh, tcsh, ksh, zsh, ...

Les scripts

Structure d'un script

- Sur la première ligne du script on place une entête indiquant le chemin de l'interpréteur `#!/bin/bash` suivit d'une ligne vide.
- Dans le script, chaque ligne correspond à une instruction,
- Sur une ligne, tout texte placé après le caractère # est considéré comme un commentaire et n'est pas interprété.

mon_premier_script.sh

```
#!/bin/bash

message="Aller au répertoire : "      # Définition de la variable message
dir=~/Intro_Systeme/                  # Définition de la variable dir
echo $message $dir                    # Affichage de l'action
cd $dir                               # Changement de répertoire courant
```

Évaluation des commandes du script

- Les instructions d'un script sont évaluées dans leur ordre d'écriture. Par exemple, ici, la commande `echo $message $dir` est évaluée avant la commande `cd $dir`.
- Une commande d'un script n'est évaluée qu'après la fin de l'exécution de la commande précédente - *i.e.* évaluation séquentielle des commandes.

Les scripts

Cas d'étude

Soit le script `mon_premier_script.sh` et son emplacement dans l'arborescence des fichiers :

mon_premier_script.sh

```
#!/bin/bash

message="Aller au répertoire : "
dir=~/Intro_Systeme/
echo $message $dir
cd $dir
```

```
chez_moi/ ..... Répertoire Courant
├── mes_scripts/
│   └── mon_premier_script.sh ..... Le script
└── Intro_Systeme/
    ├── TP_1/
    │   └── ...
    └── TP_2/
        └── ...
```

Exécuter un script (1)

- Pour rendre un script *exécutable*, il faut donner les droits d'exécution sur le fichier (`chmod`).

```
[ login@localhost ~ ] chmod u+x ~/mes_scripts/mon_premier_script.sh

[ login@localhost ~ ] ls -l ~/mes_scripts/mon_premier_script.sh
 -rwxr--r-- 2 santini ensinfo 93 18 jul 12 :59 mon_premier_script.sh

[ login@localhost ] ■
```

Les scripts

Cas d'étude

Soit le script `mon_premier_script.sh` et son emplacement dans l'arborescence des fichiers :

mon_premier_script.sh

```
#!/bin/bash

message="Aller au répertoire : "
dir=~/Intro_Systeme/
echo $message $dir
cd $dir
```

```
chez_moi/.....Répertoire Courant Initial
├── mes_scripts/
│   └── mon_premier_script.sh ..... Le script
└── Intro_Systeme/..Répertoire Courant Final
    ├── TP_1/
    │   └── ...
    └── TP_2/
        └── ...
```

Exécuter un script (2)

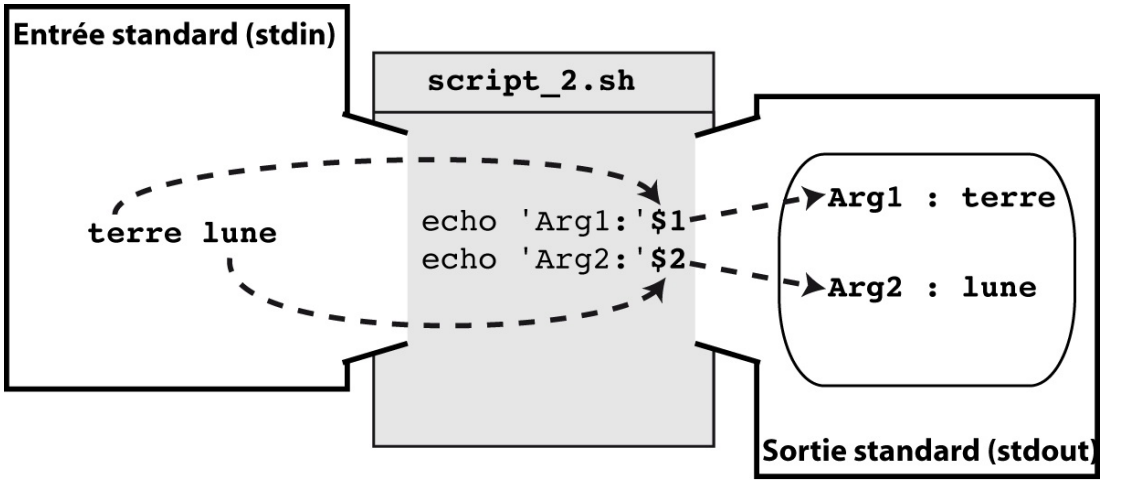
- Pour exécuter un script il suffit ensuite de taper son chemin -*i.e.* le chemin vers le fichier contenant le script-.

```
[ login@localhost ~ ] mes_scripts/mon_premier_script.sh
Aller au répertoire :~/Intro_Systeme

[ login@localhost ~/Intro_Systeme ] pwd
~/Intro_Systeme

[ login@localhost ~/Intro_Systeme ] ■
```

Les paramètres des scripts

Passage des paramètres

- Dans un script, il est possible de récupérer des informations passées sur la ligne de commande.
- Ces informations sont passées au script comme des variables.
- Ces variables peuvent être utilisées dans les instructions.




```
[ login@localhost ~ ] ./script_2.sh terre lune
Arg1 : terre
Arg2 : lune

[ login@localhost ~ ] ■
```

Appel de la variable	Valeur de la variable
$0	Le nom du script
$1 à $9	Les (éventuels) premiers arguments passés à l'appel du script
$#	Le nombre d'arguments passés au script
$*	La liste des arguments à partir de $1

Les paramètres des scripts

Exemple de scripts avec passage de paramètres

Soit le script suivant :

affiche_param.sh

```
#!/bin/bash

echo "Nom du script :" $0
echo "Nombre de parametres :" $#
echo "Premier parametre :" $1
echo "Deuxieme parametre :" $2
```

Son exécution produit les affichages suivants :

```
[ login@localhost ~ ] ./affiche_param.sh terre lune soleil
Nom du script : affiche_param.sh
Nombre de parametres : 3
Premier parametre : terre
Deuxieme parametre : lune

[ login@localhost ~ ] ■
```

Plan

Caractères spéciaux

Définition

Ce sont des caractères qui :

- peuvent apparaitre sur la ligne de commande,
- prennent un sens particulier lors de l'interprétation

Caractères spéciaux déjà vus

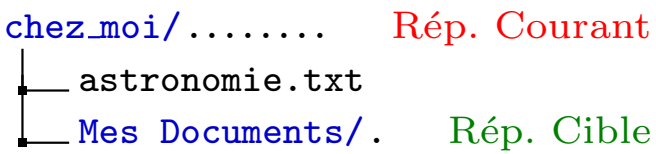
Caractère	Interprétation
>	Redirection de la sortie standard d'une commande.
<	Redirectiond e l'entrée standard d'une commande.
|	Tube : redirection de la sortie standard d'une commande vers l'entrée standard d'une autre commande.
*	Complétion d'un nom de fichier
$	Appel de la valeur d'une variable
;	Fin de commande (permet de taper plusieurs commandes sur la même ligne)
␣	Séparateur de paramètres
"	Délimiteur de texte
#	Commentaires

Caractères spéciaux

L'échappement

- Permet d'échapper au sens donné aux caractères spéciaux par l'interpréteur.
- Redonne leur sens littérale aux caractères spéciaux.

Exemple : Taper un nom de répertoire contenant un espace




```
[ login@localhost ~ ] cd Mes Documents
-bash : cd : Mes : No such file or directory

[ login@localhost ~ ] ■
```

Problème

- Le répertoire `'Mes Documents'` n'est pas trouvé,
- Le caractère 'espace' compris dans le nom du répertoire est interprété comme un séparateur de paramètre.
- Le répertoire `'Mes'` n'existe pas dans le répertoire courant...

Échapper à l'interprétation

Plusieurs modes d'échappement

Échappement	Caractère	syntaxe
Ponctuel	Back Slash	`cd Mes\ Documents`
Complet	Guillemets Simples	`cd 'Mes Documents'`
Partiel	Guillemets Doubles	`cd "Mes Documents"`

Fonctionnement

Ponctuel Le caractère qui suit le Back Slash (\) échappe à l'interprétation (reprend son sens littérale).

Complet Tous les caractères compris entre les Guillemets Doubles échappent à l'interprétation (reprennent leur sens littérale).

Partiel Tous les caractères compris entre les Guillemets Doubles échappent à l'interprétation (reprennent leur sens littérale), sauf le caractère $. Du coup, les noms de variables sont interprétés et remplacé par leur valeur.

Échapper à l'interprétation

Exemples d'échappement Ponctuel

```
[ login@localhost ~ ] PLANETE=Terre

[ login@localhost ~ ] echo $PLANETE # $PLANETE
Terre
[ login@localhost ~ ] echo \$PLANETE \# $PLANETE
$PLANETE # Terre
```

Exemples d'échappement Partiel et Complet

```
[ login@localhost ~ ] PLANETE=Terre

[ login@localhost ~ ] echo "$PLANETE # $PLANETE"
Terre # Terre
[ login@localhost ~ ] echo '$PLANETE # $PLANETE'
$PLANETE # $PLANETE
```