INFO 301 – **2005/2006**

Pratique du C: Fiche de TD 1

Interpréteur de commandes

septembre 2005

1 Introduction

Cette note commence par expliciter quelques fondamentaux avant d'aborder des notions relatives aux interpréteurs de commandes.

1.1 Structuration du système de fichiers : point de vue utilisateur

Un fichier est l'abstraction d'un flux linéaire d'octets.

Aucune information sur l'organisation de l'espace du support physique le stockant ne nous est utile à ce niveau d'abstraction. Pour manipuler les fichiers, il faut juste pouvoir les identifier par leurs caractéristiques :

- nom, type, taille du fichier ;
- propriétaire du fichier (cf. sa représentation en section 1.2) ;
- date de création, date de dernière modification ;
- protection : qui a droit de le lire et de le manipuler ;

sans s'occuper de l'implantation de ces dernières.

1.1.1 Droits d'accès et masque binaire

Dans les systèmes d'exploitation dérivés d'Unix, le codage des droits se fait sur 9 bits groupés par 3 bits qui sont dans l'ordre :

r le fichier est lisible par le propriétaire (- dans le cas contraire) ;

w le fichier est modifiable par le propriétaire (- sinon) ;

x le fichier est exécutable par le propriétaire (- sinon).

Le groupe suivant de 3 bits reprend le même principe mais définit les droits pour les membres du groupe auquel appartient le propriétaire et le dernier groupe concerne les autres utilisateurs suivant le même schéma.

Ces droits sont codés en un entier. Pour ce faire, on convient de la correspondance : $r = 4, w = 2$ et $x = 1$. Ainsi, les droits rw-, correspondent à l'entier (r=)4+(w=)2=6. Donc rw-rw-rw- correspond à l'entier 666.

Questions.

1. À quels droits correspondent les entiers $751, 521, 214$ et 150 ?
2. Par quels entiers sont codés les droits rw-r--r-- et rwxr-xr-x ?

Droits par défaut. Par ailleurs, à la création d'un fichier, des droits d'accès par défaut sont donnés à ce fichier. En standard, l'entier codant ces droits est 666 pour un fichier et 777 pour un répertoire.

La commande shell interne[1] `umask` permet de définir la protection maximale que l'on donne au cours de la session aux nouveaux fichiers ; sa syntaxe est : `umask <octal>`.

[1] Cette notion est définie dans la suite.

Les droits d'accès sont déterminés après l'opération logique suivante : defaut ET (NON octal). Plus simplement et lorsque c'est possible, on part des droits par défaut et on retranche le masque octal. Si ce n'est pas possible, on applique la règle précédente.

Pour connaître la valeur du masque courant, il suffit d'utiliser la commande umask sans arguments.

Questions.

1. A quels droits correspondent les masquages 77 et 244 dans le cas d'un fichier et dans celui d'un répertoire ?
2. Quel masque faudrait-il appliquer afin d'obtenir pendant la session les droits rw-rw-r-- à chaque création de fichier dans le cas d'un fichier et dans celui d'un répertoire ?

1.1.2 Organisation en graphe

Cette organisation des fichiers est basée sur un type de fichier — les répertoires — codant les arêtes constituant le graphe. Les répertoires étant des fichiers, ils ont les mêmes attributs (droits, etc).

On obtient ainsi un graphe qui permet :

- de remonter l'arborescence depuis n'importe quel répertoire sans reparcourir systématiquement depuis la racine ;
- de créer des liens dans le répertoire `sedoglav` à un fichier référencé dans un autre répertoire (lien symbolique codé par un fichier).

Ce type de représentation de la communauté des fichiers est assoupli en autorisant plusieurs arêtes à pointer sur un même élément et en faisant pointer chaque répertoire sur son prédécesseur.

1.2 Représentation de l'utilisateur par le système

Tout utilisateur — considéré comme une entité connue par le système d'exploitation — est caractérisé par

- son *login* i.e. le nom d'utilisateur ;
- son mot de passe ;
- un unique numéro d'identification (uid) ;
- un numéro de groupe d'utilisateur (guid) auquel il appartient ;
- un répertoire i.e. son espace disque ($HOME) dans l'arborescence du système de fichiers ;
- le nom d'un programme d'interface entre l'utilisateur et le système.

Par exemple pour les utilisateurs locaux (hors réseaux), on trouve les informations dans le fichier `/etc/passwd`. L'information relative au superutilisateur dans ce fichier est :

```
root:x:0:0:root:/root:/bin/bash
```

Question 1.1 : *Expliciter les informations contenues dans les lignes suivantes :*

```
manu:x:500:500:manu:/home_local/manu:/bin/csh
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

extraites d'un fichier `/etc/passwd`.

2 Interpréteurs de commandes : les shells

Un *shell* est un interpréteur de commandes qui sert d'interface entre l'utilisateur et le système d'exploitation.

Le shell peut être utilisé suivant 2 modes :

1. En mode interactif, il permet d'exécuter des programmes tout en offrant des *opérateurs* destinés à combiner leurs actions.
2. En mode *batch*, il offre un langage de programmation : les instructions sont définies dans un *script* que le shell interprète (pas de compilation).

Il existe plusieurs interpréteurs de commandes qui sont :
– dérivés du Bourne shell (AT&T) comme sh, ksh, bash, etc. ;
– dérivés du C shell (BSD) comme csh, tcsh, etc.

Le shell présente une *invite de commande* (disons %) qui permet de saisir une commande du type :

```
% <commande> [option(s) de la commande] [argument(s) de la commande]
```

Les commandes shell sont de 2 types : interne et externe.

2.1 Commandes externes

Ce type de commande provoque l'exécution d'un processus à partir d'un fichier exécutable éponyme de la commande et se trouvant dans l'arborescence du système de fichiers (dans le répertoire `/usr/bin` par exemple).

L'outil fondamental est le manuel d'utilisation `man` et la première chose à faire est de lire l'aide sur le manuel en utilisant la commande `% man man` dans votre interpréteur de commandes favori.

`% man -a mount` affiche l'ensemble des pages d'aide contenant le mot mount. Entre autre :

```
mount                (2)  - mount and unmount filesystems
mount                (8)  - mount a file system
```

`% man -S8 mount` affiche l'aide sur `mount` issue de la section 8 du manuel.

On peut aussi utiliser l'utilitaire `info` mais, bien que plus évolué (liens hypertext), il n'est pas forcement complet.

Ceci fait les commandes shell n'auront plus de secrets pour vous :

2.2 Les tâches de fond et le contrôle des processus

Par défaut, les shells attendent la fin de l'exécution d'une commande avant de permettre la saisie et l'exécution d'une autre.

Pour détruire une application dont le shell attend la terminaison, on utilise le raccourci clavier CTRL-C. Pour interrompre sans détruire une application, on utilise le raccourci clavier CTRL-Z.

Les shells permettent aussi de lancer une application en *tâche de fond* et ainsi l'exécution d'une autre (même si la première n'est pas terminée). Pour ce faire, on termine la commande par &.

Les commandes externe ps et kill Un numéro identifiant est associée à chaque processus. La commande externe `ps` permet d'afficher les informations associées aux processus.

```
% ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  6013  2434  2426  0  75   0 -   954 rt_sig pts/1    00:00:00 csh
0 S  6013  2688  2434  0  69   0 -  3523 select pts/1    00:00:04 xemacs
0 R  6013  3061  2434  0  76   0 -   895 -      pts/1    00:00:00 ps
```

Les processus peuvent recevoir un signal envoyé par la commande externe `kill -<Signal> <PID>`. Les principaux signaux sont :

Signal	Signification
15	terminaison de processus
9	destruction inconditionnelle de processus (CTRL-C)
19	suspension de processus (CTRL-Z)
18	reprise d'exécution d'un processus suspendu

2.2.1 Expressions régulières et méta-caractères

Il s'agit d'expressions décrivant des règles des propriétés de chaînes de caractères. Pour ce faire, on utilise en shell les *métacaractères* :

- le point d'interrogation `?` correspond à n'importe quel caractère (sauf EOL). L'expression régulière `b?l` représente les chaînes *bal* et *bol* et toutes les autres combinaisons comme *bwl* ;
- la paire de crochet `[ ]` permet de spécifier plus restrictivement un ensemble de caractères. L'expression régulière `dupon[dt]` ne représente que les chaînes *dupond* et *dupont*. L'expression régulière `dupon[d-t]` représente toutes les chaînes commençant par *dupon* et se terminant par une lettre comprise entre *d* et *t*. L'expression régulière `dupon[^dt]` représente toutes les chaînes commençant par *dupon* et ne se terminant ni par un *d* ni par un *t* ;
- l'étoile `*` désigne 0, 1 ou plusieurs caractères quelconques. L'expression régulière `*` représente toutes les chaînes.

Le préfixe \ (antislash) transforme un métacaractère en caractère.

Question 2.1 : *Utilisez la commande externe* `ls` *pour réaliser l'exercice suivant.*

1. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m`*.*
2. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *et comporte exactement 3 caractères.*
3. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *et comporte au moins 3 caractères.*
4. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *et comporte une extension (suffixe suivant un* `.`*) non vide.*
5. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *ou* `j`*.*
6. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *et comporte une lettre majuscule.*
7. *Lister les noms de fichiers comportant une étoile* `*`
8. *Lister les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *et ne comporte pas de lettre majuscule.*

Il n'est pas possible de traiter la dernière question avec la commande `ls` et les méta-caractères. Il faudrait donc soit pouvoir utiliser une autre commande, soit prendre le résultat de `ls` pour finir de le traiter avec une autre commande.

2.3 Commandes internes et variables

Toutes les commandes d'un shell ne sont pas externe. Par exemple, la commande `cd` qui permet de se déplacer dans l'arborescence du système de fichier n'est pas associée à un fichier exécutable du même nom.

Autre exemple, la commande `setenv` ne correspond à aucun fichier exécutable mais est implantée dans le code du shell. Cette commande est interne et permet d'afficher et de modifier des variables internes au shell :

```
% setenv
USER=sedoglav
LOGNAME=sedoglav
HOME=/home/enseign/sedoglav
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr1/bin:/usr/X11R6/bin
MAIL=/var/mail/sedoglav
SHELL=/bin/csh
HOSTTYPE=i586-linux
PWD=/home/enseign/sedoglav
GROUP=enseign
LANG=fr_FR
```

```
SYSFONT=lat0-16
TMP=/home/enseign/sedoglav/tmp
HOSTNAME=lxt2
```

Définition et affectation de variables : une variable est définie dès qu'elle est affectée. En sh, `FOO="Bonjour le monde"`. En csh, on utilise `% set FOO="Bonjour le monde"`

La commande `echo`[2] permet d'afficher l'argument qui lui est fourni :

```
% echo FOO
FOO
```

Pour évaluer une variable, il faut préfixer son nom par $.

```
% echo $FOO
Bonjour le monde
```

En sh, la commande interne `export` étend la porté d'une variable : par défaut, cette dernière n'est connue que par le processus courant ; après coup, cette variable est connue par tous les processus fils. En csh, on utilise : `% setenv FOO "Bonjour le monde"`.

Quelques variables d'environnement. Les variables définies dans les fichiers `/etc/profile` et `~/profile` sont créées lors de l'ouverture d'une session.

$PATH	les répertoires dans lesquels sont cherchés les exécutables des commandes externes
$HOME	votre répertoire de travail
$TERM	le type de terminal
$PWD	le répertoire courant
$DISPLAY	cette variable est utilisé par l'interface graphique pour savoir où se fait l'affichage
$PS1	l'invite de commande

Ces variables d'environnement peuvent être utilisées depuis un programme C (fonction getenv) lancé depuis le shell.

Variables prédéfinies du Bourne shell

- `$#` : nombre d'arguments
- `$*`, `$@` : tous les arguments de la commande
- `$0` : le nom de la commande
- `$1` à `$9` : les 9 premiers arguments de la commandes.
  La commande interne `shift` permet la renumérotation des arguments (`$1` est perdu) et donc l'accès aux arguments ≥ 10 ; la variable `#` est mis à jour.
- `$$` : numéro du processus courant
- `$?` : toutes les commandes ont un code de retour — codé sur un octet — (*exit-status*) i.e. une valeur entière qui fournie une information sur le déroulement de la commande.
  - déroulement normal $\Rightarrow$ `$?` $= 0$
  - déroulement anormal $\Rightarrow$ `$?` $\neq 0$
- nous verrons comment renvoyer en C le code de retour.

Typage. Dans un shell, **tout n'est que chaîne de caractères**.

Chaque commande est une chaîne que le shell évalue. On peut influer sur cette évaluation grâce aux délimiteurs suivants :

- les quotes `' '` bloquent l'évaluation ;
- les guillemets `" "` forment une chaîne après évaluation des composantes ;

[2] externe sur mon système mais qui peut être interne sur d'autres.

– les backquotes ‘ ‘ forment une chaîne évaluée comme une commande.

```
% echo '$FOO'
$FOO
% echo "echo '$FOO'"
echo 'Bonjour le monde'
% set BAR="n\'importe quoi"
% echo $BAR
n\'importe quoi
% set BAR=‘n\'importe quoi‘
n'importe: Command not found.
```

Conséquence sur la manipulation d'entiers. Pour utiliser l'arithmétique de base, il faut évaluer des chaînes de caractères codant des expressions arithmétiques grâce à la commande externe `expr` :

```
% set i=12;set i=‘expr $i + 1‘
% echo $i $?
13 0
% expr 2 \* 2
4
```

Le code de retour de la commande `expr` :
0 si le résultat est différent de 0 ;
1 si le résultat est égal à 0 ;
2 si un argument est non numérique.

2.4 Les opérateurs de composition de commandes dans un shell

Le point virgule ; permet de séparer des commandes.

L'esperluette & permet de lancer une commande en tâche de fond.

Une ligne de commande entre parenthèse est exécutée dans un autre shell lancé par le shell courant.

Le shell dispose de 2 opérateurs conditionnels :
`cmd1 && cmd2` : cmd2 est exécuter ssi cmd1 retourne 0 ;
`cmd1 || cmd2` : cmd2 est exécuter ssi cmd1 retourne un code non nul.

Redirection de flux. Par défaut, chaque processus possède 3 fichiers d'entrée-sortie :
stdin 0 est l'entrée standard (par défaut, le clavier) :
stdout 1 est la sortie standard (par défaut, l'écran) :
stderr 2 est la sortie des erreurs (par défaut, l'écran).

Ces flux peuvent être redirigés :
< entrée standard à partir d'un fichier ;
> sortie standard dans un fichier (création ou écrasement) ;
>> sortie standard dans un fichier (création ou ajout) ;
`<<EOF texte EOF` insertion de texte dans l'entrée standard ;
| tube de communication ;
2 > sortie des erreurs dans un fichier (bash uniquement) ;
>& sortie standard et erreur dans un fichier.

Question 2.2 : *Sachant que la commande* `sleep` *décompte un temps donné en argument sans rien faire, tentez de déterminez à l'avance le comportement des instructions suivantes :*

```
sleep 5 ; echo A
echo A ; sleep 5
sleep 5 & echo A
echo A & sleep 5
(echo A ; sleep 5) &
echo A ; sleep 5 ; echo B
echo A ; sleep 5 & echo B
(echo A ; sleep 5 ) & echo B
echo A ; (sleep 5 & echo B)
echo A ; (sleep 5 ; echo B) &
sleep 5 & echo A ; ( sleep 5 ; echo B)
sleep 5 & echo A & ( sleep 5 ; echo B)
sleep 5 & echo A & ( sleep 5 & echo B) &
sleep 5 & echo A & ( sleep 5 ; echo B) &
```

Question 2.3 :*Le nom de la commande externe* `grep` *vient de "global regular expressions parsing" ("analyse d'expressions régulières globales"). Elle permet de rechercher une expression régulière dans chaque ligne d'un ou de plusieurs fichiers passés en paramètres (ou dans chaque ligne de l'entrée standard) et affiche ces lignes sur la sortie standard.*

Que devrait retourner la ligne de commande suivante :

```
% grep a << separateur
> Alexandre
> Virginie
> separateur
```

Indication. *La commande grep reçoit le texte situé entre les deux occurrences du séparateur.*

2.4.1 La notion de filtre

Une ligne de commande qui peut recevoir des données par l'intermédiaire de son entrée standard et fournir des résultats à sa sortie standard est appelée un filtre.

Les filtres présentent le gros avantage de pouvoir être "branchés" les uns à la suite des autres : en redirigeant la sortie standard d'un filtre sur l'entrée standard d'un autre, on peut réaliser un nouveau filtre.

Question 2.4 :*[Redirection] Dans cet exercice, nous allons utiliser les redirections couplées à d'autres commandes :* `grep`, `cut`, `tr` *et* `sort`.

1. *Listez les entrées du répertoire* `/usr/bin` *dont le nom commence par la lettre* `m` *et comporte au moins une lettre majuscule ! (Avec la commande* `grep`*).*
2. *En vous servant de la question précédente, réalisez la dernière question de l'exercice 1.*

Question 2.5 : *La commande* `ls -l` *permet d'afficher l'ensemble des informations sur les fichiers d'un répertoire.*

```
total 104K
-rw-r--r--  1 dumont  west  234 Sep 28 11:33 td1.aux
-rw-r--r--  1 dumont  west 6.2K Sep 28 11:33 td1.dvi
-rw-r--r--  1 dumont  west 7.1K Sep 28 11:33 td1.log
-rw-r--r--  1 dumont  west 3.7K Sep 28 12:01 td1.tex
```

1. *Listez uniquement les propriétés des fichiers se trouvant dans un répertoire (on se servira de l'exemple ci-dessus).*
2. *Même question mais en listant cette fois-ci, le nom du propriétaire, le mois de la dernière modification et le nom de ces fichiers. Attention il y a un piège !*
3. *Sauvegardez le resultat de la dernière commande dans un fichier. Triez suivant le nom des propriétaires et le nom du fichier (dans l'ordre alphabéetique) et sauvegardez le résultat à nouveau dans ce même fichier. Recommencez en vous passant de fichier auxilliaire.*

3 Annexe

Les descriptions des commandes dans cette annexe sont sommaires. Utilisez le manuel en ligne pour avoir une description complète.

cut Cette commande permet de supprimer une partie des lignes d'un ou de plusieurs fichiers, ou de l'entrée standard. Sa syntaxe est :

```
% cut -cliste [fichier ...]
% cut -fliste [-ddelimiteur] [fichier ...]
```

L'option

-c ne retient sur chaque ligne que les caractères situés à une position donnée par liste. Par exemple, l'option -c1-50 ne garde que les 50 premiers caractères de chaque ligne.

-f ne retient qu'une liste de champs (séparés soit par des tabulations, soit par un délimiteur spécifié par l'option -d). Par exemple, l'option -f1,3-5 conduira la commande à ne garder que les champs numéros 1, 3, 4 et 5 de chaque ligne.

tr Cette commande permet la "traduction" de caractères provenant de l'entrée standard. Sa syntaxe est :

```
% tr -s|-d string
```

L'option

-s élimine les répétitions des caractères composant la chaîne de caractères sring, en n'en laissant qu'un à chaque fois.

-d supprime toutes les occurrences des caractères composant la chaîne de caractères string.

sort Cette commande lit un ou plusieurs fichiers ou l'entrée standard, et affiche les différentes lignes, triées sur un ou plusieurs champs selon l'ordre lexicographique. Les séparateurs de deux champs sont l'espace et la tabulation. Sa syntaxe est :

```
% sort -k clef -u
```

L'option

-k permet d'indiquer les champs à prendre en compte pour le tri. L'argument clef doit être écrit sous la forme premier_champ[,dernier_champ] la partie entre crochets étant optionnelle. Si cette partie est absente, cela signifie que le tri prend en compte tous les champs á partir du champ de numéro premier_champ.

-u permet l'affichage d'un seul exemplaire des lignes identiques.