

Programmation Système

Cours 2: Système de fichiers

Khaddouja ZELLAMA

Khaddouja.zellama@dauphine.psl.eu

Licence L2 Mathématiques et Informatique
Département MIDO

Mode texte / mode graphique

L'utilisation d'un ordinateur sous Unix se fait

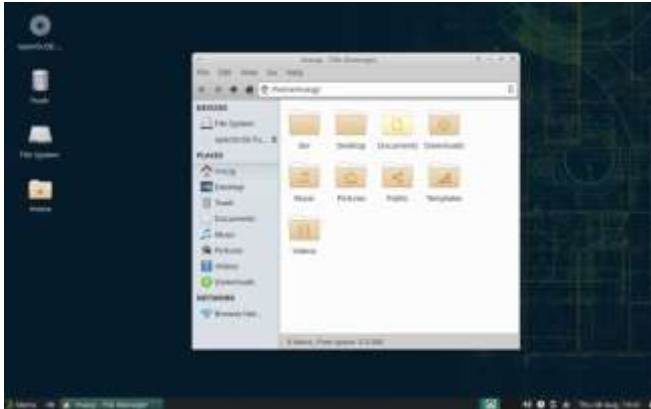
- soit en mode texte (commandes)
- soit en mode graphique (interface graphique).

Mode texte / mode graphique

L'utilisation d'un ordinateur sous Unix se fait

- soit en mode texte (commandes)
- soit en mode graphique (interface graphique).

Interface graphique (Xfce):



Mode texte / mode graphique

L'utilisation d'un ordinateur sous Unix se fait

- soit en mode texte (commandes)
- soit en mode graphique (interface graphique).

Interface graphique (Unity):



Le plus souvent : interface graphique + terminal

Session Unix

Toute session Unix (graphique ou texte):

- commence par une connexion (login)
 - ▶ permet d'authentifier l'utilisateur
- termine par une déconnexion (logout)

Au cours de la session

- Les fichiers créés
- Les programmes exécutés

appartiennent à l'utilisateur (et à un groupe)

Cela permet:

- D'assurer une distribution équitable des ressources
Quotas d'espace disque ou de temps CPU
- De contrôler l'accès aux données de l'utilisateur
Via un système de permissions sur les fichiers

Quelques commandes de base

Connaître son nom d'utilisateur

```
1 $ whoami
```

Connaître la liste des utilisateurs connectés

```
1 $ who
```

Obtenir des informations sur un fichier (e.g. son propriétaire)

```
1 $ ls -l test.c  
2 -rw-r--r-- 1 bnegreve bnegreve 253 May 18 2020 test.c
```

Terminer une session

```
1 $ exit
```

Repertoire home

- Chaque utilisateur dispose d'un répertoire home dans lequel il se trouve au début de la session. (souvent /home/utilisateur)
- Le repertoire contient les fichiers de l'utilisateur
- Les utilisateurs sont libres d'organiser leur repertoire home comme ils le souhaitent
- Il est possible d'accéder aux fichier, via l'interface graphique, ou via des commandes tapées dans le terminal.
 - Il s'agit de la même hiérarchie de fichiers!

Parcourir son répertoire home

Lister le contenu d'un répertoire

```
1 $ ls # rien si le repertoire est vide
```

Créer un sous-répertoire

```
1 $ mkdir unix # mkdir = make directory  
2 $ ls  
3 unix
```

Se déplacer dans un sous répertoire et en revenir

```
1 $ pwd # pwd = print working directory, affiche le repertoire courant  
2 /home/bnegreve/  
3 $ cd unix # cd = change directory, se déplacer dans un sous-repertoire  
4 $ pwd  
5 /home/bnegreve/unix  
6 $ cd .. # revenir dans le repertoire parent  
7 $ pwd  
8 /home/bnegreve/
```

Manipuler des fichiers

mkdir/rmdir/rm/mv/cp

```
1 $ mkdir dir # creer un repertoire dir
2 $ rmdir dir # supprimer le repertoire dir (vide)
3 $ rm file # supprimer le fichier file
4 $ rm -r dir # supprimer le repertoire dir et tout son contenu
5 $ mv file file2 # renommer file en file2
6 $ mv file dir/ # déplacer file dans dir
7 $ cp file file2 # copier le fichier file dans file2
```

Répertoire parent / repertoire courant

Le répertoire '.' fait toujours référence au répertoire courant

```
1 $ pwd
2 /home/bnegreve/unix
3 $ cd .
4 $ pwd
5 /home/bnegreve/unix
6 $ # on a pas bougé
```

Le répertoire '..' fait toujours référence au repertoire parent

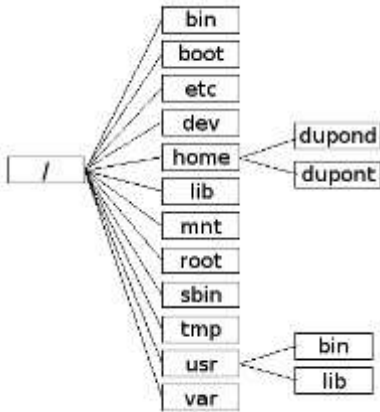
```
1 $ pwd
2 /home/bnegreve/unix
3 $ cd ..
4 $ pwd
5 /home/bnegreve
6 $ # on a est remontéd'un cran!
```

Racine

Racine

```
1 $ pwd
2 /home/bnegreve
3 $ cd ..
4 $ pwd
5 /home
6 $ cd ..
7 $ pwd
8 /
9 $ cd .. # ???
10 /
```

Hiérarchie classique UNIX



/bin: exécutables nécessaires

/lib: librairies nécessaires

/usr/bin: exécutables destinés aux utilisateurs

/etc: fichiers de configuration

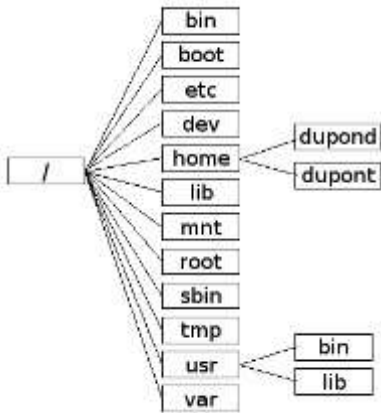
/tmp: repertoire pour les fichiers temporaires (supprimé au reboot)

/home: contient les repertoires des utilisateurs

/home/dupont: repertoire de l'utilisateur dupont.

image: coursunix.files.wordpress.com

Hiérarchie classique UNIX



/bin: exécutables nécessaires

/lib: librairies nécessaires

/usr/bin: exécutables destinés aux utilisateurs

/etc: fichiers de configuration

/tmp: repertoire pour les fichiers temporaires (supprimé au reboot)

/home: contient les repertoires des utilisateurs

/home/dupont: repertoire de l'utilisateur dupont.

image: coursunix.files.wordpress.com

- Les logiciels sont installés automatiquement via des paquets (e.g. apt)

Hiérarchie abstraite & supports physiques

Le montage de supports physiques permet d'associer un noeud de la hiérarchie avec un support physique.

```
1 $ df -Tvh
2 Filesystem      Type   Size  Used Avail Use% Mounted on
3 /dev/mapper/neb--vg-root ext4   28G   18G  8.5G  68% /
4 /dev/sda2       ext2   237M  162M   63M  73% /boot
5 /dev/mapper/neb--vg-home ext4  425G  219G  185G  55% /home
6 ...
```

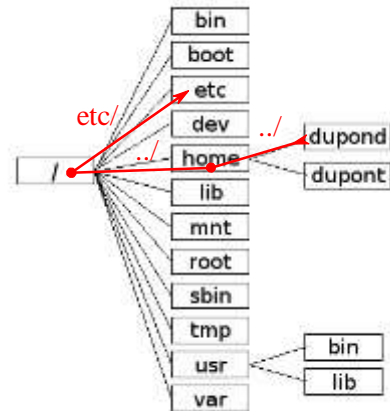
Hiérarchie abstraite & supports physiques

Le montage de supports physiques permet d'associer un noeud de la hiérarchie avec un support physique.

```
1 $ df -Tvh
2 Filesystem      Type  Size Used Avail Use% Mounted on
3 /dev/mapper/neb--vg-root ext4   28G  18G  8.5G  68% /
4 /dev/sda2       ext2   237M 162M   63M  73% /boot
5 /dev/mapper/neb--vg-home ext4  425G 219G 185G  55% /home
6 ...
```

- Le disque `/dev/mapper/neb--vg-root` est monté en tant que partition `root`
- Le disque `/dev/sda2` est monté en tant que partition `boot`
- Le disque `/dev/mapper/neb--vg-home` est monté en tant que partition `/home`

Chemin relatif, chemin absolu



'../..etc' : Chemin relatif

(au répertoire courant)

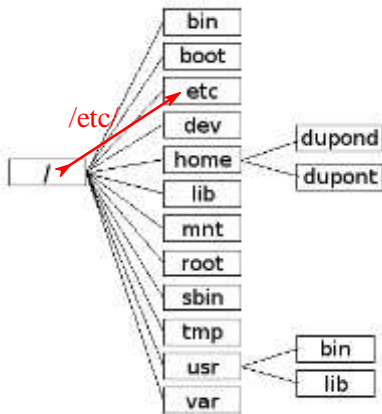
```
1 $ pwd
2 /home/dupond
3 $ cd ../../etc
4 $pwd
5 /etc
```

'/etc': Chemin absolu

(relatif à la racine)

```
1 $ pwd
2 /home/dupond
3 $ /etc
4 $ pwd
5 /etc
```

Chemin relatif, chemin absolu



'.././etc' : Chemin relatif

(au répertoire courant)

```
1 $ pwd
2 /home/dupond
3 $ cd .././etc
4 $ pwd
5 /etc
```

'/etc' : Chemin absolu

(relatif à la racine)

```
1 $ pwd
2 /home/dupond
3 $ /etc
4 $ pwd
5 /etc
```

Chemin relatif, chemin absolu (2)

Chemin relatif (par rapport au repertoire courant)

```
1 $ pwd
2 /home/bnegreve
3 $ ls test.c
4 $ ls ../bnegreve/test.c
5 $ ls ../stamby/unix/solution_projet.c
```

Chemin absolu (par rapport à la racine '/')

```
1 /home/bnegreve
2 $ ls /home/bnegreve/test.c
3 $ ls /home/stamby/unix/solution_projet.c
```

```
1 $ pwd
2 /home/bnegreve
3 cd ../../etc/../../home/bnegreve/cours_unix/tds/..
4 $ pwd ???
5 $ cd # ramène au repertoire home
6 $ pwd
7 /home/bnegreve
```

Fichiers

Un fichier

- existe dans le système de fichiers
- peut être ouvert, lu, écrit et refermé

Fichiers

Un fichier

- existe dans le système de fichiers
- peut être ouvert, lu, écrit et refermé

Types de fichiers

- Fichiers classiques (regular files).
 - Stockés sur le disque
 - Support d'information persistant (=RAM)
 - Destiné aux utilisateurs, ou aux applications
 - Exemple: fichier jpeg (image), fichier texte, fichier doc etc
 - Moyen d'échange entre les applications.

Fichiers

Un fichier

- existe dans le système de fichiers
- peut être ouvert, lu, écrit et refermé

Types de fichiers

- Fichiers classiques (regular files).
 - Stockés sur le disque
 - Support d'information persistant (=RAM)
 - Destiné aux utilisateurs, ou aux applications
 - Exemple: fichier jpeg (image), fichier texte, fichier doc etc
 - Moyen d'échange entre les applications.
- Répertoires
 - Fichier contenant une liste de fichiers
 - Tous les répertoires ont un repertoire parent, noté ..
 - L'ensemble des fichiers forme un arbre
 - Racine de l'arbre est notée /

Fichiers

Un fichier

- existe dans le système de fichiers
- peut être ouvert, lu, écrit et refermé

Types de fichiers

- Fichiers classiques (regular files).
 - Stockés sur le disque
 - Support d'information persistant (=RAM)
 - Destiné aux utilisateurs, ou aux applications
 - Exemple: fichier jpeg (image), fichier texte, fichier doc etc
 - Moyen d'échange entre les applications.
- Répertoires
 - Fichier contenant une liste de fichiers
 - Tous les répertoires ont un repertoire parent, noté ..
 - L'ensemble des fichiers forme un arbre
 - Racine de l'arbre est notée /
- Liens physiques/symboliques
 - Fichier qui référence un autre fichier

Fichiers

Un fichier

- existe dans le système de fichiers
- peut être ouvert, lu, écrit et refermé

Types de fichiers

- Fichiers classiques (regular files).
 - Stockés sur le disque
 - Support d'information persistant (=RAM)
 - Destiné aux utilisateurs, ou aux applications
 - Exemple: fichier jpeg (image), fichier texte, fichier doc etc
 - Moyen d'échange entre les applications.
- Répertoires
 - Fichier contenant une liste de fichiers
 - Tous les répertoires ont un repertoire parent, noté ..
 - L'ensemble des fichiers forme un arbre
 - Racine de l'arbre est notée /
- Liens physiques/symboliques
 - Fichier qui référence un autre fichier

Fichiers spéciaux

Les fichiers UNIX: interface généralisées pour faire des E/S.

- Fichiers périphériques `"/dev/"`
 - Fichier sans existence physiques sur le disque
 - Permet de communiquer avec un périphérique de manière unifiée (via un logiciel pilote)

```
1 cat /dev/input/mouse1 # coordonnées de la souris
2 echo "salut" > /dev/printer # imprime "salut" sur l'imprimante par
  default
```

- Fichiers virtuels `"/proc"` `"/sys"`
 - Fichier sans existence physiques sur le disque
 - Leur contenu reflète le contenu de la mémoire d'un composant du noyau
 - Permet de piloter ou d'obtenir des données depuis un composant du noyau

```
1 cat /proc/cpuinfo # récupère les infos sur les CPUs
2 cat /proc/mem # récupère les infos sur l'état de la RAM
```

Nom des fichiers

Règles strictes:

- taille entre 1 et 255 caractères

Recommandation

- Utilisez lettres, chiffres, -(tiret) ou _(souligné)
- Éviter les caractères spéciaux pour le shell:
<, >, |, \$, ?, &, [], *, !, ' , " , ' , (,) , @ , ~ , <espace>
- En général, le . (point) est utilisé pour séparer le nom de l'extension

```
1 fichier.c # source en c
2 fichier.jpeg # image
3 fichier.sh # script shell
4 fichier.c.old # ..
5 fichier # souvent un fichier texte
```

- le . en début de ligne rend le fichier caché

```
1 $ echo "salut" > test
2 $ ls
3 test
4 echo salut > .test
5 $ ls
6 $ ls -a
7 .test
```

Caractères spéciaux

Pour désigner plusieurs fichiers à la fois, on peut utiliser des caractères spéciaux

Exemples:

- le caractère '*' désigne n'importe quel nom de fichier
- *.pdf permet de désigner n'importe quel fichier qui termine par '.pdf'

```
1 $ ls # tout
2 a.jpeg b.pdf c.jpeg d.jpeg
3 $ ls * # pareil
4 a.jpeg b.pdf c.jpeg d.jpeg
5 $ ls *.pdf # que les noms de fichiers qui terminent par .pdf b.pdf
6 $ ls *.jpeg # que les noms de fichiers qui terminent par .jpeg
7 a.jpeg c.jpeg d.jpeg
8 $ ls *d* # que les noms de fichiers qui contiennent 'd'
9 b.pdf d.jpeg
10 $ ls b* # que les noms de fichiers qui commencent par 'b'
11 b.pdf
12 $ ls *a*j* # que les noms de fichiers qui contiennent un 'a' puis un 'j'
13 a.jpeg
14
```

Autres caractères spéciaux

- `'?'` désigne un caractère quelconque

```
1 $ ls
2 file.c file.h file.jpeg
3 $ ls file.? # tous les fichiers avec un seul caractère après le . file.c
4 file.h
```

- `'[...]'` n'importe quel caractères parmi l'ensemble entre les crochets

```
1 $ ls file.[ch]
2 file.c file.h
3 $ ls *.*[ch] # tous les fichiers .c ou .h
4 ...
5 $ ls rapport_201[0-3].pdf # tous les rapport de 2010 a 2013 rapport_2010.pdf
6 rapport_2011.pdf rapport_2012.pdf rapport_2013.pdf
```

Attributs des fichiers

Les fichiers sont associés a des persmissions

```
1 $ ls -l test.sh
2 -rwxr-xr-x 1 bnegrevergne lamsade 31 Mar 22 2019 test.sh
```

- -rwxr-xr-x droits d'accès au fichiers
- 1 nombres de liens pointant vers ce fichier
- bnegrevergne propriétaire du fichier (owner)
- lamsade groupe propriétaire du fichier (group)
- 31 Mar 22 2019 date de création
- test.sh nom du fichier

Permissions

Droits possibles:

- Read (r): permission de lire le fichier
- Write (w): permission d'écrire ou modifier le fichier
- Execute (x): permission d'exécuter le fichier

Les droits sont définis pour 3 catégories d'utilisateurs

- Le propriétaire du fichier
- Les membres du groupes propriétaire du fichier
- Les autres

-rwxr-xr-- droits d'accès au fichiers à séparer en trois parties: rwx, r-x et r--

- rwx: droits du propriétaire: lecture, écriture, exécution
- r-x: droits du groupe: lecture, exécution
- r--: droits des autres: lecture seulement

Manipuler le mode d'un fichier (chmod)

$r(\text{read}) = 4, w(\text{write}) = 2, x(\text{execute}) = 1$

- Mode = 7: Read + Write + Execute
- Mode = 5: Read + Execute
- Mode = 4: Read
- Mode = 1: Execute

```
1 bnegreve@neb:/tmp$ chmod 777
2 file_bnegreve@neb:/tmp$ ls -la file
3 -rwxrwxrwx 1 bnegreve bnegreve 0 Feb 10 14:19 file
4 bnegreve@neb:/tmp$ chmod 722 file
5 bnegreve@neb:/tmp$ ls -la file
6 -rwx--w--w- 1 bnegreve bnegreve 0 Feb 10 14:19 file
7 bnegreve@neb:/tmp$ chmod 744 file
8 bnegreve@neb:/tmp$ ls -la file
9 -rwxr--r-- 1 bnegreve bnegreve 0 Feb 10 14:19 file
10 bnegreve@neb:/tmp$ chmod 755 file
11 bnegreve@neb:/tmp$ ls -la file
12 -rwxr-xr-x 1 bnegreve bnegreve 0 Feb 10 14:19 file
```

Manipuler le mode d'un fichier (2)

```
1 $ chmod g+x file # ajoute la permission x aux utilisateur du groupe
2 $ chmod ug+x file # ajoute la permission x au propriétaire du fichier, et aux
  membres du groupe propriétaire .
3 $ chmod o-rwx # supprime toutes les permission pour les utilisateurs qui n'
  appartiennent pas au groupe propriétaire .
4 $ chmod o-rwx # supprime toutes les permission pour les utilisateurs qui n'
  appartiennent pas au groupe propriétaire .
5 $ chmod a-x # supprime la permission 'x' à tout le monde.
6 $ chmod og-x # supprime la permission 'x' à tout le monde, sauf au propriétaire du
  fichier
```


Changer le propriétaire d'un fichier

Changer le propriétaire d'un fichier

```
1 chown stamby test.sh
```

Changer le groupe propriétaire d'un fichier

```
1 chgrp ceremade test.sh
```

find/locate

- **locate** : trouve un fichier à partir d'un pattern grâce à un index

```
1 $ locate "*.jpeg" # tous les fichiers .jpeg
2 ...
```

- ▶ rapide
- ▶ updatedb pour mettre l'index à jour

- **find** : cherche un fichier à partir d'un pattern sans index
 - ▶ plus lent, mais toujours à jour

```
1 $ find . -name "*.jpeg" # tous les fichiers .jpeg dans le répertoire courant
2 ...
```

grep

Print lines that match patterns

man grep

```
1 $ grep 1862 file # toutes les lignes qui contiennent le 'mot' 1862
2 $ ls | grep .jpeg # tous les fichier dont le nom se termine par jpeg
3 $ grep poulet lesmiserables.txt # toutes les lignes qui contiennent le mot poulet
4 $ grep -o '18[0-9][0-9]' lesmiserables.txt | sort | head -n 1 # ??
```

Utile pour

- Extraire toutes les dates d'un fichier
- Extraire toutes les adresses emails
- Extraire les mots qui commencent par a et qui terminent par z
- ...

Expression rationnelles

Langage des motifs (patterns)

- `ab` (opérateur implicite): concaténation de `a` et de `b`
- `[ab]`: `a` ou `b`
- `a*`: 0 ou plus répétitions de `a`
- `^a`: motif qui commence par `a`
- `a$`: motif qui termine par `a`

```
1 $ grep ab # tous les mots qui contiennent un a directement suivi d'un b
2 $ grep a*b # tous les mots qui contiennent zero, un ou plusieurs a suivi d'un b
3 $ grep a[a-z]*z # tous les mots qui commencent par a et qui finissent par z
4 $ grep a[a-z]*k[a-z]*z # tous les mots qui commencent par a et qui finissent par z et qui
   contiennent un k
5 $ grep [12][1-9][1-9][1-9] # toutes les dates entre 1000 et 2999
```

sed

sed - stream editor for filtering and transforming text

man sed

18xx → 19xx

```
1 $ cat lesmiserables.txt | sed 's/18\[0-9][0-9]\1/19\1/g' > lesmiserables2000.txt
2 $ cat lesmiserables2000.txt | grep '19[0-9][0-9]' # enjoy!
```