TP OS n°1

Sommaire

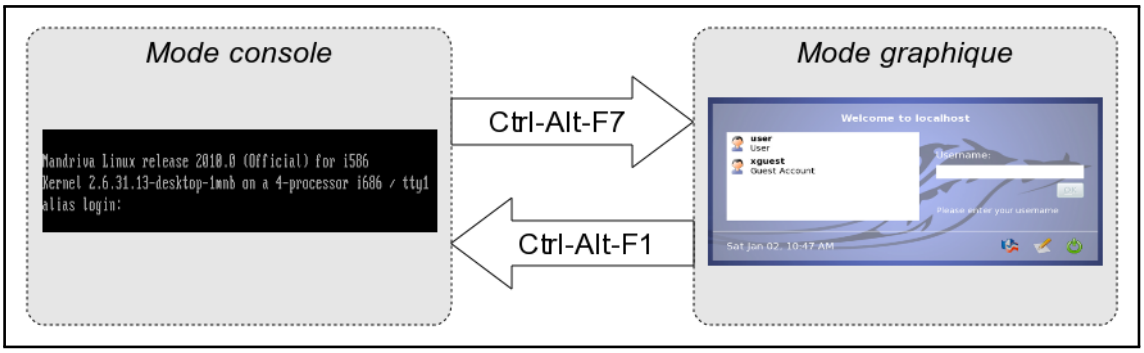
Les objectifs de ce premier tp sont d'être capable, en utilisant des commandes de base sous GNU/Linux, de manipuler des fichiers « texte » et d'en comprendre leur contenu.

Remarque : les TP *ont pour but d'établir ou de renforcer vos compétences pratiques. Vous pouvez penser que vous comprenez tout ce que vous lisez ou tout ce que vous a dit votre enseignant mais la répétition et la pratique sont nécessaires pour développer des compétences en informatique. Ceci est comparable au sport ou à la musique ou à tout autre métier demandant un long entraînement pour acquérir l'habileté nécessaire. Imaginez quelqu'un qui voudrait disputer une compétition dans l'un de ces domaines sans pratique régulière. Vous savez bien quel serait le résultat.*

Introduction

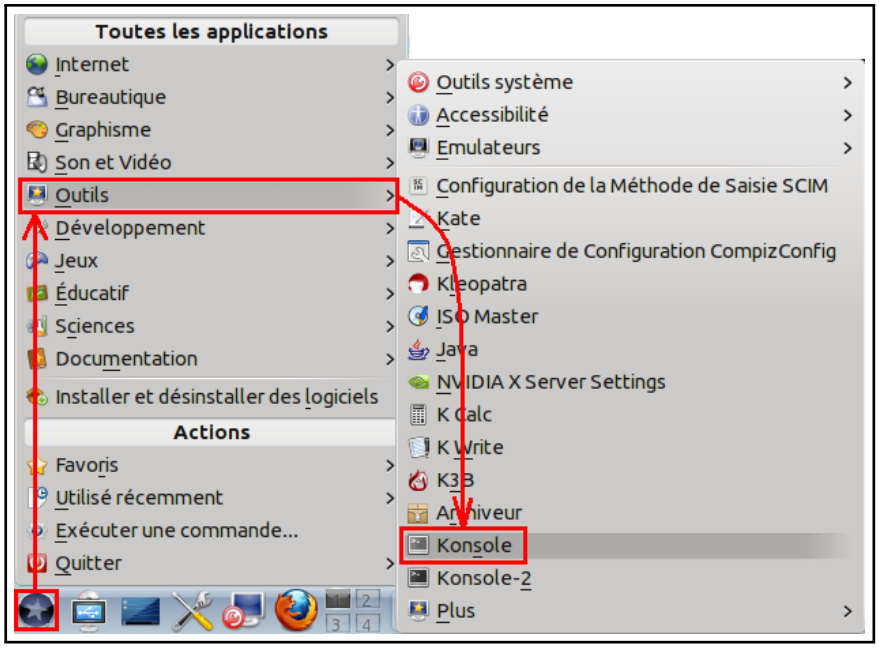
Environnement de travail

Il vous faut ouvrir une **session** sur votre poste de travail. Vous pouvez utiliser soit le mode console soit l'interface graphique. Dans les deux cas, vous allez travailler **« en ligne de commande »**.

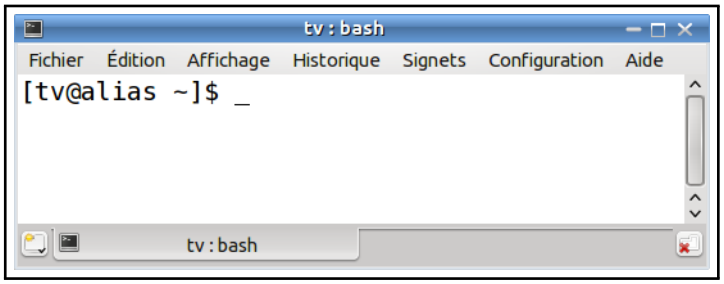


Ouvrir une session sur un système Mandriva

Remarque : Une « session » est l'ensemble des actions effectuées par l'utilisateur d'un système informatique, entre le moment où il se connecte à celui-ci et le moment où il s'en déconnecte.



Ouvrir une console sur un système Mandriva à partir de l'interface graphique



On peut maintenant travailler « en ligne de commande » à partir de l'interface graphique

Pour chaque TP, il vous faudra préalablement créer un répertoire de travail `tposx` où `x` est le numéro du TP correspondant. On va travailler dans l'arborescence suivante :

```
$HOME
     |
     - tpos
        |
        - tpos1
```

Pour créer l'arborescence de répertoires, il faut taper la commande suivante :

```
$ mkdir -p $HOME/tpos/tpos1
```

Pour se déplacer dans l'arborescence de travail, il faut taper la commande suivante :

```
$ cd $HOME/tpos/tpos1
```

Explications :

- invite ou *prompt* : Tous les exemples d'exécution des commandes sont précédés d'une invite utilisateur ou *prompt* spécifique au niveau des droits utilisateurs nécessaires sur le système. Toute commande précédée de l'invite `$` ne nécessite aucun privilège particulier et peut être utilisée au niveau utilisateur simple. Toute commande précédée de l'invite `#` nécessite les privilèges du super-utilisateur (`root`). Évidemment, il ne faudra jamais taper l'invite (`$` ou `#`) lorsque vous testez par vous même les commandes indiquées.
- `mkdir` : commande permettant de créer des répertoires s'ils n'existent pas
- `cd` : commande permettant de changer de répertoire courant
- `$HOME` : variable d'environnement contenant le chemin du répertoire personnel de l'utilisateur actuel

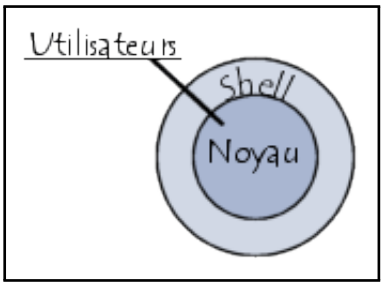
Shell Bash

Bash (*Bourne-again shell*) est le shell du projet GNU. Bash est un logiciel libre publié sous GNU GPL. Il est l'interprète par défaut sur de nombreux Unix libres, notamment sur les systèmes GNU/Linux. C'est aussi le shell par défaut de Mac OS X et il a été porté sous Windows par le projet Cygwin.

Aujourd'hui `bash` est le shell le plus répandu, bien qu'il existe beaucoup d'autres interpréteurs de commandes, comme `sh`, `ksh`, `csh`, `tcsh`, `zsh`, `ash`, ...

Un **shell** Unix, aussi nommé **interface en ligne de commande** Unix, est un shell destiné au système d'exploitation Unix et de type Unix. L'utilisateur lance des commandes sous forme d'une entrée texte exécutée ensuite par le shell. Celui-ci est utilisable en conjonction avec un terminal (souvent virtuel). Dans les différents systèmes d'exploitation Microsoft Windows, le programme analogue est `command.com` ou `cmd.exe`.

[Source Wikipedia]



Le shell (coquille) est une interface permetttant d'accéder au noyau (kernel) d'un système d'exploitation

Tout processus Unix/Linux démarre avec 3 **flux** déjà ouverts :
– un pour l'**entrée des données** (canal 0)
– un pour la **sortie des données** (canal 1)
– un pour les **messages d'erreur** (canal 2)

Remarque : un processus (identifié par un PID) est un programme en cours d'exécution.

Par défaut, ces flux sont :
– `0` : le **clavier** (*stdin* : *standard input*)
– `1` : l'**écran** (*stdout* : *standard output*)
– `2` : `/dev/null` (*stderr* : *standard error*)

Il est possible de rediriger ces flux (en utilisant les opérateurs `<`, `>`, `<<` et `>>`). Un tube (`|`) est un canal entre deux processus (redirection de la sortie d'un processus vers l'entrée d'un autre processus).

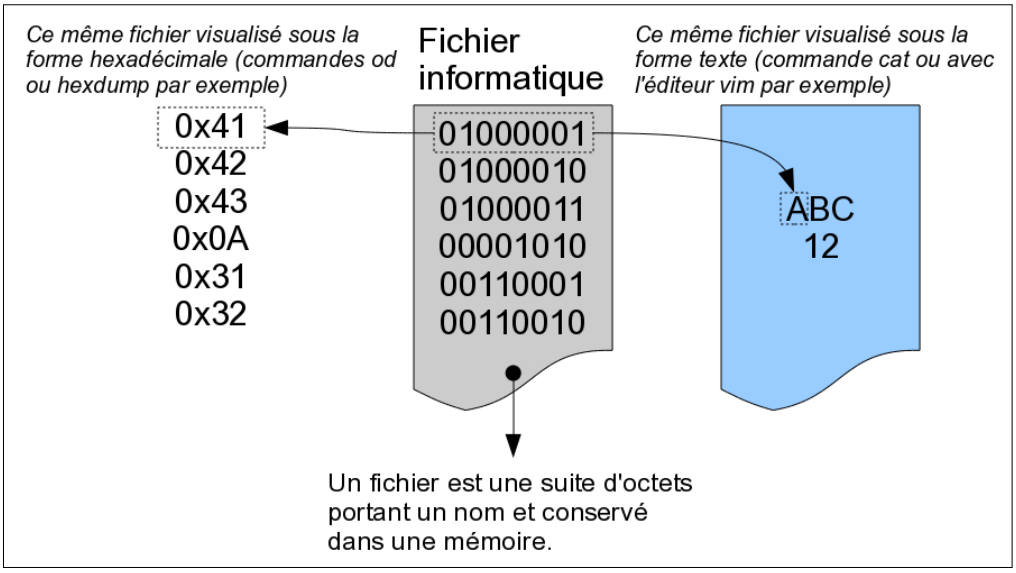
Les Fichiers « texte »

On distingue en général deux types de fichiers : **texte** et **binaire**.

Remarque : "Un fichier binaire est un fichier informatique qui n'est pas assimilable à un fichier texte." (source wikipedia). Donc, tout ce qui n'est pas un fichier texte est un fichier binaire.

Les fichiers texte ont un contenu pouvant être interprété directement comme du texte (une suite de bits représentant un caractère), la plupart du temps en codage **ASCII** (*American Standard Code for Information Interchange*).

Remarque : L'ASCII est la norme de codage de caractères en informatique la plus ancienne et la plus connue. Avec l'avènement de la mondialisation des systèmes d'information, son usage se restreint progressivement à des domaines très techniques.



Un fichier texte au "microscope"

Remarque : Comment sera interprété en ASCII l'octet 0x0A ? La réponse (et bien plus) est accessible dans le manuel en ligne de commande en faisant `man ascii`.

On utilise généralement un **éditeur de texte** (vi, **vim**, emacs, kwrite, kate, Notepad, Notepad++, UltraEdit, ...) pour manipuler ce type de fichiers.

Remarque : L'éditeur de texte est le programme le plus important et le plus utilisé par un informaticien dans l'exercice de son métier (administration, programmation). Comme on le verra par la suite, il ne faut pas confondre éditeur de texte et traitement de texte.

Quelques exemples de fichiers textes : code source d'un programme, fichiers de configuration, etc .

Remarque : Un fichier "Word" ou "OpenOffice" ne sera pas considéré comme un fichier texte par un informaticien.

Autres termes : fichier texte ou fichier texte brut ou fichier texte simple ou fichier ASCII.

Historique des commandes

Le *shell* permet de rappeler les commandes précédemment exécutées. Pour cela, vous pouvez utiliser les **flèches HAUT et BAS**.

```
// Visualiser l'ensemble de l'historique :
$ history
// ou
$ history | more

// Rechercher une commande :
$ history | grep commandeRecherchée

// Rappeler une commande et l'exécuter :
$ !ls      : rappelle la dernière commande commençant par ls
$ !100     : rappelle la commande n°100
$ !!    : rappelle la dernière commande
$ !10:p    : rappelle la commande n°10 et l'affiche (aucune exécution)

// Formes syntaxiques :
// !$ : correspond au dernier argument de la dernière commande
// !* : représente tous les arguments de la dernière commande sauf le premier
```

L'aide de la commande interne `history` se trouve dans :

```
$ help history

$ man bash
// Pour rechercher dans l'aide faire : /history puis on se déplace avec n (en avant) ou N (
    en arrière)

// Ou :
$ man bash | colcrt | egrep -A 5 history
// Les options -A (After) -B (Before) -C (autour) -n (numéro de ligne) de la commande egrep
```


Manipulations

Objectifs

L'objectif de cette partie est d'être capable de réaliser les manipulations de base sur des fichiers « texte » sous GNU/Linux. Il est important de savoir que l'interaction de l'utilisateur (clavier-écran) avec le système d'exploitation Unix/Linux se fait au format **texte**.

Remarque : Avant d'utiliser un éditeur de texte, on va utiliser quelques commandes de bases bien utiles.

Étape n°1 : créer un fichier texte

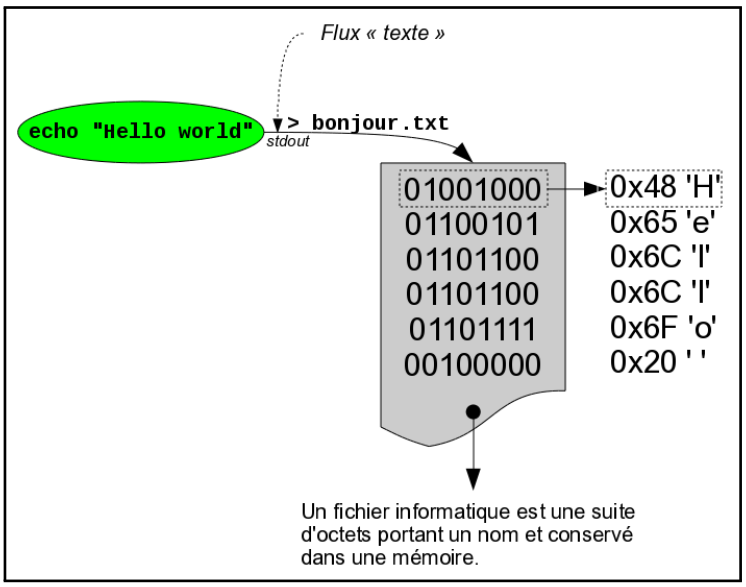
– vide :

```
$ touch vide
$ ls -l vide
-rw-r--r-- 1 tv tv 0 2010-07-17 15:56 vide
$ file vide
vide: empty
```

– avec un contenu :

```
$ echo "Hello world" > bonjour.txt
$ ls -l bonjour.txt
-rw-r--r-- 1 tv tv 12 2010-07-17 15:55 bonjour.txt
$ file bonjour.txt
bonjour.txt: ASCII text
```

Remarque : Les redirections d'entrées/sorties
Par défaut, les commandes récupèrent les données tapées par l'utilisateur au clavier (`stdin`). Le résultat de leur exécution s'affiche à l'écran (`stdout`). En cas d'erreur à l'exécution, les messages d'erreur apparaissent aussi à l'écran (`stderr`). Il est possible d'indiquer à l'interpréteur de commandes de rediriger ces flux d'E/S vers (ou depuis) un fichier. Par exemple : `> sortie` signifie que les données générées par la commande seront écrites dans le fichier de nom `sortie` plutôt qu'à l'écran. Si le fichier `sortie` existait déjà, son ancien contenu est effacé, sinon ce fichier est créé au lancement de la commande.



Utilisation d'une redirection de `stdout` (>) vers un fichier

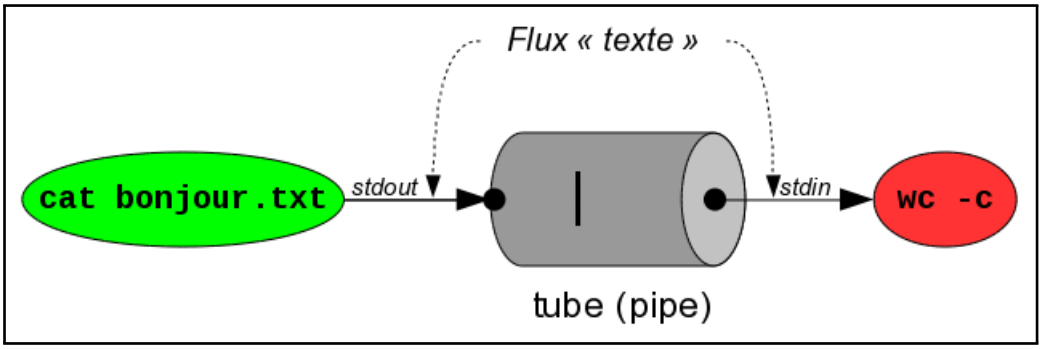
Étape n°2 : afficher le contenu d'un fichier texte

Il existe de nombreuses possibilités pour afficher le contenu d'un fichier texte. En voici quelques-unes :

```
$ cat bonjour.txt
$ cat -n bonjour.txt ; nl bonjour.txt
$ strings bonjour.txt
$ more bonjour.txt
$ less bonjour.txt
```

Sous Unix/Linux, il est possible de "relier" des commandes :

```
$ cat bonjour.txt | wc -c
```



Utilisation d'un tube (pipe) en ligne de commande

Remarque : Le shell Unix dispose d'un mécanisme appelé ***tube*** *(ou pipe). Ce mécanisme permet de chaîner des processus (commandes en cours d'exécution) de sorte que la sortie d'un processus (`stdout`) alimente directement l'entrée (`stdin`) du suivant. Le symbole utilisé pour créer des tubes dans les shells Unix est la barre verticale `|`, appelée communément pipe. Le pipe est très utilisé sur Unix pour associer plusieurs commandes dont on enchaîne les traitements. C'est un mécanisme de communication inter-processus (IPC).*

Une dernière qui illustre bien la philosophie UNIX/Linux :

```
$ while read ligne ; do echo "contenu : $ligne"; done < bonjour.txt
```

Remarque : Les redirections d'entrées/sorties
Ici l'utilisation de `<` permet de rediriger le flux d'E/S depuis un fichier (`bonjour.txt`).

Étape n°3 : examiner le contenu d'un fichier texte

Un fichier texte contient fondamentalement une suite de bits. La particularité d'un fichier texte est que l'ensemble du fichier respecte un codage de caractères standard. Il existe de nombreux standards de codage de caractères, ce qui peut rendre problématique la compatibilité des fichiers texte.

La norme ASCII (*American Standard Code for Information Interchange*) est la norme de codage de caractères en informatique la plus connue et la plus largement compatible. L'ASCII définit 128 caractères numérotés de 0 à 127 et codés en binaire de 0000000 à 1111111. Sept bits suffisent donc pour représenter un caractère codé en ASCII. Toutefois, les ordinateurs travaillant (presque) tous sur huit bits (un octet), chaque caractère d'un texte en ASCII est stocké dans un octet dont le 8e bit est 0. Les caractères 0 à 31 et le 127 ne sont pas affichables. Ils correspondent à des caractères (commandes) de contrôle de terminal informatique.

Pour en savoir plus :
– `man ascii`
– fr.wikipedia.org/wiki/Ascii

Pour afficher le contenu brut d'un fichier (texte ou binaire), on utilisera soit la commande `od` soit la commande `hexdump` :

```
$ od -ca -t x1 bonjour.txt
$ hexdump -C bonjour.txt
```

Étape n°4 : modifier le contenu d'un fichier texte

Le système d'exploitation ne permet que de très simples modifications d'un fichier : on peut soit modifier un (ou plusieurs) octet soit ajouter des octets en fin de fichier.

Vous pouvez modifier 'w' en 'W' :

```
$ hexedit bonjour.txt
$ cat bonjour.txt
```

Remarque : il est impossible en utilisant les services de l'OS de supprimer ou d'insérer du texte dans un fichier (sauf à la fin). Ce sont des opérations bien trop complexes car elles nécessiteraient un décalage d'un ensemble d'octets dans le fichier. Pour réaliser cela, il faut soit utiliser un éditeur de texte soit écrire soi-même un programme équivalent.

Ou on peut ajouter du texte à la fin du fichier :

```
$ date +"le %A %d %B %Y à %T" >> bonjour.txt
$ echo "by $USER" >> bonjour.txt
$ cat bonjour.txt
```

Remarque : Les redirections d'entrées/sorties
`>> sortie` semblable à la redirection `>` sauf que si le fichier `sortie` existait déjà, son ancien contenu est conservé et les nouvelles données sont copiées à la suite.

Questions de révision

L'idée de base des questions de révision est de vous donner une chance de voir si vous avez identifié et compris les points clés de ce TP.

Question 1. Quel est le rôle du *prompt* ?

Question 2. Quelle est la définition d'un fichier informatique ?

Question 3. Quels sont les deux catégories possibles pour classifier un fichier ?

Question 4. Quel est le rôle d'un *shell* ?

Question 5. Qu'est-ce que `bash` ?

Question 6. Est-il possible de supprimer de caractères dans un fichier texte en utilisant les services de base de l'OS ?

Question 7. Que signifie l'extension `.txt` à la fin d'un nom de fichier ?

Question 8. Que fait la commande `mkdir` ?

Question 9. Quelle est la différence entre un fichier texte et un fichier binaire ?

Question 10. Qu'est-ce qu'une session de travail ?

Travail demandé

Exercice 1 : manipulation avec des commandes de base

L'objectif de cet exercice est de manipuler des fichier textes à partir des commandes de base d'un système Unix/Linux.

Question 11. Que font les commandes suivantes ? (on suppose le fichier `bonjour.txt` non vide)

a) `$ wc -l bonjour.txt`

b) `$ sort bonjour.txt`

c) `$ tac bonjour.txt`

d) `$ head -1 bonjour.txt`

e) `$ tail -2 bonjour.txt`

f) `$ md5sum bonjour.txt > bonjour.md5`

g) `$ md5sum -c bonjour.md5`

h) `$ echo "fin" >> bonjour.txt`

i) `$ md5sum -c bonjour.md5`

j) `$ touch bonjour.txt`

k) `$ cat bonjour.txt | tr -s ' ' '.'`

Question 12. Donnez la ligne de commande qui permet d'écraser le contenu du fichier `bonjour.txt`.

Exercice 2 : deux programmes qui collaborent

L'objectif de cet exercice est d'apprendre à d'adapter des commandes pour ses propres besoins.

Question 13. Il n'existe pas de commande qui permet d'afficher une seule ligne quelconque d'un fichier texte. Donner la ligne de commande qui affiche uniquement la deuxième ligne du fichier `bonjour.txt`.

Exercice 3 : comptage de caractères

L'objectif de cet exercice est de comprendre la notion de caractère.

Question 14. On désire compter le nombre de caractères d'un fichier texte. Que permettent de faire les commandes suivantes ?

a) `$ echo "Hello world" > bonjour.txt`

b) `$ wc -c bonjour.txt`

c) `$ cat bonjour.txt | wc -c`

d) `$ ls -l bonjour.txt`

Question 15. La chaîne de caractères "Hello world" contient 11 caractères. Pourtant, la commande `wc` compte 12 caractères contenus dans ce fichier. Expliquez cette différence ?

Exercice 4 : surveillance de fichiers

L'objectif de cet exercice est de s'initier au rôle d'administrateur.

L'administrateur d'un système est souvent amené à surveiller visuellement le contenu de certains fichiers (par exemple les fichiers de *log* ou de journalisation).

Pour réaliser cet exercice en pratique, il est demandé d'utiliser deux consoles : une pour le suivi du fichier et l'autre pour le modifier.

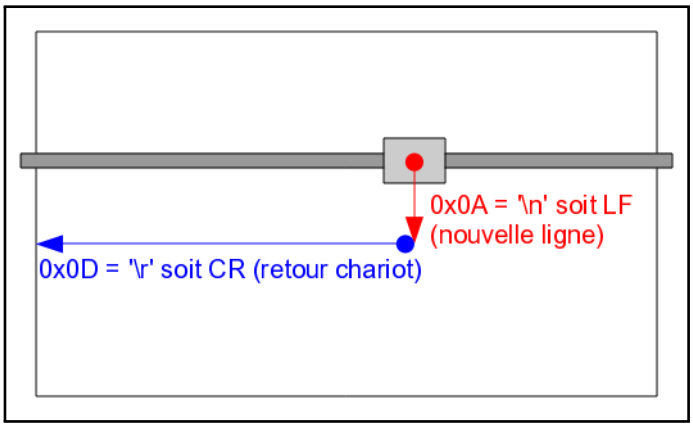
Question 16. Donnez l'option à utiliser avec la commande `tail` pour assurer le suivi du fichier `bonjour.txt`.

Exercice 5 : format des fichiers texte ?

L'objectif de cet exercice est de comprendre les différences entre OS concernant les fichiers texte.

En fait, les fichiers texte n'ont pas de structure car ce ne sont qu'une suite d'octets encodant des caractères.

Par contre, la notion de "fin de ligne" est ambiguë. Historiquement, cela provient des premiers terminaux qui nécessitaient deux actions pour un "saut de ligne" :



Principe simplifié du saut de ligne sur un télétype (TTY)

Dans un fichier texte, la fin d'une ligne est représentée par un caractère de contrôle (ou une paire). Plusieurs conventions coexistent :
- sous les systèmes Unix/Linux, la fin de ligne est indiquée par une nouvelle ligne (LF, 1 octet) ;
- sous les machines Apple II et Mac OS jusqu'à la version 9, la fin de ligne est indiquée par un retour chariot (CR, 1 octet) ;
- sous les systèmes CP/M, MS-DOS, OS/2 ou Microsoft Windows, la fin de ligne est indiquée par un retour chariot suivi d'une nouvelle ligne (CR+LF, 2 octets).

Ainsi, lorsque l'on ouvre un fichier ASCII créé par un système sur un autre système, il faut en général faire de la mise en forme (c'est-à-dire refaire les fins de ligne) afin de pouvoir l'afficher et le lire de manière confortable. Mais les éditeurs de texte intelligents (ce qui n'est pas le cas du classique Notepad même sur les derniers Windows) peuvent détecter le type de fin de ligne et agir en conséquence. Les programmes utilisant les fichiers ASCII ne sont en général pas perturbés par un changement de type de fin de ligne ce qui permet d'échanger des fichiers texte entre OS différents.


```
$ echo -e -n "Hello World\nBienvenue le monde\n" > bonjour_unix.txt
$ file bonjour_unix.txt

$ echo -e -n "Hello World\r\nBienvenue le monde\r\n" > bonjour_dos.txt
$ file bonjour_dos.txt
$ vim bonjour_dos.txt
```

*Remarque : pour quitter l'éditeur **vim**, faire* `Echap` *puis* `:q`

Question 17. Observez en bas de page la détection du format `[dos]` par l'éditeur **vim**. En vous aidant de l'aide (`help echo`), que permet l'option `-e` de la commande interne `echo` ? Et l'option `-n` ?

Exercice 6 : l'encodage des caractères

L'objectif de cet exercice est de comprendre les limites et les différences des encodages des caractères à l'intérieur des fichiers texte.

Les éditeurs de texte peuvent créer des fichiers texte avec l'encodage de caractères de leur choix. Un codage de caractères définit une manière de représenter les caractères (lettres, chiffres, symboles) dans un système informatique.

Le premier codage largement répandu fut l'ASCII. Pour des raisons historiques (les grandes sociétés associées pour mettre au point l'ASCII étaient américaines) et techniques (7 bits disponibles seulement pour coder un caractère), ce codage ne prenait en compte que 27 soit 128 caractères. De ce fait, l'ASCII ne comporte pas les caractères accentués, les cédilles, etc. utilisés par des langues comme le français. Ceci devint vite inadapté et un certain nombre de méthodes furent utilisées pour l'étendre.

L'ISO a donc défini de nouvelles normes, ISO 8859-1, ISO 8859-2, etc. jusqu'à ISO 8859-15. Ces jeux de caractères permettent de coder la plupart des langues occidentales. Le français utilise le plus souvent ISO 8859-1, aussi nommé latin1, ou ISO 8859-15 (latin9), qui a l'avantage de contenir des caractères (ligatures) comme le « œ » ou le symbole « € ».

Il est indispensable pour l'échange d'information de connaître le codage utilisé. Ne pas le savoir peut rendre un document difficilement lisible (remplacement des lettres accentuées par d'autres suites de caractères, ...).

Le besoin de supporter de multiples écritures demandait un nombre nettement plus élevé de caractères supportés et nécessitait une approche systématique du codage de caractère utilisé. Le codage Unicode (a pour ambition d'être un surensemble de tous les autres, et est souvent représenté en UTF-8 ou en UTF-16.

L'UTF-8, spécifié dans le RFC 3629, est le plus commun pour les applications Unix et Internet. L'UTF-16 est utilisé par Java et Windows.

La norme internationale ISO/CEI 10646 définit l'*Universal Character Set* (UCS) comme un jeu de caractères universel. Ce standard est le fondement d'Unicode. Environ 10 000 caractères (symboles, lettres, nombres, idéogrammes, logogrammes) sont recensés dans l'UCS.

L'Unicode (ISO 10646) est un standard destiné à représenter sans ambiguïté tous les signes écrits de toutes les langues humaines connues. La structure de l'Unicode offre 21 bits pour chaque caractère.

Question 18. Déterminez l'encodage utilisé sur votre session. Pour cela, on recherche (`grep`) le contenu de la variable d'environnement (`env`) `LANG` :

```
$ env | grep LANG
LANG=fr_FR.UTF-8
```

```
// Les messages d'erreurs seront donc en français (fr_FR) :
$ ls fff
ls: ne peut accéder fff: Aucun fichier ou dossier de ce type

// Tiens amusons-nous à changer la langue :
$ LANGUAGE=en_US.UTF-8 ls fff
ls: cannot access fff: No such file or directory
```

Linux représente l'**Unicode** en utilisant le format de transfert sur 8 bits (**UTF-8**). L'UTF-8 est un **codage à longueur variable**. Il utilise un octet pour coder 7 bits, 2 octets pour 11 bits, 3 octets pour 16 bits, 4 octets pour 21 bits, 5 octets pour 26 bits, 6 octets pour 31 bits.

Il est conseillé pour la suite de consulter les pages de manuel suivantes :

```
$ man ascii
$ man iso_8859-1   (et man iso_8859-15)
$ man utf-8
$ man unicode
$ man charsets
```

L'encodage UTF-8 (un encodage Unicode multi-octets compatible ASCII) a les propriétés suivantes :
- le jeu ASCII classique est encodé simplement par les octets 0x00 à 0x7f (compatibilité ASCII). Ceci signifie que les caractères du jeu ASCII 7 bits ont exactement le même codage en ASCII et en UTF-8.
- Le premier octet d'une séquence multi-octets représentant un caractère UCS non ASCII est toujours dans l'intervalle 0xC0 à 0xFD et indique la longueur de la séquence multi-octets. Tous les octets suivants de cette séquence sont dans l'intervalle 0x80 à 0xBF.

Encodons en UTF-8 une chaîne de caractères contenant le caractère 'à' :

```
$ date +"le %A %d %B %Y à %T" > bonjour.txt

$ cat bonjour.txt
le mardi 17 juillet 2012 à 12:42:51

$ file bonjour.txt
bonjour.txt: UTF-8 Unicode text

$ hexdump -C bonjour.txt
00000000 6c 65 20 6d 61 72 64 69 20 31 37 20 6a 75 69 6c |le mardi 17 juil|
00000010 6c 65 74 20 32 30 31 32 20 c3 a0 20 31 32 3a 34 |let 2012 .. 12:4|
00000020 32 3a 35 31 0a                                 |2:51.|
```

Remarque : L'ASCII (jeu standard sur 7 bits) n'est pas modifié par UTF-8, et les gens utilisant uniquement l'ASCII ne remarqueront aucun changement : ni dans le codage, ni dans les tailles de fichiers.

Question 19. À partir de l'affichage fourni par la commande `hexdump`, donnez la valeur des deux octets qui encodent le caractère 'à'.

Pour les utilisateurs de l'ISO-8859-1 (latin1), ceci signifie que les caractères avec le bit de poids fort à 1 (jeu étendu sur 8 bits) sont désormais codés sur deux octets. En UTF-8, un octet `110xxxxx` représente le début d'un code sur 2 octets, et `110xxxxx 10yyyyyy` est assemblé en `00000xxx xxyyyyyy`. Sinon il n'y a pas de problèmes particuliers car les symboles Unicode correspondant aux caractères ISO-8859-1 conservent les mêmes valeurs (étendues avec 8 bits à zéro en tête). Donc ici, la valeur xxyyyyyy correspondra au caractère ISO-8859-1.

Question 20. Donnez alors la valeur assemblée des deux octets qui encode le caractère 'à'.

Il existe plusieurs commandes sous Linux qui permettent de convertir des fichiers texte d'un encodage vers un autre : `iconv`, `recode`, etc ...

On va maintenant convertir le fichier texte `bonjour.txt` (qui est en UTF-8) en **ISO8859-1** (latin1) :

```
$ iconv -f UTF-8 -t ISO8859-1 bonjour.txt -o bonjour_latin1.txt

// oups !
$ cat bonjour_latin1.txt

$ hexdump -C bonjour_latin1.txt
00000000 6c 65 20 6d 61 72 64 69 20 31 37 20 6a 75 69 6c |le mardi 17 juil|
00000010 6c 65 74 20 32 30 31 32 20 e0 20 31 32 3a 34 32 |let 2012 . 12:42|
00000020 3a 35 31 0a                                     |:51.|
```

Question 21. Que se passe-t-il lors de l'affichage du fichier avec la commande `cat` ?

Question 22. À partir de l'affichage fourni par la commande `hexdump`, donnez la valeur qui encode le caractère 'à' en ISO8859-1.

Question 23. Correspond-elle à la valeur trouvée précédemment lors de l'assemblage réalisé en UTF-8 ?

La commande `iconv -l` permet de lister l'ensemble des jeux codes connus et supportés.

Question 24. Donnez alors la ligne de commande permettant de fournir (approximativement, c'est-à-dire sans tenir compte des alias) le nombre de jeux codes connus et supportés par la commande `iconv`.

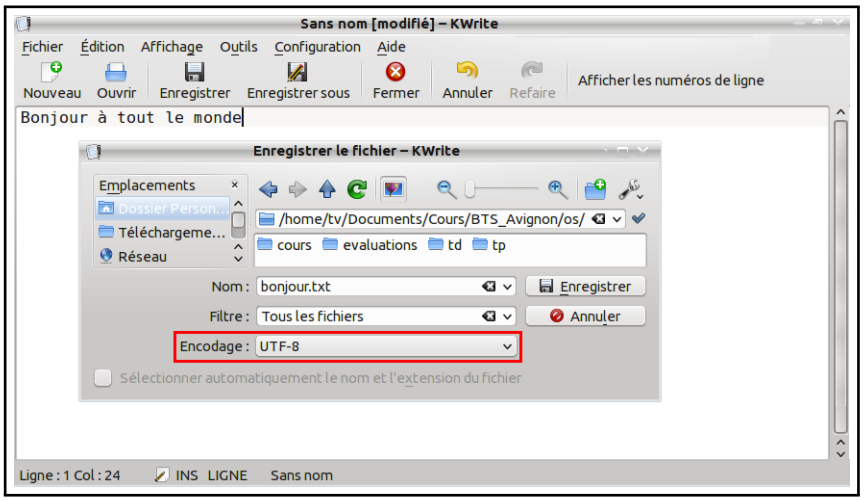
Question 25. Idem mais pour la commande `recode`.

Bilan

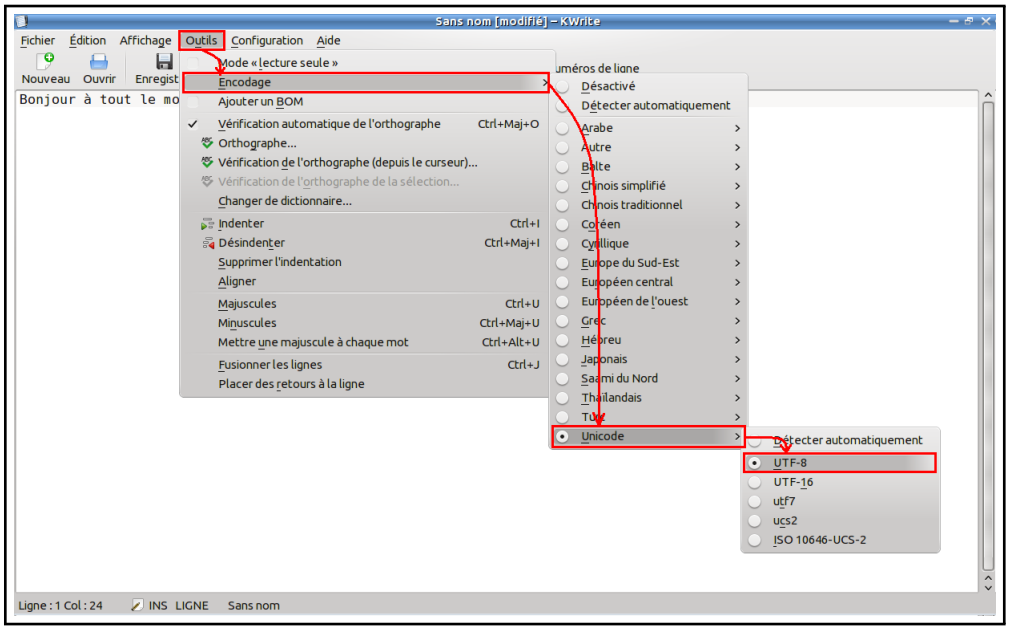
Ces dernières manipulations sur l'encodage des caractères prouvent que la situation n'est pas encore stable. Les problèmes se régleront probablement avec l'uniformisation Unicode. Mais, il y a des risques dans le cas d'échange entre systèmes hétérogènes. Cela concerne :
– l'utilisation des flux de texte dans les programmes
– les échanges sur internet
– les noms de fichiers et de répertoires

Par précaution (et le technicien informatique est prudent !), il est donc conseillé de ne jamais utiliser de caractères étendus ou spéciaux (comme l'espace) dans les noms de fichiers et de répertoires, de privilégier l'encodage Unicode et d'être cohérent avec les fichiers qui permettent de déclarer l'encodage utilisé (cas des fichiers **html** et **xml** par exemple).

Exemple : choix de l'encodage avec l'éditeur de texte kwrite



On peut choisir l'encodage au moment de l'enregistrement du fichier



Ou pendant l'édition du fichier

