COURS PROJET ENCADRE - PLURITAL.ORG

Projet "*La vie multilingue des mots sur le web*"`

Préambule Unix (suite)

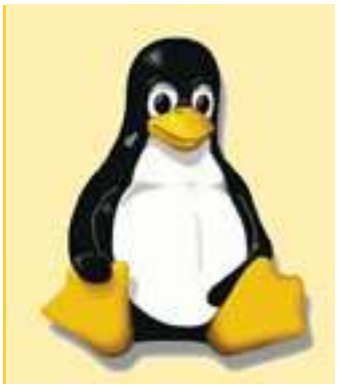
A LIRE ABSOLUMENT :

- **Initiation au shell : LAB1413_TP1_Initiation_Shell.pdf** :: *2 MB*
- **Initiation au scripts BASH : LAB1413_TP2_Scripts_Bash.pdf** :: *1.62 MB*

(ces 2 documents sont disponibles sur iCampus)

Documents réalisés par Antoine Gademer (professeur d'informatique ESIEA)
http://professeurs.esiea.fr/gademer/.

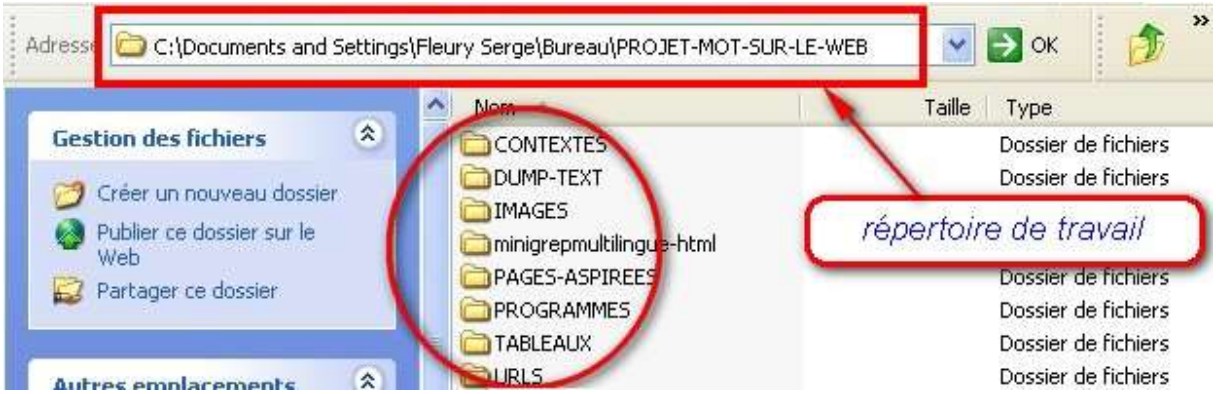
(des extraits de ces 2 documents sont insérés ci-dessous)



Le projet se déroule dans un environnement de travail tel qu'il est décrit ici :

http://www.tal.univ-paris3.fr/cours/PROJET-MOT-SUR-LE-WEB/

rubrique : REPERTOIRE ET ARBORESCENCE DE TRAVAIL



Pour préparer cet environnement de travail

1. on aura récupéré le script fourni : http://www.tal.univ-paris3.fr/cours/PROJET-MOT-SUR-LE-WEB/prepare-environnement-projet.sh que l'on aura mis dans son répertoire utilisateur
2. On éditera le script précédent, on le corrigera
3. On l'exécutera

In fine, l'arborescence de travail devra avoir l'allure décrite par le schéma précédent (modulo vous travaillez sous Ubuntu ou sous MacOSX, ce qui peut induire quelques ajustements…)

Rappel : syntaxe commande unix

D. Bouillet, D. Conan, F. Silber-Chaussumier — Sept. 2005 — TélécomINT/UX11

- La syntaxe générale d'une commande Unix est la suivante :

  `nom [ - options ] [ argument1 ... ]`

  où `nom` est le nom de la commande ;
  et `options` représente une ou plusieurs options ;
  et `argument1` est le premier argument.
- Les options sont composées d'un seul caractère suivant un tiret.
- Il est possible d'accoler plusieurs options (donc, plusieurs caractères)
  Par exemple, `-asli` pour les options `-a -s -l -i`.
- Si l'option demande un paramètre, il est séparé par un espace comme dans `-o fichier`.
- Les crochets désignent un élément facultatif, ils ne doivent donc pas être tapés.
- Les points de suspension indiquent la possibilité de répéter un élément
  Par exemple, `ls /etc /usr` pour plusieurs arguments.
- Dans une commande, chaque mot est séparé des autres par un espace ou une tabulation.

Chemin

Référence absolue : la référence absolue consiste à fournir le chemin **de la racine** jusqu'au fichier.

Exemple :

```
$ pwd
/home/Fleury Serge
```

Référence relative : la référence relative consiste à fournir le chemin **du répertoire courant** jusqu'au fichier.

Exemple :

```
Fleury Serge@imacsf ~
$ pwd
/home/Fleury Serge

Fleury Serge@imacsf ~
$ ls ../../bin
```

Références spéciales :

./	Le répertoire courant
../	Le répertoire parent
~/	Le dossier personnel de l'utilisateur

Exécuter un script

Sur un ordinateur, nous devons souvent réaliser des tâches récurrentes, par exemple :
- renommer un groupe de photos, les réduire, faire une archive et l'envoyer par mail,
- supprimer tous les fichiers temporaires d'un dossier et des sous-dossier pour faire de la place,
- se connecter à un parc de machine, les mettre à jour, puis produire un rapport d'activité,
- récupérer des images satellites, leur appliquer un traitement, puis générer une vidéos de la séquence, etc.

Or ces répétitions manuelles sont pénibles et fastidieuses... mais surtout très faciles à éviter !

La description même du problème décrit **un algorithme** : nous allons alors créer des programmes de commandes `Shell` appelés scripts `Bash`.

Les scripts `Bash` sont des **fichiers textes** qui commencent par la ligne :

```
#!/bin/bash
```

Remarque

Il est d'usage d'utiliser l'extension `".sh"` pour les scripts `Shell`, `Bash`, etc., mais ce n'est pas obligatoire.

Les scripts regroupent plusieurs commandes **Shell** qui seront exécutées les unes à la suite des autres. Par exemple :

```
#!/bin/bash
# Ceci est mon premier script Bash !
mkdir monDossier
cd monDossier
touch fichier1
touch fichier2
```

Remarque

Le caractère # marque les lignes de commentaires. Tout ce qui est écrit après # ne sera pas considéré comme une commande, ce qui nous permet d'écrire des commentaires explicatifs!
Attention, la première ligne, elle, n'est pour autant pas facultative!

Pour exécuter un script :

Solution 1 :

```
sh nomduscript.sh
```

Solution 2 :

```
chmod +x nomduscript.sh

./nomduscript.sh
```

- **Afficher les permissions**

$ ls -l doit.sh

- **Kezako ?**
  - r lecture
  - w écriture
  - x exécution, accéder au répetoire

-rwxr-xr-x
u g o

  - u utilisateur
  - g groupe
  - o autres utilisateurs

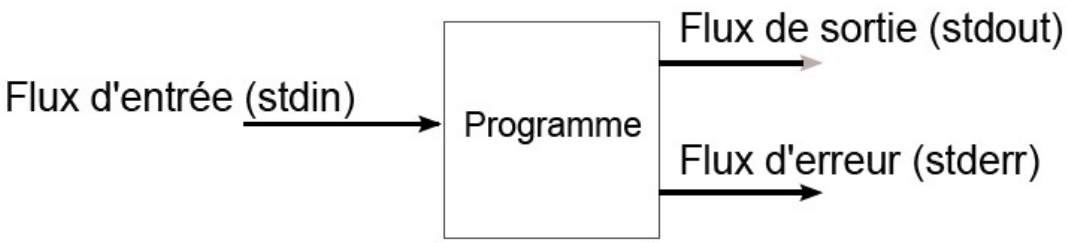
Phase 1 : les entrées –sorties

Les informations envoyées au programme sont appelées **flux d'entrée** du programme (`stdin` pour STandarD INput).

Les informations renvoyées par le programme sont appelées **flux de sortie** du programme (`stdout` pour STandarD OUTput).

Il existe un troisième flux, appelé **flux d'erreur** (`stderr` pour STandarD ERRor) et qui reçoit les messages d'erreur renvoyés par le programme.

Par défaut, le flux d'entrée correspond à la saisie clavier, alors que les flux de sortie et d'erreurs sont affichés dans le `Terminal`.



D. Bouillet, D. Conan, F. Silber-Chaussumier — Sept. 2005 — TélécomINT/UX11

(sources)

À la connexion, le shell dispose de trois flots de communication :

- Entrée standard : `stdin` (numéro 0)
- Sortie standard et erreur standard : `stdout` et `stderr` (numéros 1 et 2)

L'association par défaut de ces flots est l' écran pour `stdout` et `stderr`, et le clavier pour `stdin`.

Une redirection est une modification de l'une ou de l'autre de ces associations.

- Elle est valable uniquement le temps de la commande sur laquelle elle porte.

Ainsi, la redirection de la sortie standard permet de récupérer le résultat dans un fichier[a] : `commande > nom_fich`. Par exemple,

```
$ who > qui                                   ⟹ rien à l'écran
$ cat qui                                     ⟹ fichier qui créé
  domy  console Aug 2 09:45
  paul  ttyp2   Aug 2 12:17 (celia)
  clode ttyp3   Aug 3 11:52 (:0.0)
```

Attention, avec « > », si le fichier de redirection existe, son contenu initial est perdu.

```
$ date > qui
$ cat qui
  Fri Sep 13 17 :11 :22 MET DST 1996
```

La redirection double (`commande >> nom_fich`) permet de ne pas détruire le fichier existant, mais ajoute le nouveau contenu en fin de fichier.

```
$ pwd >> qui
$ cat qui
  Fri Sep 13 17 :11 :22 MET DST 1996
  /mci/inf/domy/public
```

Avec « >> », si le fichier n'existe pas, il est créé, comme pour une redirection simple.

Moins utilisée que la redirection de la sortie standard, la redirection de l'entrée standard (`commande < nom_fich`) permet à une commande d'utiliser comme données le contenu d'un fichier à la place d'une lecture clavier.
Les exemples d'utilisation sont moins évidents.
Exemple avec la commande `write` :

```
$ write paul                                   ⟹ texte lu au clavier
  un petit bonjour
  ^D                                                  touche CTRL-D
                    ⟹ Message reçu par paul contenant « un petit bonjour »
  Création d'un fichier contenant le message :
$ cat > message                                ⟹ texte lu au clavier
  un autre bonjour
  ^D                                                  touche CTRL-D
$ write paul < message                  ⟹ redirection de l'entrée standard
                    ⟹ Message reçu par paul contenant « un autre bonjour »
```

Table 1 – *Opérateurs Shell liés à la redirection.*

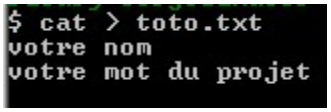
Opérateur	Commande	Effet
>*	`CMD > FILE`	Redirige la sortie de CMD vers le fichier FILE (Écrase le fichier préexistant).
>>*	`CMD >> FILE`	Redirige la sortie de CMD vers le fichier FILE (Ajoute à la fin du fichier).
2>	`CMD 2> FILE`	Redirige le flux d'erreur de CMD vers le fichier FILE (Écrase le fichier préexistant).
2>>	`CMD 2>> FILE`	Redirige le flux d'erreur de CMD vers le fichier FILE (Ajoute à la fin du fichier).
<*	`CMD < FILE`	Redirige le fichier FILE sur le flux d'entrée de CMD (À la place du clavier).
|*	`CMD1 | CMD2`	Redirige la sortie de CMD1 vers le flux d'entrée de CMD2.

⇨ On aussi regardera les transparents suivants : http://www.tal.univ-paris3.fr/cours/Plurital_Formation_Unix.pdf , page 15

Application : à réaliser par vous

Liste de commandes à taper dans votre fenêtre de commandes :

```
mkdir TEST1

mkdir TEST2

comment créer un repertoire qui s'appellerait « TEST 3 » ?

cd TEST1

touch vide1.txt  ./TEST1/vide2.txt ../TEST2/vide3.txt

echo «votre nom » > vide1.txt

echo « votre mot du projet » >> vide1.txt

mv vide1.txt jenesuisplusvide.txt

cp jenesuisplusvide.txt ../TEST2

cp jenesuisplusvide.txt moinonplus.txt

ls
```




```
(tapez control-D pour sortir)

ls

ls > liste.txt

ls >> liste.txt

lsd >> liste.txt 2> erreur.txt

ls >  ../TEST2/liste.txt

ls >> ../TEST2/liste.txt

rm vide2.txt

rmdir ../TEST3
```

Listes des commandes à étudier via le man : touch, echo, cp, mv, rm, rmdir

autres commandes à exemplifier :

less

Enchaînement de commandes

Un tube (*pipe*) est une zone mémoire permettant à deux processus d'une même machine de communiquer tout en étant synchronisés.

- L' écrivain attend une place disponible pour écrire.
- Le lecteur attend une information disponible à la lecture.

L'objectif d'utiliser un tube est de faire agir une commande sur le résultat d'une autre sans fichiers intermédiaires.

Le symbole « | » placé entre deux commandes redirige la sortie standard de la première sur l' entrée standard de la seconde.

```
$ who | wc -l
  3
```

Cela évite des entrées/sorties disques plus coûteuses :

```
$ who > qui                    ⟹ création d'un fichier temporaire
$ wc -l qui                    ⟹ traitement de ce fichier
  3
$ rm qui                       ⟹ suppression du fichier temporaire
```

Application

Liste de commandes à taper :

Compter le nombre de fichier dans votre répertoire de travail ? Compter le nombre de fichier dans le répertoire TEST2 ?

Listes des commandes à étudier via le man : wc

Exécution séquentielle : le séparateur « ; » permet d'enchaîner des commandes sans relation entre elles.

`$ ls ; who ; pwd` ⟹ exécution en séquence de `ls`, `who` et `pwd`

Regroupement : `(...)` permet de considérer les commandes incluses comme une seule pour un tube ou une redirection.

`$ (date ; who) > /tmp/qui` : redirection des 2 commandes

Imbrication : entre « $(» et «) » ou entre anti-*quotes* « ‘ », la commande à exécuter est remplacée par son résultat.

- Cela permet d'utiliser le résultat d'une commande comme argument d'une autre.

`$ echo Je suis sous $(pwd)`

1. Exécution de la commande `pwd`
2. Exécution de la commande
   `echo Je suis sous` *`résultat de pwd`*

Application

Liste de commandes à taper : vous testerez les commandes précédentes…

Phase 2 : rappel HTML

L'Hypertext Markup Language, généralement abrégé HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage qui permet d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias dont des images, des formulaires de saisie, et des éléments programmables tels que des applets. Il permet de créer des documents interopérables avec des équipements très variés de manière conforme aux exigences de l'accessibilité du web. Il est souvent utilisé conjointement avec des langages de programmation (JavaScript) et des formats de présentation (feuilles de style en cascade). HTML est initialement dérivé du Standard Generalized Markup Language (SGML).

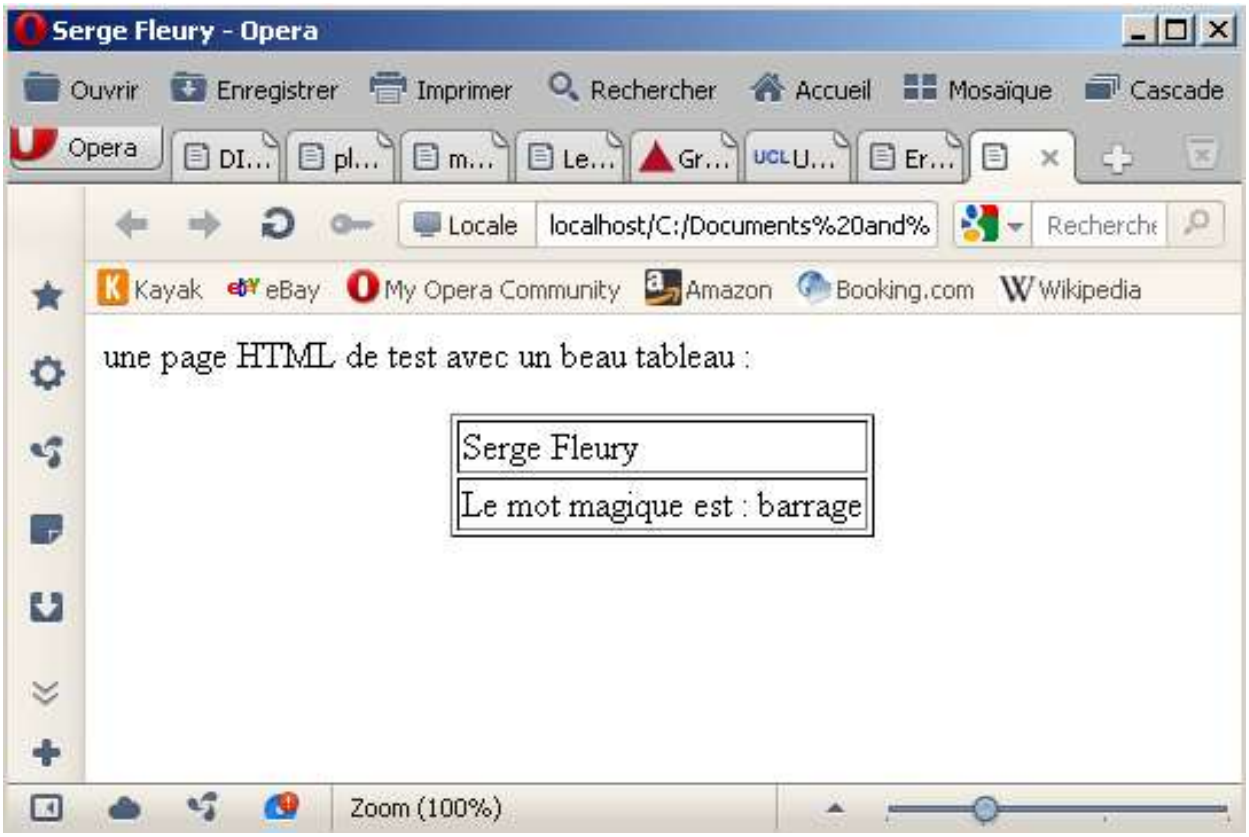
Source : http://fr.wikipedia.org/wiki/Hypertext_Markup_Language

Exercice :

Construire une page HTML contenant les éléments suivants :

- ⇨ Titre de la page : votre nom
- ⇨ Contenu de la page : un tableau avec 2 lignes (sur la première, votre nom, sur la seconde, le mot choisi pour votre projet)

Résultat attendu :



Lecture : Charset iso-8859-1, iso-8859-15, utf-8, lequel choisir ?

http://www.alsacreations.com/astuce/lire/34-charset-iso-8859-1-iso-8859-15-utf-8-lequel-choisir.html

- En HTML5 : `<meta charset="UTF-8">`
- En HTML4 : `<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >`

Phase 3 : au travail

Mémo

« Tout ce que vous faites sur l'interface graphique peut se faire en commande `Shell`. »
« A la main c'est fastidieux, alors faites un script ! »
« Un script complexe peut se décomposer en petits scripts simples chaînés. »

A. Ecrire un script bash (que vous mettrez dans le dossier PROGRAMMES) permettant de générer dans le répertoire DUMP-TEXT un fichier txt contenant 2 lignes (sur la première, votre nom, sur la seconde, le mot choisi pour votre projet)
B. Ecrire un script bash (que vous mettrez dans le dossier PROGRAMMES) permettant de générer dans le répertoire TABLEAUX un fichier html contenant 1 tableaux avec 2 lignes (sur la première, votre nom, sur la seconde, le mot choisi pour votre projet)

L'ensemble des activités de cette séance devra être intégré sur votre blog de projet…

Prochaine étape : début du projet avec vos fichiers d'URLs !

Table 3 – *Commandes* `Shell` *liées à l'arborescence.*

Nom	Commande	Effet
List Segment*	`ls`	Affiche le contenu du répertoire courant.
	`ls PATTERN`	Affiche les fichiers qui correspondent au motif PATTERN.
	`ls -l`	Affiche les fichiers sous forme de liste détaillée
	`ls -a`	Affiche les fichiers cachés.
Path Working Directory*	`pwd`	Affiche le chemin absolu jusqu'au répertoire courant.
Change Directory*	`cd DIR`	Saute jusqu'au dossier DIR.
MaKe DIRectory*	`mkdir DIR`	Crée le répertoire DIR.
	`mkdir -p DIR`	Crée le dossier DIR ainsi que tous les dossiers parents nécessaires (s'ils n'existent pas).
ReMove DIRectory*	`rmdir DIR`	Supprime le répertoire DIR (s'il est vide).
conCATenate*	`cat FILE`	Affiche le contenu du fichier FILE
	`cat FILE1 FILE2`	Affiche le contenu du fichier FILE1 immédiatement suivi du contenu de FILE2
FILE	`file FILE`	Affiche le type du fichier FILE.
TOUCH	`touch FILE`	Crée un fichier vide appelé FILE.
MoVe*	`mv FILE1 DIR`	Déplace le fichier FILE1 vers DIR
	`mv FILE1 FILE2`	Déplace et renomme le fichier FILE1 en FILE2.
CoPy*	`cp FILE1 DIR`	copie le fichier FILE1 vers DIR
	`cp FILE1 FILE2`	Copie et renomme le fichier FILE1 en FILE2.
ReMove*	`rm FILE1`	Supprime le fichier FILE1.
	`rm PATTERN`	DANGEREUX ! Supprime tous les fichiers correspondant au motif PATTERN.
	`rm -r DIR`	DANGEREUX ! Supprime le répertoire DIR ainsi que tout son contenu (fichiers et sous-répertoires).

Table 2 – *Commandes `Shell` liées à la manipulation de flux.*

Nom	Commande	Effet
MORE*	`CMD | more`	L'affichage se restreint à la taille de votre terminal, et une action de votre part est attendue pour afficher la suite (tampon). Nous ne pouvons pas remonter.
LESS	`CMD | less`	Comme more, mais nous pouvons remonter dans l'affichage (pagination).
HEAD	`CMD | head -n N`	N'affiche que les N premières lignes du flux d'informations.
TAIL	`CMD | tail -n N`	N'affiche que les N dernières lignes du flux d'informations.
GREP*	`CMD | grep PATTERN`	Filtre le flux d'information et ne garde que les lignes qui contiennent PATTERN.
	`grep PATTERN FILE`	Affiche les lignes du fichier qui contiennent PATTERN.
SORT	`CMD | sort`	Trie les lignes du flux d'informations. (man sort pour plus d'info.)
	`sort FILE1 FILE2`	Affiche les lignes triées des fichiers FILE1 FILE2. (man sort pour plus d'info.)
Word Count	`CMD | wc -l`	Renvoie le nombre de lignes du flux d'informations.
	`CMD | wc -w`	Renvoie le nombre de mots du flux d'informations.
	`CMD | wc -c`	Renvoie le nombre de caractères du flux d'informations.
CUT	`CMD | cut -d DEL -f N`	Pour chaque lignes du flux d'informations, renvoie le Nième fragment délimité par DEL.

Table 2 – *Commandes Shell liées à la manipulation de flux.*

Nom	Commande	Effet
MORE*	`CMD | more`	L'affichage se restreint à la taille de votre terminal, et une action de votre part est attendue pour afficher la suite (tampon). Nous ne pouvons pas remonter.
LESS	`CMD | less`	Comme more, mais nous pouvons remonter dans l'affichage (pagination).
HEAD	`CMD | head -n N`	N'affiche que les N premières lignes du flux d'informations.
TAIL	`CMD | tail -n N`	N'affiche que les N dernières lignes du flux d'informations.
GREP*	`CMD | grep PATTERN`	Filtre le flux d'information et ne garde que les lignes qui contiennent PATTERN.
	`grep PATTERN FILE`	Affiche les lignes du fichier qui contiennent PATTERN.
SORT	`CMD | sort`	Trie les lignes du flux d'informations. (man sort pour plus d'info.)
	`sort FILE1 FILE2`	Affiche les lignes triées des fichiers FILE1 FILE2. (man sort pour plus d'info.)
Word Count	`CMD | wc -l`	Renvoie le nombre de lignes du flux d'informations.
	`CMD | wc -w`	Renvoie le nombre de mots du flux d'informations.
	`CMD | wc -c`	Renvoie le nombre de caractères du flux d'informations.
CUT	`CMD | cut -d DEL -f N`	Pour chaque lignes du flux d'informations, renvoie le Nième fragment délimité par DEL.