

# Programmation et projet encadré - L7TI005

## Shell Unix

Crédits supports : Serge Fleury

---

Yoann Dupont, Serge Fleury [prenom.nom@sorbonne-nouvelle.fr](mailto:prenom.nom@sorbonne-nouvelle.fr)

Pierre Magistry [pierre.magistry@inalco.fr](mailto:pierre.magistry@inalco.fr)

2022-2023

Université Sorbonne-Nouvelle

INALCO

Université Paris-Nanterre

# Introduction



## Existence et importance du système de fichiers

*Tout est fichier*

- dans les années 60, ce n'était pas évident,
- même les programmes, les périphériques, le réseau, ...
- simplicité → on ne pense qu'en termes de fichiers et dossiers,
- d'où l'importance de bien les organiser !

## Existence et importance du système de fichiers

*Tout est fichier*

- dans les années 60, ce n'était pas évident,
- même les programmes, les périphériques, le réseau, ...
- simplicité → on ne pense qu'en termes de fichiers et dossiers,
- d'où l'importance de bien les organiser !

## Un ensemble de commandes pour manipuler les fichiers et leur contenu.

(depuis **la ligne de commandes**)

- Chaque commande doit faire **une** chose et la faire **bien** (K.I.S.S.)
- Chaque commande est pensée pour pouvoir interagir avec les autres.
- On fait des choses complexes en combinant des commandes simples.  
(*pipelines* ou *scripts*)

# Le Système de fichier



## Points clefs

- les **fichiers** sont dans un **dossier**
- pouvant eux même être dans un **dossier** « parent »
- ce qui forme une **arborescence**

L'ensemble des données sur la machine sont regroupées en une seule **arborescence**, il est intéressant de pouvoir identifier les fichiers et les dossiers par un **chemin** dans l'arbre.

/ désigne la **racine** de l'arbre

~/ désigne le dossier personnel de l'utilisateur ( "*HOME*")

./ désigne le dossier courant (*working directory*)

../ désigne le dossier parent

un **chemin** est formé par une suite de noms de dossiers séparés par des /, pouvant se finir par le nom d'un fichier.

**un chemin absolu** indique la position d'un fichier en partant de la **racine**.

ex: `/home/pierre/PPE1`

**un chemin relatif** indique la position d'un fichier en partant du dossier courant.

ex: `../../dev/input/mouse3`

→ habituez vous à toujours être capable de donner le chemin relatif et absolu vers les fichiers que vous manipulez.



## ou « caractères de remplacement »

Dans un chemin, caractères ? et \* ont un comportement spécial.

- ? peut remplacer n'importe quel caractère (unique)

- \* peut remplacer n'importe quelle suite de caractères

→ le chemin peut alors désigner plusieurs fichiers !

À tester avec la commande `ls`.

# Les Commandes

---

Les commandes sont des fichiers comme les autres,

- qui ont la propriété d'être exécutables
- qui sont placés par convention dans un dossier où le système sait les trouver (typiquement `/usr/bin/`)

# La syntaxe d'une commande

nom [-options...] [arguments...]

- les **options** peuvent avoir une forme courte (-o avec un seul tiret)
- ou une forme longue (--option avec deux tirets).
- Les **arguments** sont typiquement des chemins vers des fichiers (mais pas toujours).
- La première action de l'interpréteur de commande est de *tokeniser* la ligne pour découper le nom de la commande, les options et les arguments.

→ habituez vous à anticiper comment les lignes de commandes que vous saisissez vont être découper et à repérer nom, options et arguments, comme l'interpréteur

## Quelques commandes à connaître

`cd, ls, pwd, cat, less, wc, echo, head, tail, mkdir, cp, mv, rm, rmdir, file...`

Et surtout: `man`

La plupart des commandes ont aussi une option `--help` qui permet d'obtenir une description concise du fonctionnement et des options de la commande

# Se promener dans l'arbre

**pwd** *print working directory*

**ls** *list* le contenu d'un dossier

**cd** *change directory*

**cp** *copy*

**mv** *move* déplacer

**rm** *remove* supprimer

**mkdir** *make directory* créer un dossier

**touch** crée un fichier (effet de bord bien pratique)

**file** donne des informations sur le type de fichier

**cat** lit le contenu d'un ou plusieurs fichiers

**head** lit le début d'un fichier

**tail** lit la fin d'un fichier

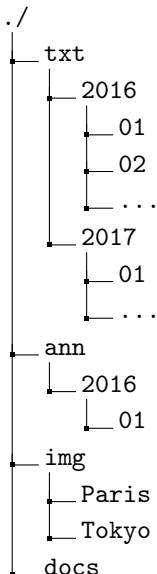
**cut** sélectionne une ou plusieurs colonnes dans un fichier tabulé

**less** lecteur (interactif)



- Télécharger l'archive fichier.zip ...
- créer un dossier de travail où la décompresser
- créer une arborescence pour classer les documents
  - par type de fichier
  - puis par date pour
  - puis par lieux pour les photos.

# Exercice



## Consignes

- Télécharger l'archive fichier.zip ...
- créer un dossier de travail où la décompresser
- créer une arborescence pour classer les documents
  - par type de fichier
  - puis par date pour
  - puis par lieux pour les photos.(Voir ci-contre)
- copier les fichiers dans les dossiers correspondants

# Les Pipelines



## Généralités

Toutes les commandes communiquent via trois flux de données

**stdin** l'entrée standard (par défaut le clavier)

**stdout** la sortie standard (par défaut l'écran)

**stderr** la sortie d'erreurs standard (par défaut l'écran)

Par défaut, les commandes échangent dans le terminal via le clavier et l'écran, mais on peut **rediriger** ces flux.

## Redirections vers et depuis des fichiers

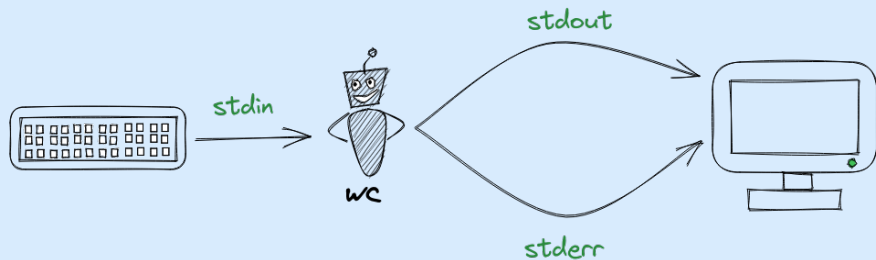
- < remplace le clavier par le contenu d'un fichier
- > écrit stdout dans un fichier
- 2> écrit stderr dans un fichier
- >& écrit stdout et stderr dans un fichier

En écriture, si on double le chevron (», »&, 2»), on écrit en ajoutant la sortie à la fin d'un fichier.

ATTENTION: les chevrons simples (>, >&, 2>) écrasent le fichier si il existe déjà.

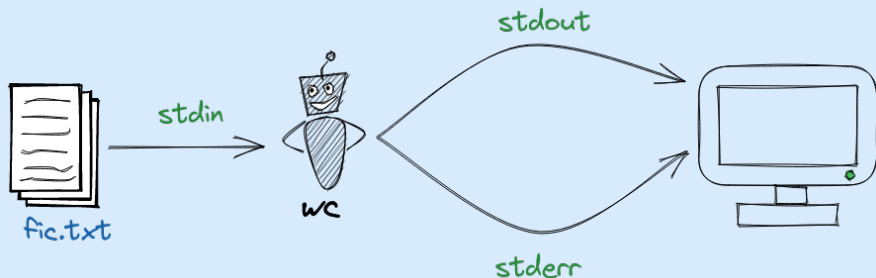
## WC

Situation par défaut



```
wc < fic.txt
```

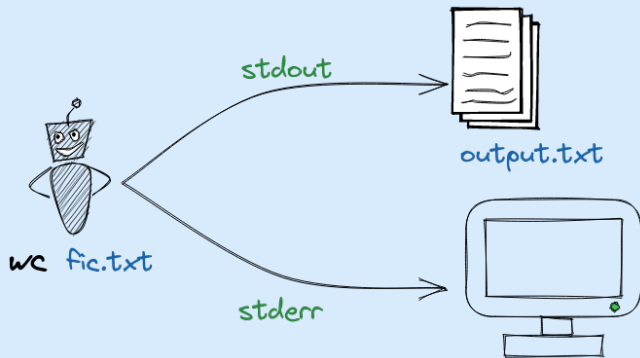
Redirection du contenu d'un fichier dans stdin



ATTENTION: ici `fic.txt` n'est PAS un argument.

```
wc fic.txt > output.txt
```

Redirection de stderr dans un fichier.



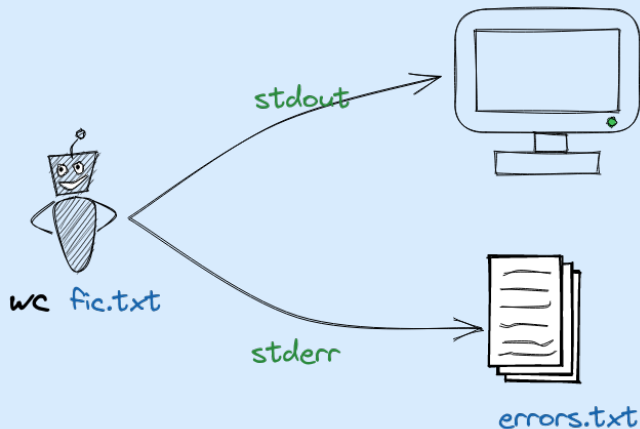
ATTENTION: ici `fic.txt` est donné en argument (et `wc` ignore l'entrée standard).

La sortie standard reste l'écran par défaut.



```
wc fic.txt 2> errors.txt
```

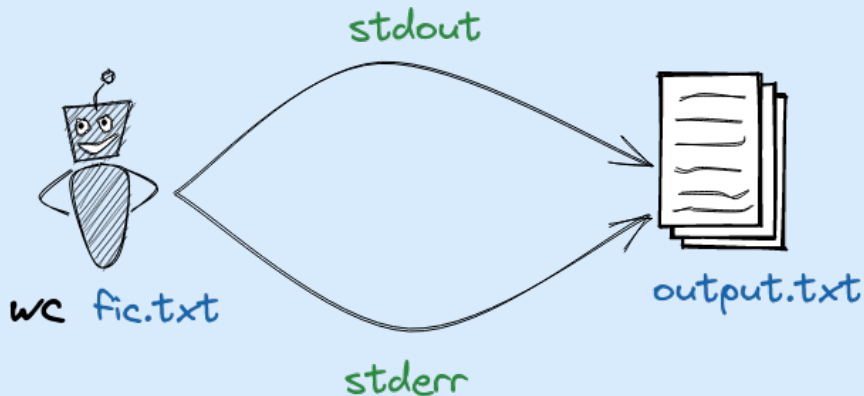
Redirection de stderr dans un fichier.



ATTENTION: ici `fic.txt` est donné en argument (et `wc` ignore l'entrée standard).

```
wc fic.txt >& output.txt
```

Redirection des sorties `stdin` et `stdout` dans un fichier.



ATTENTION: ici `fic.txt` est donné en argument (et `wc` ignore l'entrée standard).

## Redirections entre les commandes

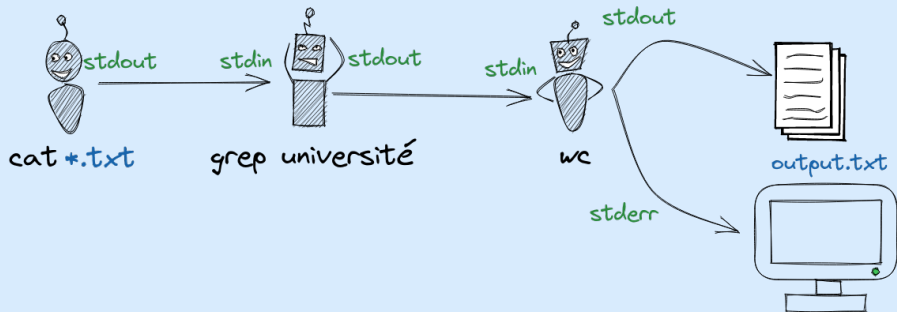
Plusieurs commandes peuvent communiquer en connectant les sorties aux entrées aux moyen du caractère «pipe» |

`cmd1 | cmd2` la sortie standard (stdout) de `cmd1` est envoyée en stdin de `cmd2`.

`cmd1 |& cmd2` les sorties stdout et stderr de `cmd1` sont toutes deux redirigées vers stdin de `cmd2`.

```
cat *.txt | grep université | wc fic.txt 1> output.txt
```

Redirection de stderr dans un fichier.



## Quelques commandes de plus

**grep** recherche de motifs dans l'entrée (ou dans des fichiers)

**sort** trier des lignes

**uniq** supprimer les lignes qui se répètent

## avec les fichiers \*.ann, Sauriez vous... ?

- Compter le nombre d'annotations par année (2016, 2017 et 2018),
- limiter ce comptage aux lieux (Location),
- sauvegarder ces résultats dans un (seul) fichier,
- établir le classement des lieux les plus cités.
- trouver les annotations les plus fréquentes pour votre mois de naissance, toutes années confondues.