Haute École de Bruxelles-Brabant
École Supérieure d'Informatique

2017 – 2018

Laboratoire de Java

Bachelor en Informatique – 1ère année

Un guide visuel de Linux

*Ce document a pour objectif de présenter de façon très visuelle les bases de l'utilisation d'un système Linux en **mode console**.*

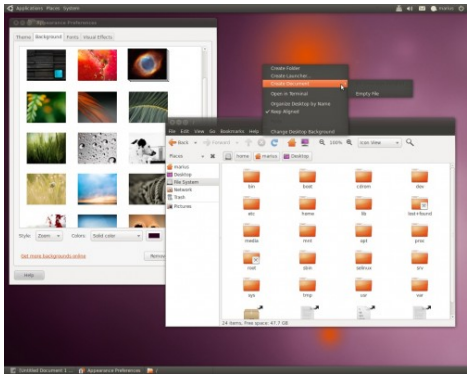
L'équipe Java

***Convention d'écriture** : Lorsque des informations sont spécifiques à l'environnement utilisé à l'école, nous le mettrons en évidence dans un encadré comme celui-ci.*

version du 16 août 2017

1 Linux

Linux est un système d'exploitation comme le sont également *Windows* ou *MacOS*. Il permet d'utiliser l'ordinateur et ses périphériques. Nous vous renvoyons à votre cours de *Système* pour les détails.



Screenshot de Ubuntu 10.10

La plupart des systèmes d'exploitation actuels proposent une interface graphique à l'utilisateur. Ainsi, les concepts de fenêtre, de menus, de souris, ... vous sont familiers. *Linux* n'échappe pas à la règle et propose un environnement proche de ce qu'on peut trouver sur d'autres systèmes.

*Vous aurez remarqué que, sur les machines des laboratoires, c'est Windows qui est installé. Linux est installé sur un serveur de l'école appelé **linux1**. Vous y accéderez à distance via un logiciel sur Windows (**putty**). Ça veut dire que vous allez la partager (travailler ensemble dessus). Ce sera l'occasion pour vous de découvrir les problèmes de permissions et de partage des ressources que ça engendre.*

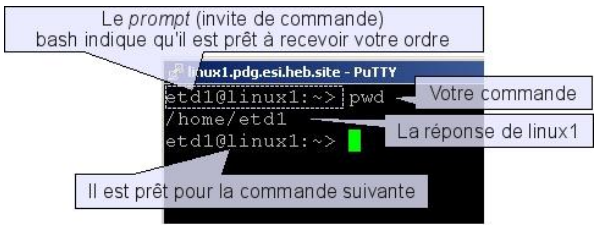
Vous sur windows

linux1

Bien que nous pourrions utiliser le mode graphique, nous allons, pour des raisons pédagogiques, vous apprendre à utiliser Linux en **mode *console***.

2 Le mode console

En mode *console*[1], nous n'allons pas utiliser des fenêtres et une souris mais nous allons donner (écrire) des commandes au système.



Shell : un programme dont le but est d'accepter des *commandes* de la part de l'utilisateur et de les exécuter. C'est ce *shell* qui *tourne* dans la console.

Bash : Plusieurs *shells* existent en *Linux*. *Bash* est l'un d'entre eux.

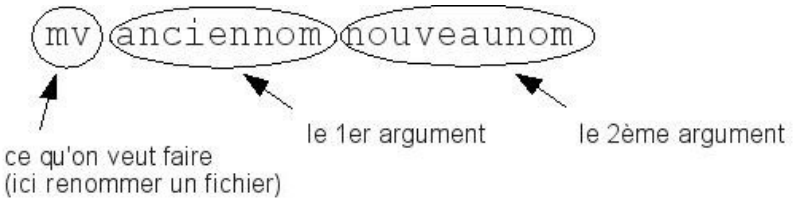
Dans l'exemple ci-dessus, vous pouvez constater que l'utilisateur entre la commande `pwd` qui a pour effet d 'afficher une réponse[2]. Ensuite, *bash* attend la commande suivante.

Pour travailler en mode console, il faut savoir dialoguer avec ce *shell*. Comment lui indiquer par exemple qu'on veut copier un fichier, exécuter un programme Java, ... ? Mais pour ça, il faut d'abord comprendre quelques concepts.

[1] On parle aussi de mode « commande » ou encore « terminal »
[2] Vous comprendrez bientôt ce que signifie `pwd`.

3 Disséquons une commande

Une commande est composée de plusieurs parties **séparées par un ou plusieurs espaces**.



La 1ère partie indique ce qu'on veut faire. Ensuite on donne des arguments pour préciser ce qu'on veut faire.

Exemples :

- Si on veut détruire un fichier (commande `rm`), il faut donner un argument : le nom du fichier à détruire.
- Si on veut renommer un fichier (commande `mv`), il faut donner 2 arguments : le nom du fichier à renommer et son nouveau nom.

4 Fichiers et dossiers

Comme sur la plupart des systèmes d'exploitation[1] les données sur le disque dur sont organisées en fichiers et dossiers :

Un **fichier** contient de l'information : un programme Java, un exécutable, un texte, ... mais aussi des informations propres au système : données de configuration, ...

Un **dossier** contient des fichiers. Il sert à organiser/regrouper les nombreux fichiers du disque. On parle aussi de **répertoire**.

Chaque fichier / dossier porte un nom. Plusieurs fichiers / dossiers peuvent porter le même nom. Ainsi, il est probable que chaque étudiant crée un dossier « td1 ». Nous verrons comment les distinguer.

Un dossier contient des fichiers mais il peut aussi contenir des dossiers qui, à leur tour, peuvent contenir des fichiers et des dossiers qui...

On a ainsi une organisation *hiérarchique* des fichiers.

[1] Windows par exemple.

5 Le dossier personnel

Sur Linux, chaque utilisateur dispose d'un « *dossier personnel* » (on dit aussi son « *home* »[1]). C'est dans ce dossier qu'il peut créer des fichiers (et des dossiers). C'est dans votre « dossier personnel » que vous créerez vos programmes Java par exemple. Par convention, ce dossier porte le même nom que le *login* de l'utilisateur[2].

Visuellement, ajoutons-lui une  pour le reconnaitre.

6 Le dossier courant

Lorsque vous dialoguez en mode console avec Linux, vous êtes à tout moment dans un des dossiers du système. On l'appelle le **dossier courant**. Juste après la connexion, il s'agit de votre dossier personnel mais ça peut changer.

Visuellement, indiquons le dossier courant avec une  .

Quel est l'**intérêt** de ce concept ? Lorsqu'on donnera une commande au *shell*, on pourra se passer de donner certaines précisions; l'action se fera dans le dossier courant.

Exemples :

- La commande `mkdir monDossier` crée un dossier qui s'appelle monDossier.
  Où ça ? Dans le dossier courant.
- La commande `ls` sans autre précision affiche le contenu du... dossier courant.

7 Se déplacer dans les dossiers

Reprenons déjà ce que nous savons.

Vous venez de commencer et vous vous trouvez dans votre dossier personnel (votre « home »).

Vous demandez à voir le contenu de ce dossier via la commande `ls` .

Le shell exécute cette commande et affiche quelques informations. En effet, quelques fichiers et dossiers de base ont été créés lors de la création de votre compte. N'y prêtons pas attention pour l'instant. Remarquez seulement que les dossiers sont affichés en bleu pour mieux les distinguer des fichiers.

Créons dans notre *home* le dossier tds via la commande `mkdir tds` . On obtient la situation ci-contre; ce qu'on peut vérifier aisément avec la commande `ls` .

Imaginons à présent qu'on veuille créer un dossier td1 dans le dossier tds. On ne peut pas écrire `mkdir td1` qui créerait un dossier td1 dans le dossier courant et, donc, à côté du dossier tds.

Une solution est de d'abord se déplacer / changer de dossier

[1] Comme avec les autres OS, on doit s'identifier avant d'utiliser Linux en donnant son *login* et son *mot de passe*. Le système sait ainsi qui on est.

[2] Dans tous nos exemples, nous prendrons le cas de l'utilisateur g12345.

courant via la commande cd.

Après la commande `cd tds` on est dans la situation ci-contre; le dossier courant a changé. Retenons : la commande cd (*change directory* en anglais, *change dossier* en français) permet de changer de dossier courant.

La situation a maintenant évolué. Si on tape la commande `mkdir td1` le dossier est créé dans le dossier courant qui est bien tds.

Et si on veut revenir en arrière ? La commande `cd` sans autre paramètre indique qu'on veut revenir au dossier personnel (votre *home*).

8 Éditer un fichier

Un **éditeur de texte** (ou, plus court, un **éditeur**) est un programme qui vous permet d'entrer, modifier, manipuler, ... le contenu d'un fichier texte. Vous allez l'utiliser abondamment pour vos programmes Java.

Vous connaissez probablement *Notepad* sous Windows. Sur Linux, il en existe beaucoup. Celui que nous vous proposons dans un premier temps au laboratoire s'appelle **nano**. Il est assez facile pour débuter et faire des choses simples. Par la suite, vous pourrez choisir de rester sous **nano** ou d'utiliser **vim**.

Ne refaisons pas ce qui est déjà bien fait ailleurs, vous pouvez consulter ce document pour en savoir plus : http://fr.openclassrooms.com/informatique/cours/reprenez-le-controle-a-l-aide-de-linux/nano-l-editeur-de-texte-du-debutant .

9 Visualiser un fichier

La commande que nous avons vue précédemment permet de modifier un fichier. Et si on veut juste regarder son contenu sans le modifier ? Il y a (au moins) 3 commandes possibles

cat	affiche tout le contenu du fichier à l'écran. Pratique s'il est petit mais intenable s'il est grand.
more	affiche le contenu du fichier « page par page ». La barre d'espace permet de passer à la page suivante. Idéal pour un long fichier à lire dans l'ordre.

10 Le système de fichiers

On a vu que chaque utilisateur dispose d'un dossier personnel. Où se trouvent ces dossiers ? Dans un dossier appelé home. Et où se trouve ce dossier home ? Dans un dossier appelé / (la barre oblique). Et où se trouve ce dossier / ? Nulle part ;) C'est le dossier principal (on dit aussi la racine) du système de fichiers.

Comparons avec le système Windows. En Windows, chaque périphérique de stockage (partition d'un disque, cd-rom, ...) possède son propre système de fichiers, désigné par une

lettre (C: par exemple pour le système de fichiers qui contient l'OS).

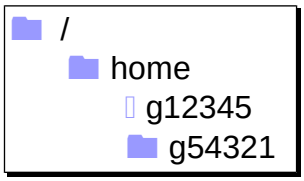
Avec Linux, tous les systèmes de fichiers sont assemblés en un seul système de fichiers dont le dossier principal est désigné par la barre oblique (/).

Ci-contre on peut voir que le dossier principal (/) contient le dossier home qui contient le dossier g12345, la home de l'utilisateur g12345.

Attention ! Le dossier appelé home ne contient pas que la home de l'utilisateur mais toutes les homes.

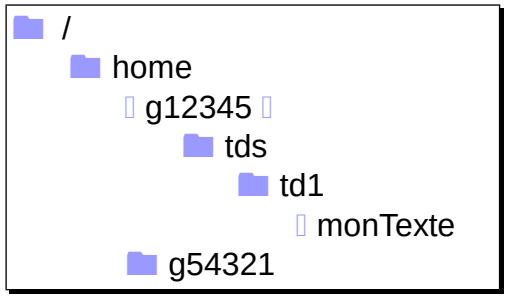
Ici, il contient aussi le dossier g54321 qui est la home de l'utilisateur g54321.

On remarque avec le schéma ci-dessus que notre façon de représenter les dossiers n'est pas pratique pour une grande hiérarchie de dossiers. Utilisons plutôt la notation ci-contre, plus compacte.



11 Chemin relatif

Partons de la situation ci-contre. La *home* de g12345 contient un dossier tds qui contient lui-même un dossier td1 qui contient le fichier monTexte. L'étoile indique quel est le dossier courant.



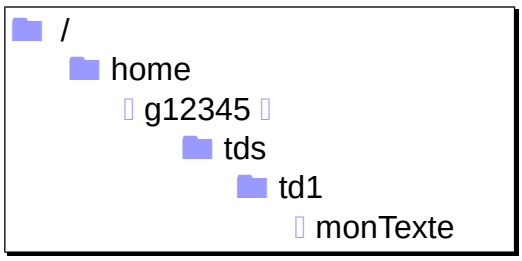
Comment indiquer qu'on veut afficher le contenu du fichier monTexte ? Une première option est de d'abord se déplacer dans td1 (changer de dossier courant). Mais il est également possible de le faire sans changer de dossier courant, ce qui se révélera très pratique par la suite.

Comment ? Pas avec la commande ~~cat monTexte~~. Pourquoi ? Le *bash* comprend cette commande comme « affiche le contenu du fichier monTexte qui se trouve **directement dans** le dossier courant); il ne prend pas l'initiative de regarder dans les sous-dossiers. Ici, il ne verrait que le dossier tds et aucun fichier monTexte.

Retenons, l**e bash ne cherche jamais** un fichier. Pourquoi ? D'une part parce que ce serait un gros travail pour lui (il y a des dizaines de milliers de fichiers dans un système de base). D'autre part, parce qu'il faudrait gérer le cas où il existe **plusieurs fichiers qui portent le même nom**; lequel choisirait-il ?

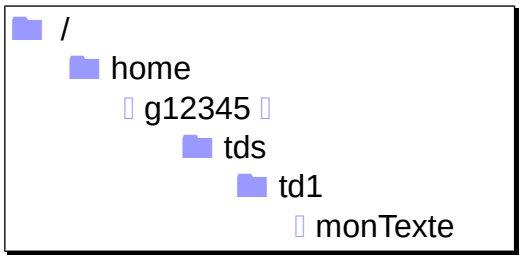
Il n'y a pas moyen alors ? Si ! Mais il faut lui indiquer le **chemin** pour arriver au fichier. Par exemple, on pourrait lui dire quelque chose comme : « d*ans le dossier courant, tu trouveras un dossier tds qui contient un dossier td1 qui contient le fichier monTexte que je veux voir* ».

On parle de **chemin relatif** car on va indiquer le chemin à suivre **à partir du dossier courant**.



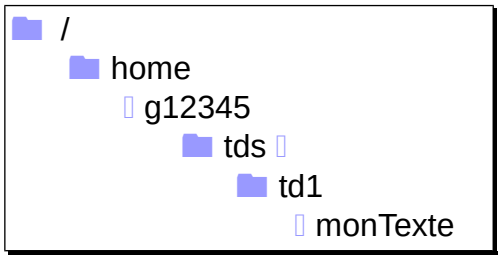
`cat tds/td1/monTexte`

Affiche le contenu du fichier monTexte.
Les étapes du chemin sont séparées par un /.



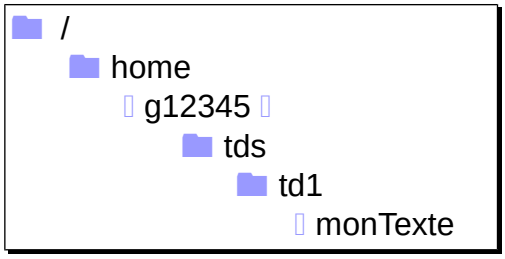
`ls tds/td1`

Liste le contenu du dossier td1 (qui se trouve dans le dossier tds).
Un chemin peut aussi être utilisé pour désigner un dossier.



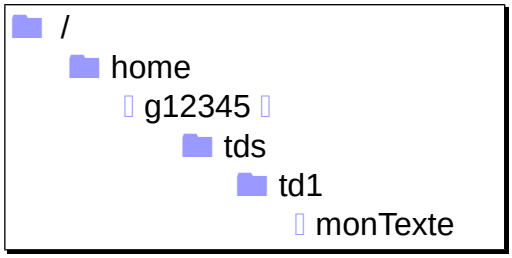
`cat td1/monTexte`

Affiche le contenu du fichier monTexte.
Le chemin commence à partir du dossier courant.



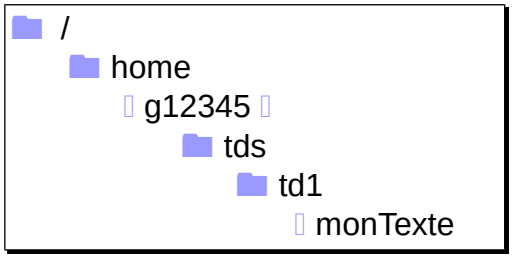
`ls ..`

Liste le contenu du dossier home.
« .. » dans un chemin indique qu'il faut *remonter*.



`ls ../g12345/tds/td1/..`

Liste le contenu du dossier tds.
Pourquoi faire simple quand on peut faire compliqué ;)



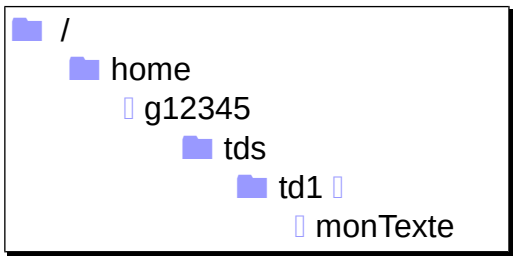
`ls .`

Liste le contenu du dossier g12345.
« . » dans un chemin indique qu'on ne bouge pas.
S'il est tout seul ça veut donc dire : le dossier courant (≡ ls)

12 Chemin absolu

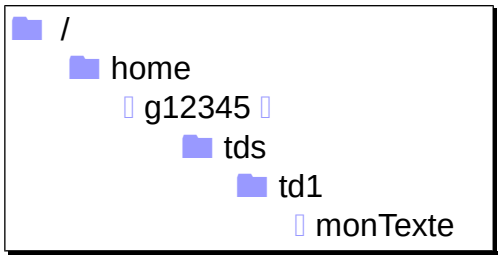
Partons de la situation ci-dessous. Si on veut lister le contenu du dossier home, on peut indiquer : `cat ../../../`

Mais il y a plus simple : indiquer le chemin non pas à partir du dossier courant, mais à partir du sommet (la racine). On parle alors de **chemin absolu**.



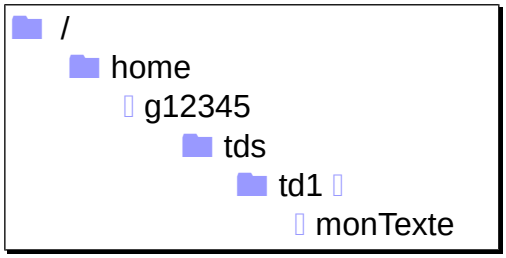
`ls /home`

Liste le contenu du dossier home qui se trouve à la racine.
Un chemin qui commence par / est un chemin absolu.



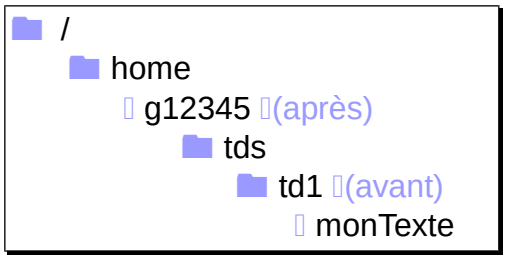
`ls /home`

Liste le contenu du dossier home qui se trouve à la racine.
Avec un chemin absolu, on désigne toujours le même endroit quel que soit le dossier courant.



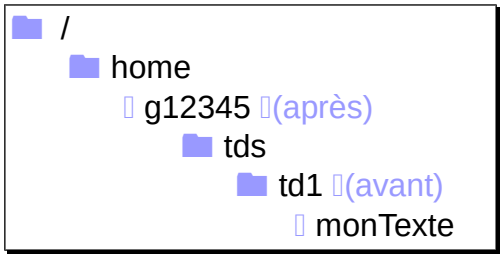
`ls /home/g12345/tds/td1`

Liste le contenu du dossier td1.



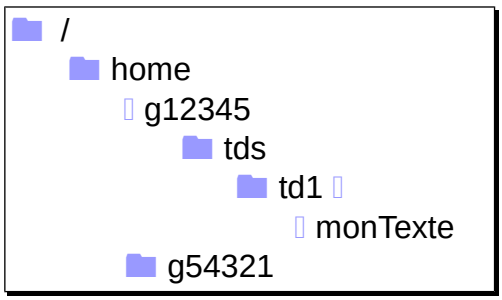
`cd /home/g12345`

Change de dossier courant.



`cd ~`

Le dossier courant est maintenant la *home*.
Le symbole ~ désigne la home de l'utilisateur. (raccourci pour /home/g12345)
Un chemin qui commence par ~ est aussi un chemin absolu.



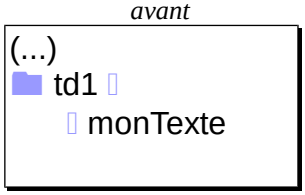
`ls ~g54321`

Liste le contenu de la *home* de l'utilisateur *g54321*.
~g54321 est un raccourci pour /home/g54321.
C'est donc aussi un chemin absolu.

13 Copier un fichier

La commande pour copier un fichier est cp (*copy* en anglais). Partons d'un exemple simple où on veut copier un fichier qui se trouve dans le dossier courant (avec la copie dans le dossier courant également)

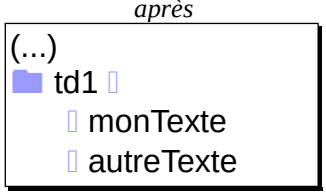
avant



`cp monTexte autreTexte`

Prend une copie du fichier monTexte et appelle la copie autreTexte.

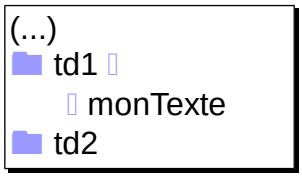
après



(...) Pour simplifier, on n'indique pas les dossiers au-dessus.

Imaginons à présent une situation plus complexe comme indiqué ci-dessous (avant modification). On voudrait que la copie ne se trouve pas dans le dossier courant mais ailleurs. Pas de problème, il suffit d'indiquer le chemin où placer la copie.

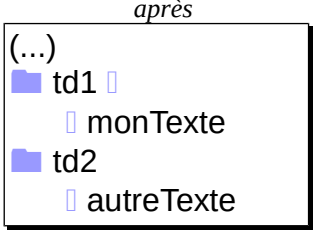
avant



`cp monTexte ../td2/autreTexte`

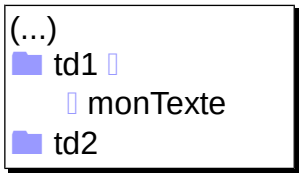
Prend une copie du fichier monTexte et la met dans le dossier td2 sous le nom : autreTexte.

après



Continuons notre exploration de la commande cp et imaginons que la copie doit porter le même nom que l'original (c'est possible si la copie est dans un autre dossier). Il suffit d'indiquer un nom de dossier comme destination et pas un nom de fichier.

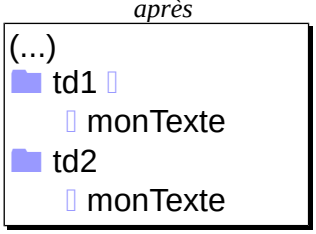
avant



`cp monTexte ../td2`

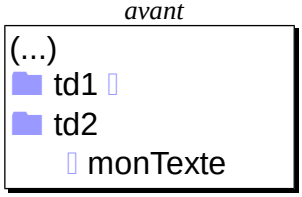
Prend une copie du fichier monTexte et la met dans le dossier td2 (sous le même nom).

après



Étape suivante. On considère cette fois que le fichier à copier n'est pas dans le dossier courant et qu'il faut placer la copie dans le dossier courant. Le mauvais réflexe à bannir (et qui vient de notre habitude des interfaces graphiques) est de d'abord se déplacer dans le fichier qui contient le fichier à copier. Il est plus facile d'indiquer le chemin où trouver le fichier à copier.

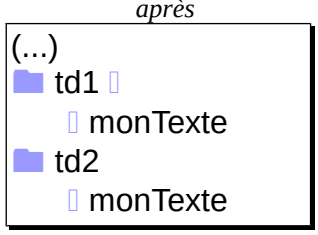
avant



`cp ../td2/monTexte .`

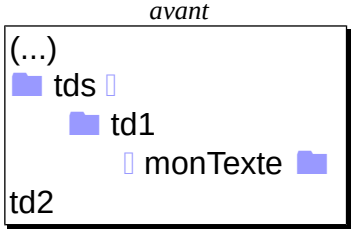
Prend une copie du fichier monTexte qui se trouve dans le dossier td2. La copie est à placer dans le dossier courant (.).

après



Enfin, on peut imaginer le cas général où, ni l'original, ni la copie ne sont dans le dossier courant.

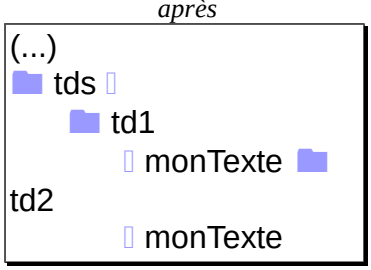
avant



`cp td1/monTexte td2`

Copie le fichier monTexte qui se trouve dans le dossier td1. La copie doit être placée dans le dossier td2.

après



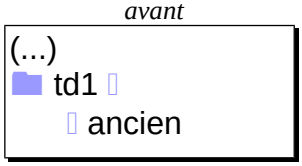
14 Déplacer / renommer un fichier

En Linux, la même commande est utilisée pour déplacer ou renommer un fichier. Il s'agit de mv (pour *move* en anglais). Elle accepte 2 paramètres :

- Le 1er indique le fichier qui faut déplacer / renommer
- Pour le 2e
  - Si c'est un dossier, il indique où déplacer le fichier (nom inchangé)
  - Si c'est un fichier, il indique également le nouveau nom du fichier

Ce qui a été vu pour la copie peut s'appliquer ici. En particulier, le fait que les 2 paramètres peuvent être des chemins absolus ou relatifs. Voyons quelques cas :

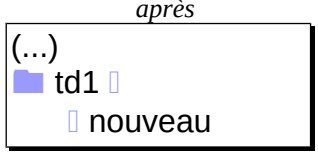
avant



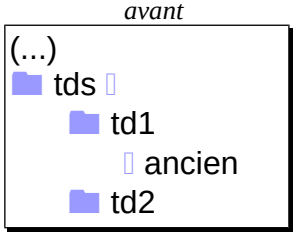
`mv ancien nouveau`

Renomme le fichier indiqué.

après



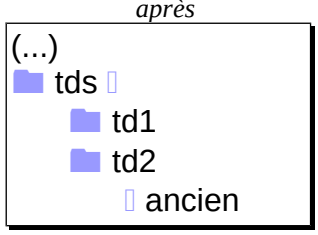
avant



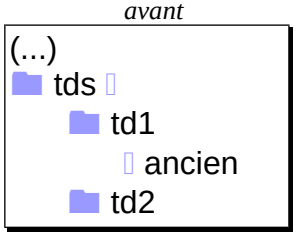
`mv td1/ancien td2`

Déplace le fichier indiqué du td1 vers le td2. Le fichier ne change pas de nom.

après



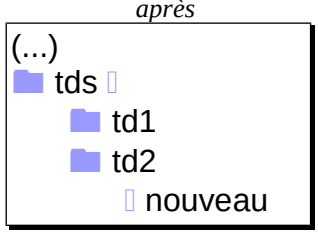
avant



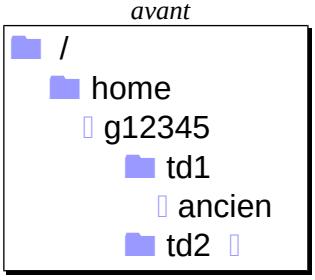
`mv td1/ancien td2/nouveau`

Déplace le fichier indiqué du td1 vers le td2. Le fichier change aussi de nom.

après



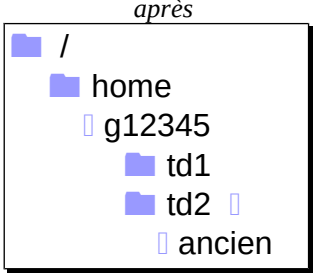
avant



`mv ~/td1/ancien .`

Déplace le fichier indiqué par un chemin absolu dans le dossier courant.

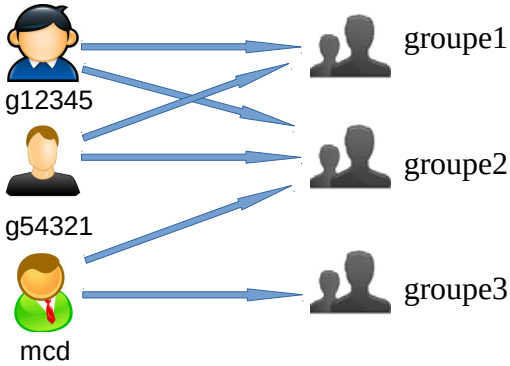
après



(...) Pour simplifier, on n'indique pas les dossiers au-dessus.

15 Les groupes d'un utilisateur

Les utilisateurs d'un système Linux sont groupés. Un groupe contient un ou plusieurs utilisateur(s) et un utilisateur appartient à un ou plusieurs groupe(s).



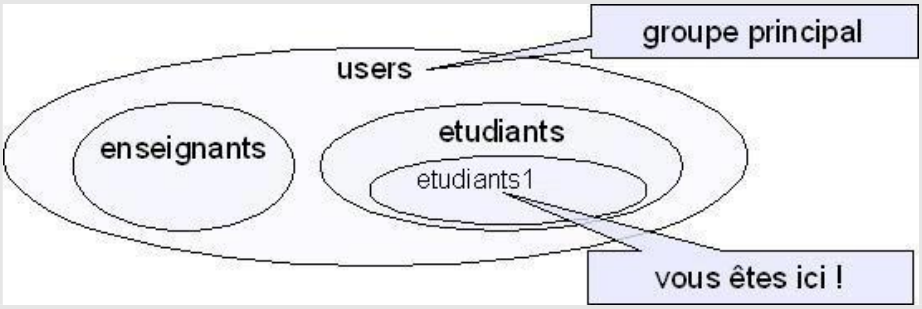
Dans l'exemple ci-contre on peut constater que :

- l'utilisateur g12345 appartient au groupe1 et au groupe2;
- g54321 appartient aux mêmes groupes (groupe1 et groupe2);
- mcd appartient au groupe2 et au groupe3.

À quoi servent les groupes ? À gérer les permissions. On va pouvoir dire un truc du genre : « Seuls les professeurs peuvent voir le contenu de ce fichier ».

Les groupes qui existent concrètement sur une machine sont définis par l'administrateur[1] de la machine.

Sur linux1, on a fait le choix d'une organisation hiérarchique des groupes.



- *users : tous les utilisateurs sont dans ce groupe*
- *enseignants : tous les professeurs sont dans ce groupe*
- *etudiants : tous les étudiants sont dans ce groupe*
- *etudiants1 : tous les étudiants de 1re année sont dans ce groupe*

Ainsi, g12345 appartient aux groupes etudiant1, etudiants et users. Attention ! Cette notion de groupe n'a rien à voir avec les groupes définis à l'école.

Comment mettre en évidence les groupes d'un utilisateur ?

[1] L'administrateur est la personne qui installe un système d'exploitation et le gère au quotidien : installation de logiciels, gestion des comptes utilisateurs et des groupes, sauvegardes, gestion des pannes, ...
Sur Linux, on dit aussi le « super user », le « compte root » ou le « root ».

Via la commande groups qui donne tous les groupes de l'utilisateur.

- groups : exécuté par l'utilisateur g12345 affichera users etudiants etudiants1
- groups mcd : affichera les groupes de mcd, à savoir users enseignants

Une dernière notion. Parmi tous les groupes d'un utilisateur, il y en a un qui est le g**roupe principal** (c'est le 1er donné par la commande groups). C'est celui qui sera utilisé par défaut lors de certaines opérations (par exemple lors de la création d'un fichier; cf plus bas)

16 Propriétaire et groupe d'un fichier

Tous les fichiers ont un propriétaire et appartiennent à un groupe. C'est la base pour définir les permissions sur ce fichier. On pourra ainsi donner des permissions différentes : au propriétaire du fichier, aux utilisateurs qui appartiennent au même groupe que le fichier, à tous les autres.

Comment le mettre en évidence ? La commande ls possède une option demandant d'afficher plus de détails.

Sur le schéma ci-dessus, on peut lire, par exemple, que le fichier Max.java appartient à g32671 et a été placé dans le groupe users.

Lorsqu'un fichier est créé, il appartient à celui qui l'a créé et est placé dans le groupe **principal** de son créateur. On peut le changer par la suite via des commandes

- chown mcd monFichier — change le propriétaire du fichier monFichier qui devient mcd (seul l'administrateur peut exécuter cette commande)
- chgrp etudiants1 monFichier — indique que le fichier monFichier doit être placé dans le groupe etudiants1 (le propriétaire du fichier peut exécuter cette commande mais il doit indiquer un groupe auquel il appartient)

17 Les permissions sur un fichier

Nous savons maintenant qu'un fichier appartient à un utilisateur et est placé dans un groupe (cf. point 16).

Il existe 3 types de permissions : en lecture, en écriture et en exécution.

- **Lecture** : Si une personne a le droit en lecture sur un fichier, il peut en **voir** le contenu. Par exemple, avec cat, more ou encore view.
- **Écriture** : Si une personne a le droit en écriture sur un fichier, il peut en **modifier** le contenu. Par exemple, avec nano.
- **Exécution** : Cette permission concerne les exécutables. Un exécutable est un fichier qui

contient un programme en langage machine directement exécutable par le processeur. Par exemple, il existe quelque part sur le disque un fichier nano qui contient le programme de la commande nano.

Ces 3 permissions peuvent évidemment être combinées (exemple : on donne l'accès en lecture et en écriture).

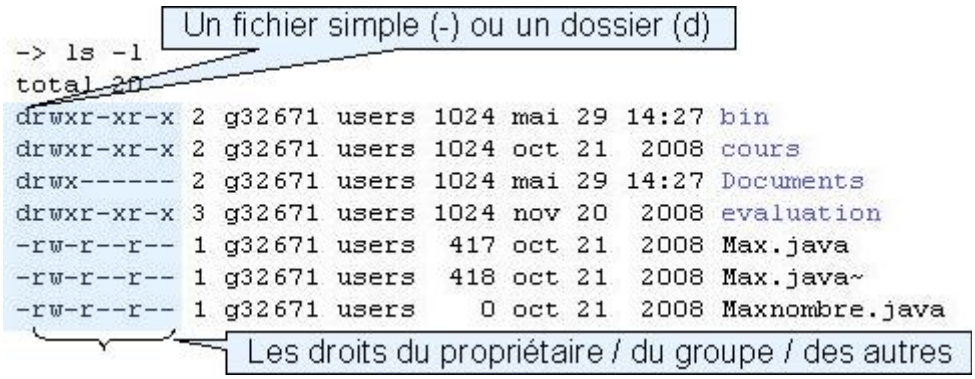
Cas pratiques :

- Un programme **Java** ne doit pas être exécutable puisqu'il n'est pas exécuté directement par la machine mais qu'il est d'abord compilé. La personne qui va le compiler doit juste avoir le droit en lecture.
- Un **bytecode** (le fichier **.class** produit par le compilateur Java) ne doit pas non plus être exécutable. L'utilisateur qui va l'exécuter, le fera au travers de la machine virtuelle; un droit en lecture est suffisant.

En pratique, on ne veut pas donner les mêmes droits à tout le monde. On pourra préciser :

- les droits du propriétaire d'un fichier;
- les droits des utilisateurs qui appartiennent au même groupe que le fichier;
- les droits pour toutes les personnes non reprises dans une catégorie ci-dessus (les *autres*).

Comment voir les permissions données à un fichier ? Toujours avec l'option « l » de ls.



Les permissions sont indiquées avec des lettres (r pour la lecture, w pour l'écriture et x pour l'exécution) dans cet ordre en mettant un tiret (-) si la permission n'est pas donnée. Il y a 3 blocs de 3 données pour les droits du propriétaire, du groupe et des autres (dans cet ordre)

Sur l'exemple ci-dessus, on peut voir que le fichier Max.java :

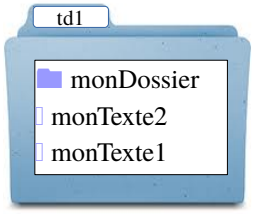
- est en lecture/écriture pour le propriétaire (g32671)
- est en lecture seule pour tous les utilisateurs du groupe users (en fait tout le monde sur linux1)
- est en lecture seule pour les autres (en fait personne sur linux1)

18 Les permissions sur un dossier

Tout comme pour les fichiers, on peut donner des permissions au dossier. La situation est sensiblement la même :

- Un dossier a un propriétaire et un groupe.
- On indique les mêmes permissions r, w et x
- On spécifie des permissions pour le propriétaire, le groupe et les autres

Ce qui change, c'est la signification des permissions. Que veut dire « exécuter » un dossier par exemple ? Pour bien comprendre, prenons une image.



Imaginons qu'un dossier possède sur sa « couverture » une liste de son contenu. Par exemple, on voit ci-contre que le dossier td1 contient un dossier et 2 fichiers.

Explicitons à présent le sens de chaque droit :

- **Lecture** : on a le droit de lire ce qui est écrit sur la « couverture » du dossier. On peut par exemple faire un ls du dossier.
- **Écriture** : on peut modifier la « couverture » du dossier. On peut ainsi effacer un fichier (via la commande rm). On remarque ainsi que pour effacer un fichier il ne faut pas de droit en écriture sur le fichier mais bien sur le dossier qui le contient.
- **Exécution** : on peut « ouvrir » le dossier / entrer dedans. On peut ainsi en faire son dossier courant (cd td1) ou le traverser dans un chemin (cat td1/monTexte1).

19 Modifier les permissions

Lorsqu'un fichier est créé, il l'est avec des permissions par défaut définies par l'administrateur du système. On peut évidemment les changer via la commande chmod.

Il existe 2 notations fort différentes pour indiquer les permissions : avec un nombre et avec des lettres.

Modifier les permissions avec un nombre.

Pour indiquer les permissions avec un nombre, on considère que r=4, w=2 et x=1. Et on additionne les valeurs ainsi obtenues.

Exemples :

lettres	nombre	signification
`r--`	4 (4+0+0)	droit en lecture uniquement
`rw-`	6 (4+2+0)	droit en lecture et en écriture
`--x`	1 (0+0+1)	droit d'exécution uniquement

On combine alors les nombres obtenus pour les 3 sortes d'utilisateurs.

Exemple : `chmod 640 monFichier.java` donne au propriétaire le droit en lecture/écriture, aux utilisateurs dans le même groupe que le fichier le droit en lecture uniquement et aucun droit aux autres.

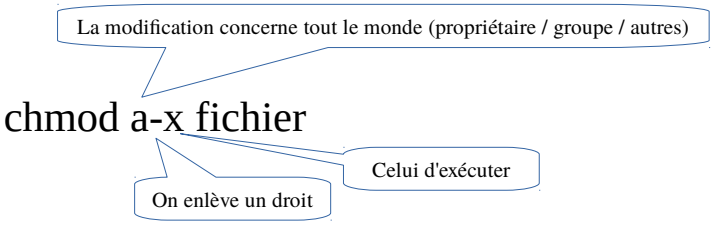
Modifier les permissions avec des lettres.

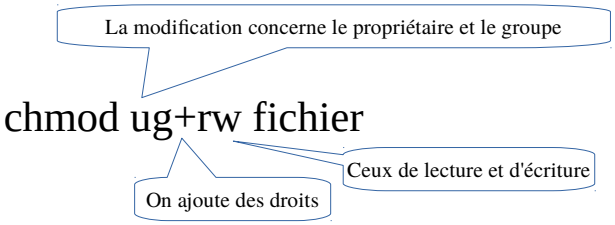
Avec la notation précédente, on spécifie **toutes** les permissions. Ici, on va aussi pouvoir effectuer des modifications : ajouter/enlever un droit sans toucher aux autres droits.

Le format de la commande est assez complexe.

- On indique d'abord les utilisateurs concernés par la modification des droits : « u » pour le propriétaire (*user*), « g » pour le groupe, « o » pour les autres (*other*) et « a » pour tout le monde (*all*)
- On indique ensuite si on veut ajouter (« + ») ou enlever (« - ») un droit
- On indique enfin quel(s) droit(s) on ajoute ou enlève (« r », « w » et/ou « x »)

Exemples :





Nous vous renvoyons à la page de manuel pour découvrir toutes les possibilités.

20 Les jokers

Imaginons qu'on ait besoin de supprimer beaucoup de fichiers. Comment faire sans que ça devienne un calvaire ?

La 1ère chose à savoir est que beaucoup de commandes linux qui traitent un fichier peuvent en traiter plusieurs à la fois; il suffit de les indiquer tous.

Exemple : `rm texte1 texte2 texte3 texte4` supprime les 4 fichiers indiqués.

Il y a progrès mais on voudrait aller plus loin et éviter de taper explicitement chaque nom. Par exemple, on pourrait avoir envie de dire :

- Supprime tous les fichiers qui commencent par « texte » ;
- Supprime tous les fichiers Java (qui se terminent par .java) ;
- ...

C'est possible grâce à la notion de « joker ». On ne va pas donner explicitement un nom de fichier mais un « motif », c'est-à-dire une *description* des noms de fichiers concernés. Il y a essentiellement deux jokers : '?' et '*'[1].

[1] L'aide-mémoire aborde également les crochets « [...] »

Le '?'

Commençons par celui-ci qui est le plus facile. Lorsque le bash rencontre un '?' dans un nom de fichier il sait qu'il peut le remplacer par n'importe quel caractère.

Exemple : rm texte? supprime tous les fichiers dont le nom commence par 'texte' suivi d'un caractère quelconque.

motif : t e x t e ?
fichier : t e x t e 2
OK

motif : t e x t e ?
fichier : t e x t e
NON

motif : t e x t e ?
fichier : t e x t e 2 1
NON

motif : t e x t e ?
fichier : t e x t e s
OK

motif : t e x t e ?
fichier : t e x t 2
NON

motif : t e x t e ?
fichier : e x t e 2
NON

Le '*'

Lorsque le bash rencontre un '*' dans un nom de fichier il sait qu'il peut le remplacer par n'importe quelle **suite** de caractères.

Exemple : rm td*.java supprime tous les fichiers java commençant par 'td'.

motif : t d * . j a v a
fichier : t d 1 0 . j a v a
OK

motif : t d * . j a v a
fichier : t d 3 . j a v a
* peut remplacer un seul caractère

motif : t d * . j a v a
fichier : t d . j a v a
* peut remplacer 0 caractère

motif : t d * . j a v a
fichier : e x 3 . j a v a
NON

motif : t d * . j a v a
fichier : t d 3
NON

motif : t d ? . *
fichier : t d 3 . c l a s s
On peut combiner * et ?