R1.04 - Systèmes - TP 1
Premiers pas

Avant propos

Ce 1er TP est très verbeux. Comme vous débutez, nous faisons en sorte de vous expliquer de façon détaillée le pourquoi et le comment. Au fil des prochains TP, les explications détaillées feront place à plus de découverte par l'expérimentation. Donc, ne soyez pas effrayés par la longueur de ce 1er TP. Restez concentré.

Prenez des notes (créez des fichiers Markdown, comme expliqué dans le *TP 0 - VSC*). Pour vous aider à prendre des notes, vous verrez ce pictogramme : ✪ quand une information mérite absolument d'être notée. Mais n'hésitez pas à prendre d'autres notes même si le pictogramme n'est pas présent !

Répondez aux questions par écrit pour que votre enseignant.e puisse rapidement savoir si vous avez compris ce que vous faites.

Lisez attentivement tout ce qui est écrit. N'allez pas plus vite que la musique !

Le rythme est d'environ 8 min / page, questions comprises mais il n'y a pas de questions sur toutes les pages.

Il vous faudra certainement revenir lire certains passages lors d'un travail personnel en dehors des séances, surtout au début. Ce 1er TP présente des **concepts essentiels** pour la compréhension du reste à venir.

Après l'introduction, vous allez commencer à manipuler. Ne paniquez pas si vous ne comprenez pas ce que vous faites ou ce qui s'affiche à l'écran. Dans ce cas, interrogez votre enseignant.e qui est là pour vous aider.

Il n'y a pas de question *bête*. La question bête est celle qui n'est pas posée.

C'est important de comprendre ce que vous faites. Les TP sont construits comme un Lego. Si une brique vous manque ou est mal placée, la construction sera bancale ou pire, impossible à poursuivre. Si une action affiche un message d'erreur, n'allez pas plus loin, le reste ne fonctionnera sans doute pas. Faites appel à l'enseignant.e avant tout.

Comme elles sont présentées d'une façon visible et particulière, vous allez vite repérer dans les sujets de TP les commandes à saisir. **Ne vous précipitez pas dans la saisie de ces commandes avant d'avoir lu le texte introductif ainsi que le texte qui suit immédiatement** car il y a parfois des choses importantes pour la compréhension de la commande !

Interface graphique

Vous êtes certainement habitués à interagir avec votre ordinateur par l'entremise de ce qu'on appelle une *interface graphique* : des fenêtres, des menus, une souris et évidemment un clavier.

Ces outils sont aujourd'hui classiques sur les ordinateurs et sont certainement bien pratiques pour les utilisateurs néophytes. Mais pour vous, informaticien.ne.s en devenir, ils ont un gros inconvénient : ils masquent beaucoup trop la mécanique interne du système d'exploitation et sont trop contraignants pour nos besoins (d'informaticien.ne.s).

Une interface (graphique) porte bien son nom, elle se place entre deux parties, l'utilisateur et le système. L'utilisateur donne des ordres au système par le biais d'un tableau de bord, cette bien nommée interface graphique. Tout ceci est séduisant sur le papier mais peut parfois (souvent ?) être frustrant si rien n'est prévu dans l'interface pour faire une action longue et fastidieuse.

Un exemple pour mieux comprendre

Prenons un exemple. Vous avez, sur votre clé USB, 150 photos d'une soirée un peu animée, dont vous êtes le seul à ne pas vous souvenir, et vous souhaitez supprimer les traces de ce moment d'égarement. Pas de problème, vous maîtrisez la souris et vous savez comment sélectionner ces 150 photos pour les mettre à la corbeille (sans oublier de la vider ensuite). 3 clics. Trop facile !

Prenons maintenant une autre situation, un peu plus *professionnelle* (c'est-à-dire : un cas concret qui vous arrivera certainement). Vous recevez 1.000 photos JPEG, un format d'image que vous devez connaître, c'est généralement le format des appareils photos numériques. Ces photos, qui doivent alimenter un site Web, sont celles de produits vendus par votre client et qu'il va falloir convertir en PNG, un autre format d'image, plutôt utilisé sur le Web (même si le JPEG l'est aussi, mais c'est histoire de pimenter notre scénario). Et, cerise sur le gâteau, comme votre client vous aime bien, il a mis ces images dans plein de sous-dossiers. Mais hélas ça ne vous arrange pas du tout et vous voudriez les grouper (donc les déplacer) dans un unique dossier. Allez, soyons fous, on aimerait aussi que les noms des images soient uniquement en majuscules (donc il faut convertir les noms actuellement en minuscules si besoin) et s'il y a des noms avec des espaces, on souhaite remplacer ces espaces par des "**_**" (des *tirets bas*[1]). Ouf, ça en fait du boulot !

[1] Le "**_**", se dit indifféremment *tiret bas*, *souligné* ou encore *underscore*. Mais arrêtez de l'appeler *tiret du 8*, ça ne veut rien dire ! Il y a des claviers (sur Mac ou en QWERTY par exemple) qui n'ont pas de "**_**" sur la touche *8*. Regardez aussi cette touche *8* sur un clavier de mobile, vous y voyez un *tiret bas* ? Idem pour "**-**" qui est simplement un *tiret* et non pas le *tiret du 6* 😀

Pensez-vous qu'il existe dans l'interface graphique une action bien cachée au fond d'un menu pour effectuer en une seule fois cette série d'actions sur vos 1.000 photos ? Sans doute pas…

Et là, le coup de grâce, le chef de projet débarque dans le bureau et vous donne 15 min pour préparer tout ça ! Quoi ? 15 min ? Il veut ma peau ? Il y en a pour la journée !

Cet exemple peut vous sembler irréaliste, mais c'est pourtant une situation très commune pour un informaticien. Et cet exemple démontre les limites des interfaces graphiques : elles sont très pratiques pour faire des actions clairement définies mais sont sans intérêt (ou presque) quand on sort du cadre prévu. Et elles ne peuvent pas prévoir toutes les situations possibles, elles couvrent les situations courantes et simples, uniquement.

À l'aide de la seule interface graphique, il va vous falloir faire des milliers d'actions à la souris et au clavier pour y arriver.

Pourtant, vous serez peut-être étonné d'apprendre que cette tâche ne prend pas plus d'une minute à être réalisée… avec vos dix doigts !

Et, en principe, vous serez capables de réaliser cette *magie* d'ici quelques semaines, mais, évidemment, pas avec une interface graphique.

PS : 1.000 photos ça peut sembler beaucoup. Il n'est pas rare de devoir traiter des dizaines de milliers de photos dans une base de photos d'entreprise !

Que la Force soit avec vous !

Vous l'aurez deviné, nous allons donc laisser les interfaces graphiques aux *utilisateurs lambda*.

Nos outils favoris pour cette ressource R1.04 s'appellent **Terminal** et **Commandes**.

Terminal

Le **Terminal** permet de donner des ordres à l'ordinateur ✪, ou plus précisément au Système d'Exploitation de l'ordinateur car c'est plutôt le Système qui contrôle l'ordinateur (pour vous). Un peu comme quand on appuie sur la pédale d'accélérateur de la voiture, on donne l'ordre à la voiture d'aller plus vite, et plus précisément, on donne cet ordre au moteur car c'est lui qui contrôle la voiture.

Finalement, le Terminal est donc aussi une **interface**. Mais à la différence de l'Interface graphique, le Terminal est une **interface texte** : on tape des ordres au clavier. Ces ordres s'appellent des **commandes ✪**

À partir de maintenant, lors des TP/TD dans cette ressource R1.04, il vous est **INTERDIT** de faire des actions à l'aide de l'interface graphique quand on attend que vous les fassiez à l'aide de commandes dans le Terminal.

Au départ, cette utilisation d'un Terminal et de commandes pourra vous sembler archaïque. C'est sans doute normal et compréhensible si on considère les usages que vous avez faits d'un ordinateur jusqu'à présent. Ces outils sont très puissants. Ils ne sont pas du tout archaïques mais sont simplement réservés à des utilisateurs tels que les informaticiens, les développeurs (à ce stade, vous devriez vous sentir concernés, sinon on a raté un truc).

Ne vous inquiétez pas et ayez confiance que ce que vous allez apprendre à faire en Systèmes d'Exploitation vous sera très utile dans votre vie professionnelle. Pensez à ces journées économisées à (ne pas) traiter manuellement vos 1.000 photos ! 😀

C'est parti !

Contrairement à Windows, Linux n'est pas né avec une interface graphique dans le berceau. Cette interface lui a été ajoutée dans un second temps. On pourrait même dire *ces interfaces*, car il existe de nombreuses interfaces graphiques pour Linux. Si vous installez un Linux sur votre ordinateur personnel, vous aurez sans doute une interface un peu voire même très différente de celle proposée sur les postes de l'IUT, mais l'esprit reste commun entre toutes ces interfaces. Un peu comme le tableau de bord entre deux modèles de voitures, il y a des différences mais l'esprit général reste le même.

Linux est donc un système qui peut être démarré sans interface graphique. ✪

Un bon moyen de vous faire utiliser le Terminal en s'assurant que vous n'abuserez pas de l'Interface graphique (au moins dans cette ressource R1.04) aurait été simplement de démarrer Linux en *mode texte*, c'est-à-dire sans aucune interface graphique, mais on a jugé que ça serait quand même un peu violent comme idée pour démarrer.

Comme on est gentils et bienveillants, on a donc choisi d'y aller en douceur et de combiner les deux *mondes*, à savoir de lancer un Terminal depuis l'interface graphique, dans… une fenêtre ! Oui, c'est un peu paradoxal mais si ça vous gène vraiment, on peut facilement couper l'interface… d'ailleurs pour ceux que ça intéresse, vous trouverez en fin de TP un bonus pour expérimenter ça.

Du coup, pourquoi ouvrir un Terminal dans une fenêtre ? Déjà parce que ça vous permettra d'avoir le *PDF* du sujet du TP ouvert dans une autre fenêtre. Ça vous permettra aussi d'avoir accès à Internet si besoin, ou encore ça vous permettra d'ouvrir plusieurs fenêtres de Terminal par exemple.

Important : voyez cette fenêtre de Terminal comme un capot relevé qui donne accès au moteur de la voiture. Le Terminal, même s'il est *accueilli* dans une fenêtre, est un accès direct au Système d'Exploitation.

Cet accès au Système se fait à l'aide de **commandes**.

Commençons par localiser le Terminal dans votre interface. Vous avez le droit d'utiliser votre souris pour trouver, dans un des menus, le programme **Terminal**. Généralement il est dans **`Applications > Utilitaires`** ✪. Faites-vous aider par votre enseignant.e si vous ne le trouvez pas. En fonction du type d'interface graphique installée, le nom du Terminal peut varier ainsi que sa localisation dans le menu. Il est même possible qu'il y ait plusieurs programmes permettant d'ouvrir une fenêtre de Terminal. Ce n'est pas grave, ils ont tous le même fonctionnement.

Lancez (ouvrez) votre 1er Terminal et passez à la suite pour quelques explications.

Shell

On l'a évoqué précédemment, un Terminal est une interface entre vous et le Système.

On a aussi vu en CM que pour taper des *commandes*, on a besoin d'un **Shell**. Terminal ? Shell ? Je m'y perds…

En fait, ce n'est pas vraiment le Terminal qui fait l'interface, mais ce fameux Shell, un logiciel qui tourne dans le Terminal. Le Terminal est juste là pour *afficher* le Shell et lui *donner accès au clavier*.

C'est donc finalement le Shell qui fait réellement l'interface entre vous (par le clavier du Terminal) et l'*OS*.

Par la suite, on s'accordera un abus de langage en parlant parfois de Terminal au lieu de Shell. On va donc les considérer comme équivalents, par simplicité.

Prompt

Un Shell a un fonctionnement très simple : ✪

1) Il attend un **ordre**, qu'on appelle une **commande** (on va voir ça en détail un peu plus loin).
2) Il **exécute la commande** en question.
   L'exécution de cette commande peut donner lieu à des affichages voire, ce qui est assez rare, une certaine interaction avec l'utilisateur (affichage d'une question et saisie d'une réponse au clavier par exemple). Les affichages se font dans le Terminal.
3) Il se remet en **attente d'une autre commande** quand l'exécution de la précédente est terminée.

Quand il est en cours d'exécution d'une commande, il est occupé et ne peut pas recevoir d'autre commande. En tout cas, si on lui ordonne autre chose, il ne le fera qu'une fois l'exécution de la commande précédente terminée.

Quand il est en attente d'une commande, il affiche ce qu'on appelle un **prompt** ou **invite de commande** en français ✪. Ca se matérialise par l'affichage d'une ligne généralement terminée par un **$**

Vérifiez que vous avez bien un prompt dans la fenêtre de Terminal ouverte précédemment.

Ne vous préoccupez pas de la signification de ce qui précède le **$** pour le moment. On reverra ça un peu plus tard.

Votre 1ère commande

Il est maintenant temps de taper votre 1ère commande, vous êtes bien ici pour ça, non ?

```
date
```

Q.1 - Que vous affiche cette commande ? ✪

Q.2 - Identifiez les parties de ce qui vient de s'afficher.

Q.3 - Êtes-vous entré en interaction avec la commande (vous a-t-elle posé des questions) ?

Q.4 - Relancez-la une seconde fois. L'affichage a-t-il évolué ?

Pour relancer une seconde fois cette commande, vous avez certainement retapé la commande au clavier. Le Shell dispose d'un *historique des commandes* ✪ tapées et il suffit d'utiliser les *flèches vers le haut* et *vers le bas* sur le clavier pour vous déplacer dans l'historique. Appuyez sur **ENTRÉE** pour exécuter celle choisie. **CTRL+C** permet de quitter l'historique.

Vous pouvez aussi corriger une commande en vous déplaçant latéralement (flèche *vers la gauche* et *vers la droite* sur le clavier).

Testons autre chose. Attention, **ne faites pas de copier-coller** depuis le sujet de TP ! N'oubliez pas que par soucis de confort on vous autorise un environnement graphique mais qu'on travaille dans le Terminal comme si on était sans interface graphique.

Tapez tout manuellement au clavier et respectez les espaces s'il y en a, mais n'en mettez pas s'il n'y en a pas. Tout a son importance ! ✪

Si vous pensez avoir un message d'erreur, ou si vous avez un doute ou une incompréhension, appelez votre enseignant.e.

Tapez ceci :

```
date -u
```

Q.5 - Alors, qu'en pensez-vous maintenant ? Quel effet a eu l'ajout d'une partie **`-u`** ? Que peut-elle signifier ? Est-ce que le terme *UTC* vous parle ? Cherchez la signification de *UTC* sur Wikipedia (simplement *UTC*, pas la commande **`date`**, ni l'Université de Technologie de Compiègne)

Pour ceux qui ont été curieux et ont tenté de lancer une commande **`time`** (qui existe), vous avez peut-être été surpris d'un affichage qui ne semble pas du tout être l'heure courante. Hé oui, perdu. La commande **`time`** a un autre usage qu'on verra l'année prochaine, et qui n'a rien à voir avec l'affichage de l'heure courante, c'est d'ailleurs pour ça que **`date`** affiche aussi l'heure courante, elle sert pour les deux (date et heure).

Lisez attentivement (2 fois si nécessaire) le chapitre suivant, **Syntaxe d'une commande**.

Vous devez en comprendre tous les détails car c'est primordial pour la suite.

En cas de doute, appelez votre enseignant.e.

Syntaxe d'une commande

Vous venez de taper deux commandes similaires mais qui ont produit un résultat différent. Nous allons voir pourquoi.

Le lancement d'une commande est constitué de la façon suivante : ✪

```
commande option(s) paramètre(s)
```

- La partie **`commande`** est simplement l'ordre, l'action, qu'on souhaite exécuter. Dans l'exemple précédent (**`date`**), il s'agissait d'afficher la date et l'heure, vous l'aviez deviné et expérimenté. Évidemment, cette partie **`commande`** est obligatoire. ✪
- Une commande a toujours un *comportement par défaut*.

  La partie **`option(s)`** permet de modifier éventuellement ce comportement par défaut. ✪ Généralement, il peut y avoir 0, 1 ou plusieurs options qui se combinent alors. Mais si on ne précise pas d'option, c'est le comportement par défaut de la commande qui est obtenu. ✪

Il est très rare qu'une commande n'ait pas un comportement par défaut et qu'elle nécessite au moins une **`option`**. Dans ce cas, c'est une **`option`** qui n'est pas… *optionnelle…* héhé, bon on va dire *facultative*, pour ne pas embrouiller vos esprits ! Mais, c'est quand même rare, très rare.

Il est aussi très rare qu'une commande ne dispose pas d'au moins une option pour changer son comportement par défaut.

Dans l'exemple précédent, on a commencé par une simple commande **`date`** qui a affiché la date+heure du moment présent, c'est le comportement par défaut de la commande **`date`**. Puis on a exécuté un **`date -u`** qui a affiché cette même date+heure mais décalée d'une ou deux heures dans le passé, semble-t-il. En fait, il s'agit toujours de la date+heure courante mais au format *UTC*, qui est l'heure Internationale (vous avez dû le trouver sur Wikipedia précédemment). La présence du **`-u`** a ainsi changé le comportement par défaut de la commande.

Chaque option commence par un "**`-`**" (*tiret*, non non pas ~~*le tiret du 6*~~ on a dit ! D'ailleurs, certains disent aussi *moins* dans le cas des options). Ce *tiret* est suivi d'un *symbole*, généralement une *lettre* ou un *chiffre*, **sans espace** ✪ entre le *tiret* et le *symbole* (**`-u`** dans l'exemple avec la commande **`date`**). Attention les minuscules et les majuscules ne sont pas interchangeables, un **`-u`** n'est pas identique un **`-U`**. ✪

Ces options sont spécifiques à chaque commande. Un **`-u`** pour la commande **`date`** signifie quelque chose pour elle (ici l'affichage en mode *UTC*) mais un **`-u`** pour une autre commande signifiera autre chose, et peut même souvent ne pas exister du tout, ce que la commande vous dira certainement, par un message d'erreur, si vous lui passez une option inconnue pour elle.

Consultez le CM #1 qui explique comment on peut combiner certaines options et explique aussi qu'il existe des options longues, précédées de "**`--`**" (2 tirets) ✪ au lieu d'un seul.

- Enfin, la partie **`paramètre(s)`** indique généralement *sur quoi* la commande doit agir (souvent des fichiers ou des dossiers, on va le voir plus loin). ✪ La commande **`date`** n'est pas le bon exemple pour cette partie **`paramètre(s)`**, nous allons voir d'autres exemples concrets très bientôt. En fonction des commandes, il peut y avoir plusieurs **`paramètre(s)`** qui seront alors séparés par des espaces (on va voir des exemples plus loin, gardez-le dans un coin de votre mémoire).

De l'aide de super man

Voici maintenant sans doute la commande la plus importante à connaître sous Linux car c'est celle qui éclairera votre chemin dans les ténèbres du monde Linux…

Dis comme ça, c'est peu engageant mais c'est effectivement une commande qu'il faut **ABSOLUMENT** connaître et utiliser ! Il s'agit de la commande **`man`** ✪ qui signifie ***man****uel*.

Elle donne accès au manuel, à la documentation de toutes les commandes Linux présentes sur votre système.

Vous êtes bloqué sur une commande dont vous ne vous rappelez plus quelle option fait telle ou telle chose, ou quelle syntaxe vous devez utiliser pour réaliser telle action ? Ou inversement, vous vous demandez ce que fait telle option sur une commande ? Le réflexe est (doit être) d'aller consulter le *manuel* !

En effet, Google n'est pas la solution. La solution, vous l'avez sous la main. Il existe de nombreuses versions différentes du manuel et Google n'a aucune idée de la version de Linux que vous utilisez. Par contre, le manuel local à votre machine correspond évidemment exactement à ce qui y est installé, donc :

> Utilisez **man** et non pas Google ✪

Celui qui sera pris "la main dans le Google" subira les foudres de l'administrateur, on lui coupera l'Internet, puis ce sera l'Interface graphique et enfin on lui arrachera les voyelles et la touche **ENTRÉE** du clavier !

man est aussi une commande. Elle répond donc à la syntaxe vue précédemment. Expérimentons la consultation du manuel de la commande **date** :

```
man date
```

Très important : Ici **man** est la commande, alors que **date** est un *paramètre* passé à la commande **man**, et il n'y a pas d'*option* spécifiée (rappel, une option est précédée d'un *tiret*).

On a vu que le *paramètre* indique la *cible* de la commande, *sur quoi* elle doit agir, quel est son *sujet* de travail. Ici la cible est simplement un *texte* (le mot "date") indiquant le *nom de la commande* dont on souhaite voir le manuel (pour **man** la *cible* signifie *voici le nom d'une commande qui m'intéresse*). Il s'agit en l'occurrence de la commande **date**

Attention, il n'y a qu'une seule commande exécutée, c'est la 1ère partie (**man**). Le reste (**date**) qui est, rappelons-le, un *paramètre*, est une simple information *texte* qui informe la commande **man**. Donc même si **date** est aussi une commande qui existe (puisque, justement, on souhaite en consulter le manuel), elle n'est évidemment pas exécutée, ce ne serait pas ce que l'on souhaite. ✪

Voyez le *manuel* comme un gros dictionnaire. Si vous cherchez le mot *guerre* dans le dictionnaire, vous ne déclenchez pas une guerre. Là c'est pareil, vous ne déclenchez pas (comprendre : vous ne lancez pas) la commande **date** ! 😀

Si c'est flou, même un peu, relisez (une 3ème fois ?) le chapitre **Syntaxe d'une commande**. Au bout de la 10ème relecture, faites appel à votre enseignant.e, n'hésitez pas, c'est maintenant, pas dans 6 mois, qu'il faut manifester une éventuelle incompréhension.

Balade dans le manuel ✪

Maintenant que vous êtes dans le manuel, vous pouvez voir que votre Terminal est occupé à vous afficher du texte décrivant ce que fait la commande (**`date`** dans notre situation). Bon, c'est un peu ce qu'on attendait de lui...

Vous ne pouvez pas taper de nouvelle commande à exécuter, vous n'avez pas encore récupéré de *prompt*. C'est normal. On rappelle que le Terminal n'acceptera un nouvel ordre de votre part que lorsque la commande précédente sera terminée. En l'occurrence il s'agit de **`man`**, qui est encore en cours d'exécution. C'est une des rares commandes qui a un peu d'interaction avec l'utilisateur. Nous allons voir brièvement quelles interactions de base on peut avoir avec **`man`**.

Généralement le contenu du manuel dépasse la hauteur d'une fenêtre (quand on est en environnement graphique) ou d'un écran (environnement texte, voir chapitre Bonus en fin de TP).

Vous pouvez utiliser les flèches **haut** et **bas** du clavier pour vous déplacer ligne par ligne.

Vous pouvez utiliser la **barre espace** pour vous déplacer d'une page complète (i.e. une hauteur de fenêtre ou d'écran). Vous pouvez aussi appuyer sur la touche **f** comme *forward* (en avant, Guingamp ?).

Pour revenir en arrière d'une page complète, utilisez la touche **b** comme *backward* (en arrière)

Enfin, pour quitter le manuel, utilisez **q** comme *quit*. Et là, vous devez alors récupérer votre *prompt* dans le Terminal.

Un petit test

Q.6 - Consultez le manuel de la commande **`cal`** ✪. Que fait-elle par défaut ? Vérifiez en quittant d'abord le manuel et en lançant ensuite :

```
cal
```

Q.7 - Retournez dans le manuel de **`cal`** et cherchez-y la manière d'afficher :

- Sans la surbrillance du jour présent (ici il vous faudra peut-être utiliser la commande **`ncal`** qui est une version améliorée de **`cal`**, et qui conserve exactement les mêmes options et la même syntaxe, avec juste des options supplémentaires, merci les barbus)
- Le mois en cours encadré par le mois précédent et le mois suivant

- L'année en cours, complète (les 12 mois)
- L'année de votre naissance, complète

Évidemment il ne faut pas *que* rechercher, on attend aussi que vous testiez pour vérifier.

Se connaître soi-même

Comme **`man`** est une commande, naturellement **`man`** dispose aussi de son… *manuel* !

Q.8 - Consultez brièvement le manuel de **`man`**. Ça ne devrait pas être compliqué.

À noter qu'étant dans un environnement graphique (sauf pour ceux qui ont subi le courroux de l'administrateur), vous pouvez très bien ouvrir plusieurs Terminaux, avoir le manuel affiché dans l'un d'eux et taper vos commandes de test dans un autre. On vous l'autorise, c'est cadeau…

> Voilà, à ce stade vous savez consulter le manuel. Vous n'avez donc plus aucune excuse si vous ne trouvez pas comment fonctionne telle ou telle option d'une commande donnée.
>
> Votre enseignant.e pourra vous renvoyer vers le manuel s'il/elle juge votre question triviale (déjà vue, déjà répondue, flemme de chercher, pas pris de note)

Fichiers & Dossiers

Avant de poursuivre sur d'autres commandes de base de Linux, il nous faut évoquer quelques notions concernant les *fichiers* et les *dossiers*.

Nous reviendrons plus en détail sur ce sujet ultérieurement, mais comme en informatique on ne fait pas grand chose sans les fichiers, on doit quand même faire quelques rappels utiles pour pouvoir avancer.

On va considérer que ce sont effectivement des rappels car vous avez sans doute un ordinateur personnel ou avez déjà manipulé un ordinateur, au moins au Lycée, sans doute avec une interface graphique, on l'a évoqué précédemment, et sans doute était-ce sous *Windows*.

Les notions de *fichiers* et de *dossiers* doivent donc vous être familières.

On appelle cela le *Système de Fichiers* ou *File System* (**FS**). ✪

Fichier

Un fichier est une sorte de conteneur dans lequel se trouvent des données, des informations ✪. Vous en connaissez déjà certains types :

- Fichier image ou photo (JPEG ou PNG par exemple)
- Fichier vidéo
- Fichier son
- Fichier HTML
- Documents Word et Excel (ou Libre Office peut-être)

Notez que pour Word et Excel, on utilise plutôt le terme de document.

Un document est un fichier mais un fichier n'est pas forcément un document. La différence est subtile et il s'agit principalement d'une sorte de convention. Un document est généralement associé à une application, souvent bureautique, c'est-à-dire qu'il faut disposer de cette application spécifique pour lire, manipuler, créer ce type de fichier. ✪

Il n'y a pas de problème à appeler tout ça des fichiers mais évitez d'utiliser le terme de document pour tout.

Sous Linux, les fichiers sont très souvent des fichiers textes ✪, c'est-à-dire des fichiers dont on peut facilement lire le contenu sans outil spécifique. Un fichier HTML par exemple est un fichier texte. Les fichiers son, vidéo ou image ne sont pas des fichiers textes.

On va apprendre à manipuler principalement des fichiers textes dans cette ressource R1.04.

Quelques caractéristiques ✪

Un fichier a obligatoirement un *nom*. Exemples : **`etudiants`** ou encore **`starwars.mp4`**

Un fichier peut avoir une *extension*, qui se matérialise par un *point* suivi de *lettres* ou de *chiffres*. Exemples : **`tp1.c`** (le **`.c`** est l'extension). L'extension indique généralement la nature du fichier (ici il s'agit d'un code source en langage C)

Un fichier a une *taille* qui indique combien d'*octets* (on va dire des *caractères*, pour faire simple) il contient. Cette taille peut être nulle, le fichier ne contient rien, mais le fichier existe, comme une simple boîte vide.

Un fichier possède aussi une *date* de dernière modification (ou de création, au départ)

Il existe d'autres caractéristiques dont on aura l'occasion de parler mais pour le moment on va se limiter à ces quelques unes qui seront suffisantes pour la compréhension des exercices à venir

Dossier

Un dossier est aussi un *conteneur*, mais il ne contient pas des données comme les fichiers, il contient des *fichiers* et d'*autres dossiers* qui, à leur tour, peuvent contenir des fichiers et d'autres dossiers etc. Comme des poupées russes.

On représente souvent cette structure par un arbre dont les branches seraient les dossiers et les feuilles les fichiers. On y reviendra très en détail prochainement.

Cette notion de dossier doit donc aussi vous être familière. Sur votre ordinateur personnel, vous devez avoir un dossier *Documents* ou *Mes documents*. Vous avez peut-être aussi un dossier dans lequel vous placez vos photos ou votre musique ou encore votre collection de films téléchargés sur des sites douteux, mais passons, c'est le contenant qui nous importe ici, pas ce que vous y mettez !

À l'IUT vous avez chacun un espace de travail sur les ordinateurs. Cet espace de travail personnel est aussi un dossier, qui porte votre nom de login (on y reviendra). Il y a donc autant de dossiers d'espace personnel qu'il y a d'étudiants. Dans cet espace de travail personnel vous avez déjà des sous-dossiers, et notamment un nommé *Documents*

Quelques caractéristiques

Un dossier a peu ou prou des caractéristiques similaires à celles d'un fichier ✪ : *nom*, *extension*, et même *taille* et *date de modification* mais ces deux dernières sont un peu étranges quand on ne connaît pas leur fonctionnement, qu'on ne vas pas expliquer ici. On ne s'intéressera juste qu'à la partie *nom+extension* pour le moment, oubliez le reste.

Manipulation

Vous devez donc savoir qu'il est possible de réorganiser votre arbre de dossiers/fichiers, en déplaçant des éléments d'un endroit de l'arbre à un autre. On peut aussi renommer des éléments (dossiers et fichiers). On peut en créer, en supprimer, en dupliquer etc.

Vous savez très certainement le faire avec l'interface graphique.

Bien évidemment, on vous l'a déjà bien rabâché, vous savez qu'on va s'interdire d'utiliser l'interface graphique dans cette ressource R1.04.

Donc, à partir de maintenant, si dans un sujet de TP de Système vous devez créer un dossier, créer un fichier, supprimer un élément, déplacer des choses, etc., vous devez **IMPÉRATIVEMENT** taper une commande dans un Terminal pour faire l'action attendue.

Exception à la règle

Linux dispose de plusieurs programmes pour éditer des fichiers textes.

On ne peut pas dire que ces programmes brillent par leur ergonomie, surtout quand on n'a connu que des environnements graphiques.

On l'a déjà évoqué, vous n'aurez pas toujours à disposition une interface graphique pour travailler, surtout sur des machines distantes.

Cependant, nous avons décidé d'une entorse à la règle concernant l'édition de fichiers textes.

Vous avez donc le droit d'utiliser un éditeur graphique (*VSC*, Visual Studio Code est conseillé) pour saisir du texte dans des fichiers, pour les exercices de TP de cette ressource R1.04.

Par contre, il reste **INTERDIT** pour cette ressource R1.04 de supprimer, renommer des fichiers et des dossiers depuis VSC, même si ces actions sont possibles depuis l'interface de VSC.

Programme ? Commande ? ✪

Votre œil de lynx aura noté qu'on vient de parler de *programmes* alors qu'on avait jusqu'ici plutôt parlé de *commandes*. Voici donc une petite parenthèse sur le sujet.

Une *commande* exécute un ordre, une action simple : supprimer un fichier, créer un dossier. Il n'y a pas ou très peu d'interaction avec le donneur d'ordre. L'exécution est souvent très courte voire instantanée.

Un *programme* est plus complet et complexe : un traitement de texte, un navigateur Web. Ce n'est pas obligatoire, mais un programme a souvent une interaction possible avec son environnement, généralement l'utilisateur. Son exécution peut parfois durer des heures, et c'est l'utilisateur qui décide de quitter une fois qu'il a terminé de travailler avec son programme.

Il n'y a pas de règle absolue de différenciation (**`man`** est une commande qui peut aussi durer des heures tant qu'on ne la quitte pas), mais dans l'esprit, la différence est celle-là.

Manipulation de fichiers

Ouvrez un Terminal, si ce n'est pas déjà fait.

Quand vous ouvrez un Terminal, vous vous retrouvez **toujours** dans un dossier. Vous ne pouvez pas être *nulle part*. ✪

Le dossier par défaut de votre environnement de travail s'appelle votre *Home directory*. *Home* signifie *chez soi*, et *directory* signifie *répertoire* (qui est un autre nom pour dire *dossier*). ✪

On utilisera désormais indifféremment les termes *dossier* et *répertoire*, et on parlera de *Home* pour désigner ce répertoire par défaut, ce répertoire qui est la racine de tous vos dossiers et fichiers de travail personnels.

Où suis-je ?

Excellente question !

Q.9 - Vous avez ouvert un Terminal et vous avez bien noté ce qui est écrit un peu plus haut : dans un Terminal on est toujours ouvert dans un dossier. Mais, dans quel dossier ?

___

Vérifions à l'aide d'une nouvelle commande : ✪

```
pwd
```

qui signifie *print working directory* (afficher le répertoire de travail).

Ne pas confondre le *répertoire de travail* (*working directory*) et le répertoire par défaut de votre *espace personnel* (qu'on appelle parfois aussi votre *espace de travail,* désolé pour la similitude d'appellation), celui-là s'appelle votre *Home directory* (on vient juste de le voir un peu avant). ✪

Le répertoire de travail est le répertoire dans lequel vous êtes actuellement, dans votre Terminal. ✪

Cette notion de répertoire de travail est donc intimement liée à votre Terminal. Si vous ouvrez plusieurs Terminaux, vous aurez donc un répertoire de travail dans chacun d'eux et il pourra être (et sera sans doute) différent d'un Terminal à l'autre, une fois que vous aurez vu comment on se déplace d'un répertoire à un autre, un peu plus loin, on y vient.

Le résultat de **pwd** devrait vous donner une ligne du genre :

/home/etuinfo/votre_login ou peut-être **/FILER/HOME/INFO/votre_login**

C'est votre *Home directory*, en principe.

Création d'un fichier vide

Créez un fichier nommé **`test.txt`** à l'aide de la commande suivante :

```
touch test.txt
```

Q.10 - Est-ce que cette commande vous a affiché quelque chose de particulier quand elle s'est exécutée ?

Quand on est en mode *ligne de commande*, on est dans un mode de fonctionnement qui n'est pas celui de l'utilisateur lambda avec son interface graphique. N'attendez pas qu'une commande vous dise "Voilà, j'ai fait ce que tu m'as demandé, ça c'est bien passé, tu as maintenant un nouveau fichier qui s'appelle test.txt".

Le mode ligne de commande est un mode "J'ordonne, tu exécutes et je ne veux pas t'entendre, c'est moi le chef". Bon, c'est un peu caricatural mais l'idée est là : une commande s'exécute généralement en silence ✪. Si elle n'affiche pas un message d'erreur c'est que tout s'est bien déroulé, pas besoin de perdre du temps à raconter sa vie si tout va bien !

Jusqu'à présent, les quelques commandes qu'on avait lancées (**`date`**, **`man`**, **`pwd`**) étaient des commandes de questionnement (quelle est la date+heure du moment ? affiche-moi le manuel de telle commande, quel est le dossier dans lequel je suis ?). Il était donc normal qu'elles affichent un résultat à l'écran.

Cette commande **`touch`** permet de créer un fichier vide ✪, c'est ce qu'elle a fait (on va le vérifier dans quelques instants), et rien d'autre n'est attendu d'elle. Rappel : une commande fait généralement une action simple, ici la création d'un fichier. Point final.

Si on a besoin de vérifier que le fichier a été créé, et bien on va faire appel à une autre commande dont c'est le rôle, tout simplement. Allons-y :

```
ls
```

qui signifie ***lis****t* (lister les fichiers/dossiers). ✪

Q.11 - Qu'affiche cette commande pour vous ?

Il est très probable que vous ayez plus de choses affichées que ce que vous attendiez. Effectivement, vous avez déjà des fichiers (peut-être) et des dossiers (sûrement) dans votre *répertoire de travail*, qui se trouve être aussi votre *Home directory* si vous avez

suivi strictement les étapes de ce TP. Et votre *Home* contient déjà des choses, en principe.

En tout cas, vous devez voir apparaître le fichier **`test.txt`** dans la liste affichée par la commande **`ls`**. Si ce n'est pas le cas, demandez de l'aide à votre enseignant.e.

Notez que la commande **`touch`** est une commande assez sûre (ce qui est rare en mode ligne de commande). En effet, si un fichier du même nom existe déjà et contient des données, elles ne seront pas perdues.

Q.12 - De la même manière, créez un second fichier que vous nommerez **`proverbe.txt`**

Saisie de texte

Maintenant que le fichier **`proverbe.txt`** existe, saisissez-y un peu de texte à l'aide de VSC (que vous devez lancer depuis un menu de l'interface graphique Linux, cherchez-le).

Texte à saisir (cette fois-ci, vous pouvez faire du copier-coller si vous voulez) :

```
L'erreur est humaine, mais pour provoquer
une vraie catastrophe, il faut un ordinateur !
```

Puis, de la même façon, saisissez ceci dans le fichier **`test.txt`** :

```
Test - Ligne 1
Test - Ligne 2
```

Vous pouvez quitter VSC maintenant (si si, pour la suite c'est mieux de le faire, merci)

Affichage du contenu d'un fichier

Nous allons maintenant voir plusieurs façons d'afficher le contenu d'un fichier.

```
cat proverbe.txt
```

`cat` ✪ signifie *con**cat**enate* (concaténer), car elle permet d'afficher le contenu d'une liste de fichiers passés en paramètres. Les contenus s'affichant les uns derrière les autres, ils apparaissent donc concaténés les uns aux autres, d'où le nom de la commande. Evidemment, qui peut le plus peut le moins, en ne passant qu'un seul nom de fichier en paramètre, son contenu s'affiche, concaténé à… rien d'autre, mais le job est fait ! ✪

Note : la concaténation ne se fait qu'à l'affichage, les contenus des fichiers restent intacts.

Essayez donc ceci :

```
cat test.txt proverbe.txt
```

Q.13 - Comprenez-vous ce qui se passe ? Est-ce cohérent avec cette histoire de concaténation d'une liste de fichiers passés en paramètre ?

Q.14 - Testez en inversant les 2 paramètres.

Q.15 - Testez ceci maintenant :

```
cat test.txt proverbe.txt test.txt
```

Vous pouvez observer que, même si **`test.txt`** est présent 2 fois, chaque occurrence est prise en compte indépendamment.

La liste des paramètres est illimitée (ou presque).

Voyons maintenant une autre façon d'afficher le contenu d'un fichier

```
less proverbe.txt
```

`less` ✪ permet d'afficher le contenu d'un fichier de façon paginée. Contrairement à **`cat`** qui affiche intégralement le contenu d'un fichier, en bloc, peu importe la longueur du texte, **`less`** affiche le texte en s'arrêtant quand une page (de fenêtre ou d'écran) est atteinte. Il fonctionne exactement comme l'affichage dans **`man`** (les flèches permettant de défiler, etc.) ✪

C'est une commande très pratique pour afficher le contenu d'un fichier très long en permettant de se déplacer dans l'affichage comme on le souhaite. Et comme avec **`man`**, on peut quitter en appuyant sur la touche **`q`**.

Vous pouvez tester cela en créant, dans VSC, un fichier **`grosfic.txt`** que vous remplissez avec beaucoup de texte. Astuce : le plugin *Lorem ipsum* du *TP 0 - VSC* peut vous aider à créer un long texte (choisissez la création de **`10`** paragraphes, 2-3 fois de suite si besoin). Ensuite vous pouvez comparer l'affichage d'un :

```
cat grosfic.txt
```

avec un :

```
less grosfic.txt
```

Pour information, vous rencontrez aussi sûrement un jour la commande **more**, qui est l'ancêtre de la commande **less**. En dépit de son nom, **less** fait plus de choses que **more**. Certainement une blague de barbus.

Copie d'un fichier

Copier un fichier **f1** revient à dupliquer son contenu dans un second fichier portant un nom différent (nommé par exemple **f2**). S'il existe déjà un fichier **f2**, son contenu sera remplacé (on dit *écrasé*) par celui de **f1**. L'ancien contenu de **f2** est perdu. On se retrouve alors avec deux fichiers. Exemple :

```
cp proverbe.txt citation.txt
```

La commande **cp** signifie ***co**py*. ✪

Q.16 - Proposez une façon de vérifier 1) que le fichier a bien été dupliqué et 2) que le contenu de la copie est bien conforme.

Q.17 - On souhaite dupliquer le fichier **test.txt** en 2 autres exemplaires nommés **a.txt** et **b.txt**. Quelle solution proposez-vous ? Testez.

Q.18 - Voici un petit exercice et une question. Sans regarder le contenu des fichiers, répondez d'abord à la question et vérifiez ensuite :

```
cp proverbe.txt fic1
cp fic1 fic2
cp test.txt fic2
```

Question : que contient **fic2** maintenant ? 1 choix possible parmi a) à e). Pourquoi ?

a) Le contenu de **proverbe.txt**
b) Le contenu de **proverbe.txt** suivi du contenu de **test.txt**
c) Le contenu de **test.txt**
d) Le contenu de **test.txt** suivi du contenu de **proverbe.txt**
e) Les lignes de **test.txt** et **proverbe.txt** mélangées

Vérifiez votre réponse. Si vous ne comprenez pas pourquoi, faites appel à votre enseignant.e.

Renommage d'un fichier

Pour renommer un fichier, la commande porte un nom qui peut prêter à confusion parfois mais qui aura une explication un peu plus loin (on y reviendra).

La commande est la suivante :

```
mv citation.txt citation.old
```

mv signifie ***m**o**v**e* (*déplacer*) ✪. En fait, renommer un fichier revient à le déplacer, au même endroit, mais sous un nouveau nom. Encore une fois, cette bizarrerie s'expliquera mieux plus loin, quand on évoquera les commandes sur les dossiers.

On précise en *paramètres* le nom d'origine suivi du nouveau nom.

Préparons le terrain pour un exercice, tapez ceci :

```
cp proverbe.txt a.txt
cp test.txt b.txt
```

Donc **a.txt** contient les proverbes et **b.txt** contient les 2 lignes de test.

Q.19 - Vous devez réaliser un échange de nom entre **a.txt** et **b.txt** de telle sorte que **a.txt** contienne au final les 2 lignes de test et **b.txt** contienne les proverbes, et ceci en n'utilisant que la commande **mv**, mais en autant d'étapes que vous voulez.

Suppression d'un fichier

Attention, comme vous en avez maintenant l'habitude, en ligne de commande on est en mode "Exécute et tais-toi". Supprimer un fichier ne vous demandera pas de confirmation, on n'est pas sous Windows non plus, le fichier ne part pas à la corbeille où on pourrait le récupérer facilement. Là, il sera supprimé instantanément et définitivement. On joue dans la cour des grands maintenant, on travaille sans filet… À chaque fois que vous utilisez cette commande, vous devez donc avoir le réflexe d'une courte vérification visuelle avant de valider ✪. Ce conseil vaut surtout pour les frénétiques du clavier qui

tapent plus vite que leur ombre. Avertissement fait, le bureau des pleurs est au fond du couloir.

La commande à taper est la suivante :

```
rm citation.old
```

rm signifie ***rem****ove* (*retirer*, *supprimer*).

Q.20 - Supprimez, en une seule commande, les fichiers **a.txt** et **b.txt** (si vous ne les avez plus, recréez-les avant)

Manipulation de dossiers

Petit rappel : un dossier est un conteneur d'autres dossiers ou de fichiers ou des deux en même temps, bien entendu.

Autre petit rappel (oui, on se répète beaucoup mais on vous connaît trop bien…) : on travaille en ligne de commande, vous devez donc faire les actions demandées en tapant les commandes dans un Terminal, et ne pas utiliser l'interface graphique.

Les prochaines actions supposent que vous êtes dans votre *Home*.

Si vous avez suivi le sujet pas à pas jusqu'ici, vous devriez toujours être localisé dans votre *Home*. Faites un **pwd** pour vous en assurer. Si ce n'est pas le cas, le plus simple, à ce stade de vos connaissances, est de fermer votre terminal (via l'interface graphique, car on verra plus loin comment on quitte un Terminal au clavier) et d'en ouvrir un autre, puisque par défaut il s'ouvrira dans votre *Home*.

Création d'un dossier

Pour créer un dossier, on utilise la commande **mkdir**, qui signifie **m**a**k**e **dir**ectory ✪. Essayez avec l'exemple suivant :

```
mkdir systeme
```

Rappel : on évite les accents et les caractères exotiques (**@**, **/**, **espace**, **$**, *****, **?**, **&** etc.). En fait, on se limitera aux *lettres* (MAJ et min), *chiffres*, **-** (tiret), **_** (souligné), sous peine de galère ensuite, surtout en ligne de commande ! ✪

On commence à en avoir l'habitude, cette commande, comme de nombreuses autres, n'affiche rien quand tout s'est bien déroulé, ce qui doit être le cas dans l'exemple précédent. Comment faire pour s'assurer que le dossier a été créé ? On peut avoir confiance en l'ordinateur, mais on aime bien voir de ses yeux, parfois.

Q.21 - Proposez une solution.

Unicité de nom

Tentez de relancer exactement la même commande :

```
mkdir systeme
```

Q.22 - Que se passe-t-il cette fois-ci ? Pourquoi ?

Créez maintenant un fichier puis tentez de créer un dossier avec le même nom que le fichier :

```
touch docs
mkdir docs
```

Q.23 - Que se passe-t-il cette fois-ci ?

Un *objet* (terme générique pour parler d'un *fichier* ou d'un *dossier*, on va en reparler un peu plus loin) porte un nom et ce **nom est unique** et ne peut pas être affecté à un autre objet. Ceci n'est valable que dans le conteneur, le dossier qui contient ces objets (ici c'est votre *Home*). Deux objets dans deux conteneurs différents peuvent bien entendu avoir le même nom. ✪

C'est un peu comme dans une ville il ne peut pas y avoir deux rues avec le même nom mais dans deux villes différentes, ça ne pose pas de problème (rue Victor Hugo à Lannion, mais aussi à St Brieuc)

Suppression d'un dossier

Pour supprimer un dossier, on utilise la commande **`rmdir`**, signifiant **r**e**m**ove **dir**ectory ✪.

Essayez avec l'exemple suivant :

```
rmdir systeme
```

Q.24 - Vérifiez que le dossier a bien disparu.

Attention, on se rappelle que Linux en ligne de commande suppose que vous savez ce que vous faites. Il obéit aux ordres et ne vous demandera jamais (ou presque) une quelconque confirmation.

Concernant **`rmdir`**, il a la particularité de n'opérer que sur des dossiers vides (sans contenu, sans sous-dossiers ou fichiers à l'intérieur) ✪. On ne va pas entrer ici dans les détails mais sachez qu'il s'agit d'une raison technique et non pas d'un garde-fou.

Espace, la fausse bonne idée

On l'a évoqué, en ligne de commande, il faut absolument éviter de nommer les objets avec des espaces. C'est dommage car ça rend pourtant les noms plus lisibles parfois. Exemples : *Mes documents*, *Rapport de stage*.

Testez cette commande :

```
touch rapport de stage
```

Q.25 - Avez-vous eu une erreur ? Vérifiez ce qui a été créé. Avez-vous une idée de ce qui s'est passé ? (Un indice se trouve dans le chapitre **Syntaxe d'une commande**, quand on a parlé des *paramètres*). Faites-vous confirmer cela par l'enseignant.e.

Cela veut-il dire qu'on ne peut pas utiliser d'espace ? Non, c'est tout à fait possible, mais au prix d'une écriture un peu plus compliquée et laborieuse.

Il faut soit encadrer le paramètre par des **"** (*guillemets*) ou des **'** (*apostrophes*), soit faire précéder chaque espace par un **** (*backslash* ou *barre oblique inversée*). ✪ Exemples :

```
touch 'mes docs'
touch "mon rapport de stage"
touch sujet\ du\ tp1
ls
```

Comme vous le voyez, ça complique grandement les choses quand même !

Astuce : utilisez des `-` (tirets) ou, mieux, des `_` (underscores) ✪, qui sépareront les mots comme le font l'espace, mais sans le problème qu'à l'espace, qui sépare les paramètres.

Arborescence

Petit rappel, un dossier est un *conteneur* d'autres *dossiers* ou de *fichiers* ou des deux à la fois. Et comme un dossier peut contenir d'autres dossiers, évidemment ces autres dossiers peuvent à leur tour contenir d'autres objets, et ainsi de suite sans presque aucune limite. Il y a une limite imposée par le système, mais elle est tellement grande que votre propre logique vous arrêtera avant d'atteindre cette limite.

Par simplicité (on l'a déjà fait à plusieurs reprises et on va continuer de le faire) on va utiliser un terme générique pour parler indifféremment des *fichiers* et des *dossiers*. On va parler d'**objets**. C'est une convention entre nous, ce n'est pas un terme officiel et vous aurez l'occasion d'apprendre, plus tard, la Programmation Orientée Objet, qui n'a absolument aucun lien avec ces *objets*-ci.

L'arborescence est donc une imbrication d'objets. ✪

Déplacement dans l'arborescence

Jusqu'à présent, on a juste créé un dossier, puis on l'a supprimé mais on n'a rien déposé dedans et on est resté dans le *Home*.

Rappel : quand vous vous loguez, vous êtes dans votre *Home*, votre dossier d'accueil, qui porte votre nom de *login*. Répétons-le : vous êtes **dans** votre *Home*, tout comme vous êtes dans votre maison, ou dans votre appartement.

Si vous tapez :

```
ls
```

vous affichez la liste des objets qui sont **dans** votre *Home*. Vous ne voyez donc pas le dossier qui porte votre nom de *login* mais uniquement son **contenu**, tout comme quand vous ouvrez les yeux chez vous, vous ne voyez pas votre maison, mais son contenu (car vous êtes **dans** la maison).

Il convient de bien noter ça et surtout de bien le comprendre.

Quand vous ouvrez un *Terminal*, c'est un peu comme quand on se logue, le *Shell* se lance et vous vous retrouvez aussi dans votre *Home*. Par défaut, tous les Terminaux que vous ouvrirez vous placeront dans votre *Home* au départ.

C'est bien mais la plupart du temps vous aurez besoin d'aller ailleurs. Chez vous, vous ne vivez pas toujours dans la même pièce, enfin, à part dans les studios d'étudiant 😀

Se déplacer, chez vous, c'est passer d'une pièce à l'autre.

Se déplacer dans l'arborescence Linux (et sur bien d'autres systèmes), c'est passer d'un dossier à un autre.

IMPORTANT : on se déplace d'un *conteneur* à un autre. Les *conteneurs* sont les *dossiers*, pas les fichiers ! Les *fichiers* sont les *feuilles* (les éléments terminaux) d'un arbre, les *dossiers* sont les *branches* ✪, on passe d'une branche à une autre, d'un dossier à un autre.

La commande qui permet de changer de dossier est **cd**, pour **c**hange **d**irectory ✪ (*directory* = *répertoire*, un autre nom pour dire *dossier*)

Testons une série de commande (on parlera du détail juste après) :

```
pwd
ls -l -a [2]
mkdir d1
ls -la
cd d1
pwd
ls -la
```

Si vous avez un message d'erreur, c'est que vous avez mal tapé une commande, n'allez pas plus loin, tout doit s'enchaîner proprement, sinon ça ne servira à rien.

Bon, on a plusieurs choses qui se sont affichées, détaillons un peu tout ça, dans l'ordre :

- **`pwd`** : on affiche le chemin dans lequel on est actuellement, en principe c'est votre *Home*.
- **`ls -l -a`** : c'est la commande **`ls`**, qu'on connaît déjà, qui liste le contenu du dossier dans lequel vous êtes, mais cette fois-ci on a utilisé une option **`-l`** qui permet d'avoir un affichage détaillé ✪ (**`-l`** signifie : format **long**, jetez un œil au manuel dans un autre Terminal : **`man ls`**) ainsi qu'une option **`-a`** (exceptionnellement, ne regardez pas encore dans le **`man`** pour celle-ci). On y revient un peu plus loin.
- **`mkdir d1`** : on crée un dossier **`d1`**, là où on est, c'est-à-dire, dans le *Home*.

[2] C'est un **`L`** minuscule, pas le chiffre **`1`**

- **`ls -la`** : on voit maintenant apparaître notre nouveau dossier **`d1`**. Notez que des options peuvent être aussi écrites de façon condensée (**`-la`** et **`-l -a`** ou même **`-al`** ou **`-a -l`**, sont toutes des écritures équivalentes) ✪
- **`cd d1`** : on se déplace dans le dossier **`d1`** (on change de pièce dans notre maison)
- **`pwd`** : on doit voir que le chemin courant a changé, **`d1`** apparaît dans ce chemin, puisqu'on est arrivé **dans `d1`**
- **`ls -la`** : on affiche le contenu de l'endroit où on est, c'est-à-dire de **`d1`**. Qui doit être vide… Non ? Ah, y'a peut-être des choses dedans effectivement. On y vient tout de suite.

Cette dernière commande vous a peut-être questionné. Le dossier **`d1`** tout fraîchement créé contient déjà des choses ! Refaites un essai :

```
ls
ls -la
```

Bizarrement, le **`ls`** (sans option) n'affiche rien, le dossier est **`d1`** est vide, c'est plutôt le résultat logique attendu. Mais avec l'option **`-a`**, deux lignes apparaissent, ce qui est perturbant, la 1ère fois. C'est le **`-a`** qui a cet effet, le **`-l`** sert juste à un affichage long. Vous pourriez aussi écrire **`ls -a`** seulement, et voir apparaître ces deux objets, l'un derrière l'autre (car sans **`-l`**, pas d'affichage long, détaillé).

C'est la particularité de **`ls`**. Par défaut, il *ignore* les objets dits *cachés*. C'est une appellation abusive, ils ne sont pas vraiment *cachés*, ils sont simplement, et par convention, *masqués* de l'affichage par **`ls`**. Ces objets cachés (on va quand même les appeler ainsi), sont tous les objets dont le nom *commence* par un **`.`** (point) ✪

Pourquoi sont-ils masqués ? Généralement ce sont des *fichiers* et plus souvent des *dossiers* qui contiennent des *paramétrages de logiciels*. Un **`ls`** ignore leur présence car vous en avez rarement besoin et ça allège ainsi les listes de fichiers. Vous savez qu'ils sont là si besoin, mais dans l'ombre.

Tout ceci est bien beau, mais que font ces deux lignes dans votre dossier **`d1`**, supposé vide ?

En fait, **TOUS** les dossiers du Système de Fichiers contiennent **TOUJOURS** ces deux objets, qui portent les noms de **`.`** et **`..`**

Celui qui va nous intéresser ici est surtout **`..`** qui est un *autre nom* pour le *dossier parent* du dossier où vous êtes actuellement. Dans le cas présent, **`..`** c'est simplement votre *Home* puisque **`d1`**, dans lequel vous êtes actuellement a comme parent, votre *Home* (**`d1`** est dans votre *Home*).

Quel avantage en tire-t-on ? Tout dossier a un **`..`**, qu'on comprend aussi par : tout dossier à un *parent*. ✪ C'est un moyen simple de faire *référence* au parent sans avoir à

connaître son nom. Tout être humain peut dire “mon père”, pas besoin de le nommer par son nom.

Voici comment on peut sortir de **d1** pour retourner dans son dossier parent : ✪

```
cd ..
```

L’avantage d’une telle écriture de commande est que ça fonctionne toujours, peu importe où vous êtes, pour retourner dans le dossier parent (dans la branche du dessus)

On reviendra plus tard sur ce que représente l’autre ligne, celle qui affiche un **.**

Q.26 - A partir de votre *Home*, créez l’arborescence suivante avec les commandes que vous venez de voir. Si vous ne visualisez pas bien cette arborescence, demandez à l'enseignant.e avant de vous lancer :

```
(votre Home)
   |
   + doss1
      + doss2
      |  + doss3
      + doss4
```

Chemins

Un *chemin* c’est, par exemple, ce qui s’affiche quand vous tapez **pwd**. Cette commande affiche le chemin où vous êtes actuellement, on appelle cela le *chemin courant*, le “Vous êtes ici” qu’on peut trouver sur une carte urbaine affichée dans la rue ou parfois dans un abri bus.

Avec **pwd**, vous observez qu’un chemin est une *succession* de *dossiers* séparés par des **/** (*slash* ou *barre oblique*). Observez aussi que ce chemin commence par un **/**. Ce **/** de début matérialise la *racine* du Système de Fichiers (on l’appelle aussi le *root*, à ne pas confondre avec l’utilisateur *root*, le *Dieu du système*).

Chemin absolu

Tout chemin *commençant* par un **/** est appelé un **chemin absolu**. ✪

Il est appelé ainsi car un chemin **absolu** donné *mène toujours au même endroit*. C’est un peu comme votre adresse postale, peu importe à qui vous la donnez et d’où la personne part, si vous lui donnez votre adresse absolue, c’est-à-dire votre adresse complète (23

rue Victor Hugo, 22300 Lannion, France), elle arrivera chez vous, qu'elle parte de Paris, Le Caire ou Kiev, c'est **absolu**ment garanti !

Chemin relatif

Tout chemin *ne commençant pas* par un **/** est appelé un **chemin relatif**. ✪

Il est relatif par rapport à l'endroit où vous êtes actuellement. ✪ De ce fait, un même chemin **relatif** vous mènera à des *endroits différents* en fonction de là où vous en faites usage. Le meilleur exemple est simplement le **..** vu précédemment. C'est un chemin *relatif*, puisqu'il ne commence pas par un **/**. Ainsi on peut faire un **cd ..** depuis n'importe quel dossier, pour retourner dans le dossier parent, et pourtant la commande est toujours la même partout : **cd ..**

Un exemple de chemin relatif pour arriver chez vous pourrait être de dire à votre voisin.e du dessus : "j'habite à l'étage du dessous, porte de gauche". Si vous donnez cette même indication à quelqu'un qui n'habite pas dans votre immeuble, il aura sans doute aussi un "en dessous, porte de gauche" dans son immeuble, mais vous risquez d'attendre longtemps sa visite !

Pour remonter au parent du parent (donc au dossier grand-parent), il suffit de préciser le chemin ainsi (ne tapez pas cette commande, c'est juste pour info) :

```
cd ../..
```

Et ainsi de suite. On peut aussi écrire des choses comme celle-ci (ne tapez pas cette commande, c'est juste pour info) :

```
cd ../../docs/
```

On remonte au *parent du parent* (**../../**) et de là on descend dans le dossier **docs** qui s'y trouve.

Faisons un petit exercice. Tapez les commandes suivantes en **respectant scrupuleusement les espaces et surtout l'absence d'espace** (n'en mettez pas s'il n'y en a pas, relisez-vous avant d'appuyer sur *ENTRÉE* pour chaque commande) :

```
cd
mkdir -p D1/D2/D3 D1/D4 D1/D5/D3
tree D1
```

Quelques explications :

- **`cd`** utilisé tout seul, sans option ni paramètre, est une astuce pour vous déplacer instantanément dans votre *Home*, peu importe où vous êtes actuellement. ✪ Pratique. On s'assure donc ici de travailler dans le *Home*.
- **`mkdir`**, vous connaissez déjà, permet de créer un dossier. Avec une option **`-p`** (comme **p**ath), permet de créer des imbrications de dossiers, sans avoir à les faire un par un, comme vous avez dû le faire précédemment. ✪ Pratique aussi. Ici on crée donc **`D3`** dans **`D2`** lui-même dans **`D1`**. Idem avec **`D4`** dans **`D1`**, puis **`D3`** dans **`D5`** qui est lui-même dans **`D1`**.
  A noter qu'avec **`-p`**, seuls les dossiers inexistants sont créés. Ainsi, lors de la création de **`D4`**, le dossier **`D1`** existe déjà, car créé pour **`D1/D2/D3`**.
- **`tree`** permet d'afficher une vue arborescente d'un dossier et de tout ce qu'il contient. ✪

Q.27 - Proposez les chemins représentant les destinations suivantes :

- **`D1`** en relatif à partir de votre *Home*
- **`D4`** en relatif à partir de votre *Home*
- **`D1`** en relatif à partir de **`D2`**
- **`D2`** en relatif à partir de **`D4`**
- **`D1`** en relatif à partir de lui-même
- **`D3`** en relatif à partir de votre *Home*. Qu'avez-vous à dire sur ce cas particulier ?
- **`D5`** en absolu à partir de votre *Home*

Vous avez certainement proposé des commandes **`cd`** dans les exemples précédents. Mais, on vous demandait seulement les *chemins représentant les destinations* et non pas de vous y rendre. C'est une erreur classique quand on débute. Les chemins ne sont pas nécessairement et exclusivement utilisés avec la commande **`cd`**. Un chemin représente un *endroit de l'arborescence*, ✪ en relatif par rapport à l'endroit où vous êtes, ou en absolu. Un tel chemin n'est pas obligatoirement un endroit où on souhaite se rendre. Si on vous donne une adresse en ville, ce n'est pas forcément pour vous y rendre, c'est simplement une indication géographique d'un lieu. Ça peut être pour consulter Google Maps et y voir des photos, ça peut-être pour envoyer un colis, ça peut-être finalement pour plein d'autres raisons. Pour les chemins du Système de Fichiers, c'est pareil. Ce n'est pas toujours pour utiliser la commande **`cd`**, et c'est même souvent pour utiliser d'autres commandes en leur indiquant où elles vont trouver le ou les objets qu'elles doivent manipuler.

A ce sujet, préparons le terrain pour un nouvel exercice. On suppose que votre arborescence précédente est toujours là et proprement créée. Si tel n'est pas le cas, corrigez le problème et/ou faites-vous aider par l'enseignant.e :

```
cd
touch D1/F1 D1/D2/F2 D1/D2/F3 D1/D4/F4 D1/D5/D3/F5
tree D1
```

La commande **touch**, qui crée des fichiers, peut elle aussi, être utilisée avec plusieurs paramètres pour créer une série de fichiers mais il faut que les dossiers soient déjà existants, elle ne les créera pas et ça provoquera des messages d'erreur si ce n'est pas le cas.

Notez au passage que ces cinq paramètres (**D1/F1**, **D1/D2/F2**, **D1/D2/F3**, **D1/D4/F4** et enfin **D1/D5/D3/F5**) sont évidemment aussi des… chemins relatifs !!! On indique ainsi à **touch** l'endroit où créer ces cinq fichiers (**F1** à **F5**). Vous voyez, on les a utilisés pour la commande **touch** et non pas pour faire des **cd** avec. Les plus observateurs l'avaient déjà noté pour la commande **mkdir -p** précédente.

Supposons qu'en étant dans notre *Home*, on souhaite copier le fichier **F1** dans un nouveau fichier **F1bis**, au même endroit que **F1**.

1ère solution (la mauvaise) :

```
cd D1
cp F1 F1bis
```

Est-ce que ça fonctionne ? Pourquoi est-ce la mauvaise solution ? Réponse : parce que ! Bon, prenons un autre exercice pour mieux comprendre.

On souhaite copier **F1** dans un nouveau fichier **F1bis** mais cette fois-ci la copie doit être faite dans **D2**, et le tout en une seule commande, parce que sinon c'est pas du jeu !

Q.28 - Proposez une commande **cp** qui fait exactement cela sans bouger de votre *Home*.

Faites-vous confirmer par l'enseignant.e votre solution.

Comprenez-vous maintenant pourquoi la 1ère solution était mauvaise (ou moins bonne) ?

Ne vous déplacez pas sans raison dans un dossier pour manipuler un fichier. Vous ne vous déplacez pas chez votre grand-mère pour lui poster une carte de vœux, si ?

Utilisez les chemins directement avec vos commandes.

Sauf si vous avez besoin de faire d'autres choses dans le dossier en question évidemment, mais pour une action unique et ponctuelle, restez où vous êtes et utilisez les chemins comme paramètres de vos commandes.

Dans les quelques commandes précédentes, nous avons utilisé des *chemins vers des fichiers*. Si dans un chemin il y a un fichier, il ne peut être qu'en *dernière position*. ✪ Un fichier est un objet terminal (une feuille de l'arbre, on ne traverse pas une feuille). Quand un fichier est présent dans un chemin (donc en dernière position), ce chemin est donc un chemin vers un fichier. On ne peut pas faire un **`cd`** (pour s'y rendre) vers ce chemin, car **`cd`** travaille uniquement avec des *dossiers* (**c**hange **d**irectory, *changer de dossier*), mais d'autres commandes travaillent avec des fichiers, c'est le cas de **`touch`**, qu'on vient d'utiliser dans les exemples précédents.

La nature d'un chemin (vers un fichier ou vers un dossier) dépendra donc de la commande que vous mettez en œuvre avec le chemin en question.

Un peu de ménage

Nous vous conseillons de travailler proprement en séparant bien les différents modules (*Algo*, *Système*, *Web*, *BDD* etc) dans des *dossiers distincts*. Votre espace de travail n'est pas une chambre d'étudiant, il doit être rangé ! 😀

Créez un dossier **`systeme`** et déplacez-y tout ce que vous avez créé pour ce TP. En ligne de commande ? Oui, c'est mieux...

Nous vous conseillons **FORTEMENT** de **ne pas mettre d'espace dans vos noms de fichiers et de dossiers** et d'éviter les caractères accentués, au moins pour tout ce qui concerne cette ressource R1.04. Ce conseil vaut aussi de façon plus générale pour tout ce qui concerne le développement logiciel et le Web aussi. En fait, les espaces et accents sont OK pour la bureautique et c'est à peu près tout. Ceux qui ne suivent pas ce conseil s'en rendront compte assez rapidement. Rappel : si vous voulez mettre des espaces, utilisez plutôt des "**`_`**" (*soulignés*, *underscores*), exemple : **`fichier_de_travail.txt`**

Dans le dossier **`systeme`** que vous avez créé, créez-en un autre nommé **`tp1`** (en minuscules). À la prochaine séance, vous créerez un dossier **`tp2`** au même niveau que **`tp1`** etc. Pour l'Algo, vous ferez de même dans un dossier **`algo`**. Ainsi vous aurez un espace de travail propre et vous retrouverez facilement vos petits.

Bonus : mode sans interface

Sur vos PC sous Linux, il est possible d'obtenir un Terminal sans interface graphique (comme un Barbu !) en tapant simultanément les 3 touches **CTRL+ALT+F1** (possible généralement jusqu'à **F6** pour 6 consoles distinctes). A partir de là, vous devrez vous loguer pour pouvoir les utiliser.

Notez que si vous avez des chiffres dans votre mot de passe, utilisez la barre de chiffres au-dessus des lettres (avec la touche **SHIFT**) car il est possible que le pavé numérique ne fonctionne pas comme attendu à l'IUT.

Ne vous inquiétez pas, ça ne quitte pas l'environnement graphique qui reste simplement temporairement masqué.

Pour retrouver le confort de l'interface graphique, il suffit de taper simultanément les 3 touches **CTRL+ALT+F7**.

Avant de partir, n'oubliez pas de quitter vos consoles ouvertes si c'est le cas, sinon vous laissez la porte ouverte à celui ou celle qui arrivera derrière vous. Contrairement à l'interface graphique, il n'y a pas d'écran de veille avec demande de mot de passe sur les consoles textes.

Pour quitter une console, il suffit de taper la commande **exit** ou **logout**, ou encore d'appuyer simultanément sur les 2 touches **CTRL+D**.

Contrairement à une idée reçue, l'usage des consoles en mode texte ne fait pas pousser la barbe...