

# Cultures numériques avancées

## Structure des répertoires. Commandes. Variables.

Ljudmila PETKOVIC

[ljudmila.petkovic@sorbonne-nouvelle.fr](mailto:ljudmila.petkovic@sorbonne-nouvelle.fr)

```
~/Documents/linux-commands via v3.9.6
> ls -lah
Permissions Size User Date Modified Name
drwxr-xr-x - daniel 8 ago 15:11 .
drwxr-xr-x - daniel 8 ago 00:27 ..
drwxr-xr-x - daniel 8 ago 00:34 commands
drwxr-xr-x - daniel 7 ago 00:45 dir1
drwxr-xr-x - daniel 7 ago 00:45 dir2
drwxr-xr-x - daniel 8 ago 00:10 dir_to_copy
drwxr-xr-x - daniel 8 ago 00:12 new_dir
-rw-r--r-- 0 daniel 8 ago 00:38 BestMoviesOfAllTime
-rw-r--r-- 0 daniel 7 ago 00:44 binarysearch.py
-rw-r--r-- 0 daniel 7 ago 00:43 dummyfile1.txt
-rw-r--r-- 0 daniel 8 ago 00:18 file_to_delete.txt
-rw-r--r-- 0 daniel 7 ago 00:44 get_keys.py
-rw-r--r-- 0 daniel 7 ago 00:44 github_automation.py
-rw-r--r-- 0 daniel 7 ago 00:44 important_file.txt
-rw-r--r-- 0 daniel 8 ago 00:04 new_file.txt
-rw-r--r-- 0 daniel 12 abr 20:45 old_file
```

Source : [Diaz, 2023](#).

Cultures numériques avancées (L2HN001)

Mineure « Humanités numériques », licence Lettres

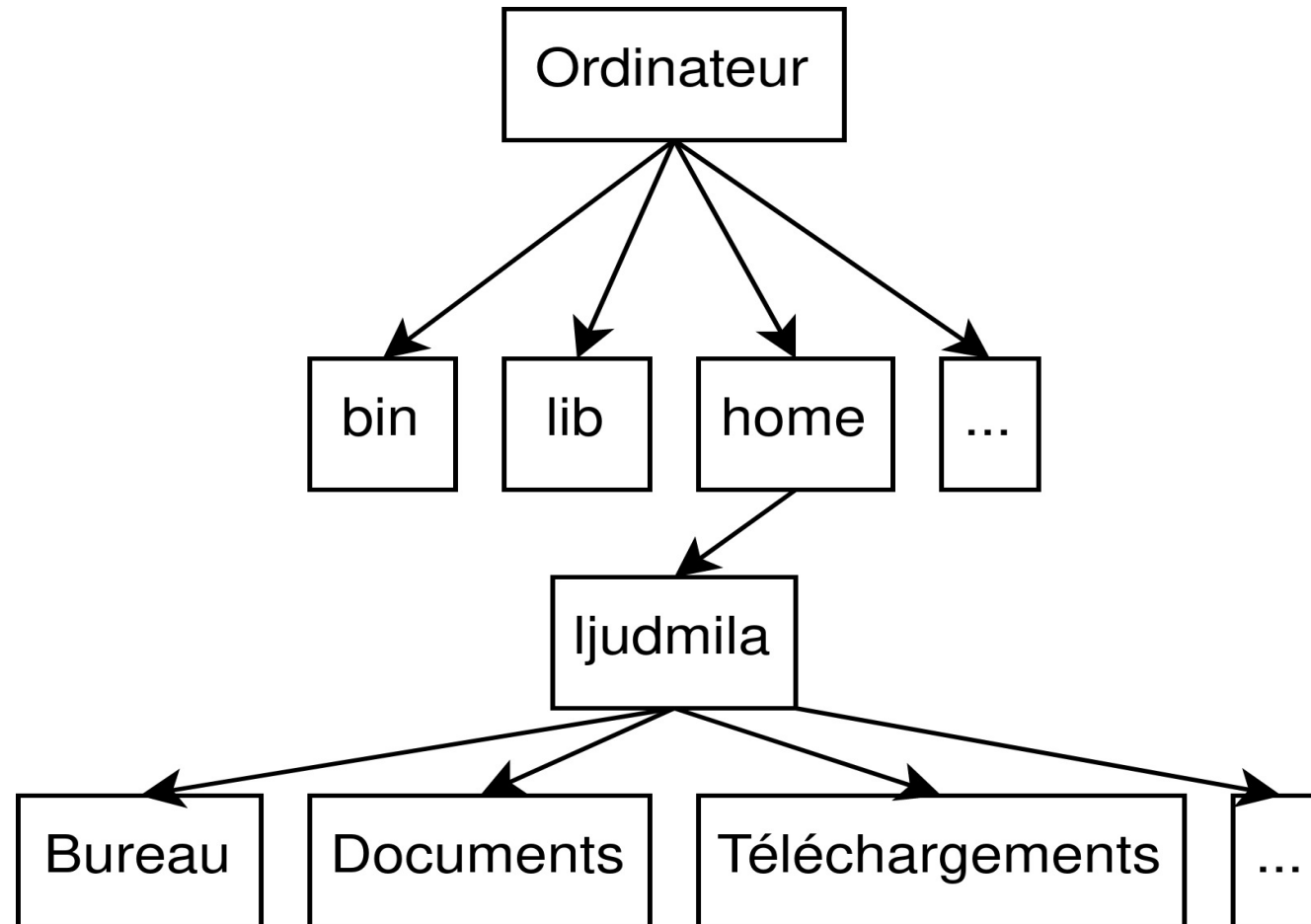
Paris, le 2 février 2024, année 2023-2024

Diapositives adaptées de [Simon Gabay](#), [Simone Rebora](#) et [Élisabeth Brunet et Gaël Thomas](#).

# Structure des répertoires

- façon dont un système d'exploitation organise les fichiers / répertoires accessibles à l'utilisateur
- les fichiers sont généralement affichés dans une arborescence hiérarchique
- angl. *Filesystem Hierarchy Standard*: « norme de la hiérarchie des systèmes de fichiers »
  - GNU/Linux et la plupart des systèmes Unix

# Structure des répertoires



*Filesystem Hierarchy Standard.*

# Fonctions des répertoires

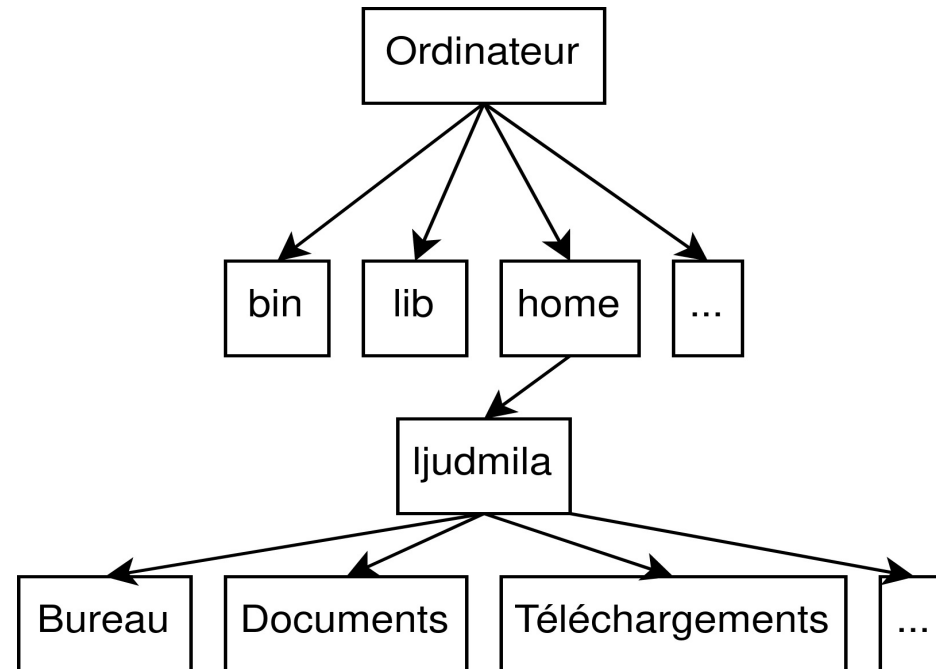
- `/bin/` : commandes de base nécessaires au démarrage et à l'utilisation d'un système minimaliste
- `/etc/` : *editable text configurations* ou fichiers de configuration
- `/lib/` : bibliothèques (*librairies*) logicielles nécessaires pour les exécutables
- `/home/` : répertoire des utilisateurs
- ...

Les répertoires de chaque utilisateur sont eux connus (fichiers de système cachés)

- Bureau
- Documents
- Téléchargements
- ...

# root

Les répertoires sont organisés comme un arbre :



- le premier répertoire (ici `Ordinateur`) est appelé le répertoire *racine* (`root`)
- il contient tous les autres, organisés comme des branches partant de ce tronc

## Chemin absolu

L'adresse du répertoire racine est `/`. Celle d'un répertoire ou d'un fichier précis est ainsi la liste des répertoires depuis `root` pour accéder à celui voulu, chaque nom étant séparé avec un `/`. Ainsi, pour atteindre le fichier `bash` dans le dossier `bin`, je suis le chemin **absolu** : `/bin/bash`.

## Chemin relatif

Le chemin que nous venons de voir est dit *absolu*, car il part de l'élément racine, mais il existe aussi des chemins **relatifs** si nous partons d'un autre endroit (normalement celui où nous sommes déjà) : `ljudmila/home/bin/bash`.

## Chemin relatif

Parfois il est impossible de savoir quel est le nom du fichier précédent dans l'arborescence, il est possible d'utiliser un raccourci : `..` signifie ainsi « remonter d'un dossier » :

```
ljudmila/home/bin/bash
```

est ainsi l'équivalent de

```
ljudmila/../bin/bash
```

# Conventions de nommage

1. Il est fortement déconseillé d'utiliser des espaces
2. Des stratégies alternatives existent comme:
  - 2.1 Le camelCase (par exemple : `nomDeFichier.extension` )
  - 2.2 Avoir recours à des tirets ( `nom-de-Fichier.extension` ) ou des tirets bas ( `nom_de_Fichier.extension` )
  - 2.3 Tentez d'être cohérent dans cette stratégie
3. Versionnez les documents ( `nom-de-Fichier-v1.extension` ) ou datez-les en commençant par l'année ( `nom-de-Fichier-AAA-MM-JJ.extension` )
4. L'extension doit être choisie avec attention : un `.txt` n'est pas un `.xml`



# Fonctionnement de Bash

- Bash exécute les instructions ligne par ligne : la fin de ligne est la fin de commande
- une commande doit être complète, sinon elle ne s'exécute pas
- une commande est appelée par son nom (par exemple `pwd` ), qui permet de retrouver la fonction (angl. *builtin*), le programme associé
  - une fonction est un bloc de commandes qui s'exécute lorsque la fonction est appelée
  - un *builtin* (« préconstruit », comprendre « prédéfini ») est une mini-opération pré-construite en bash (dont `pwd` )
  - un programme est un groupe d'instructions

# Localisation

Certaines commandes prédéfinies sont enregistrées dans la machine (comme `pwd` ou `ls`) : leur nom suffit pour les appeler.

```
pwd
```

```
ls
```

Dans d'autres cas, comme celui de scripts ou de programmes, il faut spécifier le chemin ou le fichier se trouve. Pour cela on utilise le chemin absolu ou (ici) relatif :

```
commandes/commande_1.sh
```

# Affichage des fichiers / répertoires

`ls` : lister les noms des fichiers et des répertoires *visibles* dans le répertoire courant

`ls -l` : utiliser un format de liste longue (avec les indications des permissions pour chaque fichier)

- `-l` est une *option*

## Argument

Certaines commandes vont nécessiter des précisions : copier *ceci*, aller *là-bas*. Pour donner ces précisions, on va ajouter à la commande des arguments.

Prenons l'exemple de la commande `cd` (*change directory*) qui permet de se déplacer. On pourrait la traduire par « aller à » [commande] + lieu [argument]. Le lieu où l'on se dirige prend la forme du répertoire-destination placé juste après la commande.

```
cd commandes
```

# Changer le répertoire

```
cd Documents # aller dans le répertoire 'Documents'
```

```
cd ../.. # remonter de deux dossiers
```

```
cd . # rester dans le même répertoire
```

# Arguments

Parfois on peut avoir besoin de plusieurs arguments. On les ajoute ainsi les uns après les autres.




- commande `cp` (*copy*), que l'on peut traduire par « copier » [commande] + tel chose [argument 1] + à tel endroit [argument 2] :

```
cp commandes/test.sh ..
```

- commande `rm` (*remove*) qui permet d'effacer un fichier :

```
rm test.sh
```

# Astuces pour naviguer dans le système des répertoires

- Faire glisser des éléments dans une fenêtre Terminal afin d'entrer le chemin absolu d'un fichier / répertoire
- Ouvrir le terminal à partir du répertoire souhaité
  - clique droite sur le répertoire > `Ouvrir dans un terminal`
- Flèche haut  et bas  pour se déplacer dans l'historique du terminal
- Tabulation (  ) pour l'auto-complétion

# Options

- une option modifie le comportement d'une commande
- elle est placée après la commande et est précédée d'un ( `-` ) ou de deux tirets ( `--` )
  - `ls -a` (ou `ls -Force` sous Windows) : lister les noms des fichiers et des répertoires *cachés* dans le répertoire courant, commençant par un point, p.  
ex. `.fichier_cache.txt`
    - raccourci pour afficher les fichiers cachés : `Ctrl + H` (Linux) ou `(fn) + Cmd + Shift + point` (Mac)
  - `git --version` : vérifier si Git avait été bien installé et si oui, quelle version a été installée



# Copier le fichier

La commande `mkdir` (*make directory*) permet de créer un répertoire :

```
mkdir test
```

La commande `cp` (*copy*) permet de copier un fichier :

```
cp mon_script.sh test
```

Déplacer une multitude de fichiers en les mettant à la suite :

```
cp FICHER_1 FICHER_2 RÉPERTOIRE_CIBLE
```

Copier un fichier dans le même répertoire en lui attribuant un nouveau nom :

```
cp test/mon_script.sh test/mon_script_2.sh
```

# Déplacer le fichier

Une alternative à la commande `cp` est la commande `mv` (*move*) qui permet de déplacer (et non copier) un fichier :

```
mv mon_script.sh test
```

Son fonctionnement est proche de `cp` :

```
mv FICHIER_1 FICHIER_2 RÉPERTOIRE_CIBLE
```

Si tous les fichiers ont la même extension, il est possible d'utiliser un joker ( `*` ) :

```
mv *.sh RÉPERTOIRE_CIBLE
```

# Effacer

La commande `rm` (*remove*) permet d'effacer un fichier, avec l'option `-f` pour forcer l'exécution si besoin :

```
rm mon_script.sh
```

Pour effacer un répertoire contenant des fichiers, il faut utiliser :

- l'option `-r` (*recursively*) qui permet d'effacer tous les fichiers contenus l'un après l'autre
- l'option `-f` (*force*) pour éviter d'avoir à valider pour chaque fichier

```
rm -rf test
```

# Rechercher

Faire des recherches dans un fichier : `grep` ( `Select-String` pour Windows) :

```
grep "ordinateur" fichier_test.txt
```

Nous pouvons faire des requêtes en utilisant les expressions régulières (*regex*).

Trouvons tous les mots commençant par la lettre « m », en majuscule ou en minuscule.

```
grep -Eoi "\bm\w+" fichier_test.txt
```

`-E` : expression rationnelle étendue (≠ expression rationnelle simple `-G`)

`-o` : n'afficher que l'occurrence en question (*match*)

`-i` : trouver le mot en majuscule ou en minuscule

`\b` : limite de mot, c'est-à-dire le début d'un mot

`\w+` : un ou plusieurs caractères alphabétiques

# Variables

Comme tout langage de programmation, Bash permet aux programmeurs de stocker de l'information dans des objets appelés **variables** (on déclare les variables).

Exemple d'une variable :

```
nom="Michel"
```

Les variables se caractérisent par :

- leur nom :
  - chaque variable à un nom et ce nom est unique (il n'existe pas deux variables ayant le même nom dans un même programme)
  - ce nom permet de savoir de quelle variable on parle.
- leur type : nombre, chaîne de caractères, liste...

## Créer une variable

Dans la séquence *Coucou* + *nom* si *nom* doit pouvoir changer il s'agit d'une variable. Cette dernière est stockée sous un nom arbitraire :

```
nom="Michel" # ne pas séparer la variable du signe « égal à » (nom = "Michel")
```

et appelée avec son nom précédée de **\$** :

```
echo "Coucou $nom"
```

# Références

- **Brunet, É. & Thomas, G.** (s. d.). « Le shell bash » [*diapositives*]. Télécom SudParis. <http://www-inf.telecom-sudparis.eu/cours/CSC3102/Supports/ci1-bash/ci-bash.pptx.pdf>
- **Combeau, M.** (2022). « La différence entre le terminal, la console et le shell ». <https://www.codequoi.com/difference-entre-terminal-console-et-shell/>
- **Rebora, S.** (2022). « Connaître son propre ordinateur » [*dépôt GitHub*]. EnExDi2022. [https://github.com/ABC-DH/EnExDi2022/tree/main/materials/1\\_KnowYourComputer/slides](https://github.com/ABC-DH/EnExDi2022/tree/main/materials/1_KnowYourComputer/slides)
- **Gabay, S.** (2022). « Les lignes de commandes et Bash » [*dépôt GitHub*]. Université de Genève, Chaire des humanités numériques, Faculté des Lettres. [https://github.com/gabays/Fondamentaux/blob/main/Lignes\\_de\\_commandes/DistRead\\_1\\_2.pdf](https://github.com/gabays/Fondamentaux/blob/main/Lignes_de_commandes/DistRead_1_2.pdf)
- **Diaz, D.** (2023). « Les 40 commandes Linux les plus utilisées que vous devez connaître ». Kinsta. <https://kinsta.com/fr/blog/commandes-linux/>
- **Xyoos** (s.d.). « Interface graphique ». <https://cours-informatique-gratuit.fr/dictionnaire/interface-graphique/>