USTL

Licence d'informatique
Module de C/Unix

TD : Premiers programmes en shell

Exercice 1 (Méta-caractères)

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 entrées du répertoire `/usr/bin` dont le nom commence par la lettre `m` et ne comporte pas de lettre majuscule.
8. Lister les noms de fichiers comportant une étoile `*`

Exercice 2 (Compter...)
Écrire un programme Shell qui compte de 1 à 10. Deux versions sont imaginables ; une version à base d'une boucle `for` et une version à partir d'une boucle `while`.

Exercice 3 (Arguments)
Écrire un programme Shell qui affiche un à un les arguments de la commande. Par exemple :

```
% argument un deux trois
3 arguments :
  1 - un
  2 - deux
  3 - trois
```

Exercice 4 (Suite)
Écrire un programme qui sort les valeurs successives de la suite définie par :

$$U_{n+1} = \begin{array}{ll} U_n/2 & \text{si } U_n \text{ est pair,} \\ 3 \times U_n + 1 & \text{si } U_n \text{ est impair.} \end{array}$$

La valeur de U_0 est passée en argument de la commande. L'exécution se termine à l'affichage de la valeur 1.

Exercice 5 (Poubelle)
Question 5.1 On désire écrire une commande de manipulation d'une poubelle de fichiers, `can`. Cette commande accepte les trois options suivantes :

`can -l` liste la poubelle ;

`can -r` purge la poubelle ;

`can` *`filenames`* transfert les fichiers *`filenames`* dans la poubelle.

Question 5.2 Ajouter l'option

`can -x` *`filenames`* qui supprime les fichiers *`filenames`* sans passer par la poubelle.

Id : tdishell.tex,v 1.4 2000/11/28 18 :34 :55 marquet Exp

Exercice 6 (Recherche d'utilisateurs)
Le fichier `/etc/passwd` contient les informations concernant les utilisateurs. En particulier sont mémorisés le nom de login et les noms et prénoms des utilisateurs. On développera une commande `searchuser` qui identifie un utilisateur à partir de son nom, prénom ou nom de login. Par exemple :

```
% searchuser marquet phm toto
marquet est connu :
phm:f4b0FtWrtrN3Y:5408:54:Philippe Marquet:/home/phm:/bin/tcsh
phm est connu :
phm:f4b0FtWrtrN3Y:5408:54:Philippe Marquet:/home/phm:/bin/tcsh
toto n'est pas connu
```

On utiliser la commande Unix :

`grep` qui filtre une entrée (l'entrée standard ou la liste de fichiers passée en paramètre) et affiche toutes les lignes contenant le premiers argument de la commande. De plus, `grep` retourne 0 si la recherche est positive, 1 sinon. L'option `-q` de `grep` supprime l'affichage du résultat ; seul le code de retour de la commande est alors exploitable.

Exercice 7
Écrire une commande `croissante` qui vérifie qu'une suite d'entiers donnée en paramètre est croissante.

Exercice 8 (Mon `cp`)
Réécrire la commande Unix `cp` à l'aide de l'unique primitive

`copy` *`src dest`*

qui recopie le contenu du fichier source *`src`* (qui doit exister) dans le fichier *`dest`* (qui ne doit pas exister).

Exercice 9 (Kind)
Écrire une commande `kind` qui liste les fichiers d'un type donné. La syntaxe de cette commande est

`kind` [`-d`|`-t`|`-x`] *`filenames`*

La commande liste les fichiers qui sont des fichiers de texte (par défaut ou option `-t`) qui sont des répertoires (directories, option `-d`) ; ou qui sont exécutables (option `-x`).
On utilisera la commande Unix :

`file` qui retourne le type d'un fichier ou d'une liste de fichiers passés en argument. L'information est écrite sous forme d'une chaîne de caractères sur la sortie standard. Par exemple :

```
% file *
memo.txt:       ASCII text
kind.sh:        Bourne shell script text
slides:         directory
clean:          ELF 32-bit LSB executable, dynamically linked
```

Exercice 10 (Dernier login)
Écrire une commande Shell qui gère le dernier login. Lors de chaque login, on appelle la commande `lastlog -l` ; l'appel `lastlog` affiche la date du dernier login (mais pas celui en cours !).

Exercice 11 (Où est-il ?)
Le but est d'écrire une commande Shell qui recherche les fichiers dont le nom est passé en argument dans les chemins de recherche des commandes. Ces chemins sont codés dans la variable `$PATH`.
On pourra développer les questions suivantes :

Question 11.1 (La commande `subst`) La variable `$PATH` contient une suite de noms de répertoires séparés par le caractère `:`. Par exemple :

```
% echo $PATH
.:/opt/kde/bin:/usr/local/lib/texmf/bin/i86_linux:/usr/local/bin:/usr/sbin:\
:/usr/X11R6/bin:/sbin:/etc:/home/phm/bin/linux:/home/phm/bin
```

Pour la transformer en une liste de répertoire plus facilement manipulable par le Shell, il faut remplacer chacun des caractères `:` par un espace. Ceci fait l'objet de la commande `subst`.

Question 11.2 (La commande `whereisinpath`) La commande `whereisinpath` accepte un argument et recherche dans les répertoires donnés par la variable `$PATH` si une entrée correspondante à l'argument existe. Elle doit imprimer toutes les possibilités existantes.

Question 11.3 (La commande isbuiltin) Un certain nombre de commandes du shell ne correspondent pas à des exécutables recherchés dans les répertoires de la variable `$PATH` mais sont des commandes internes (*built-in command*) du shell et sont directement exécutées par le shell.

La commande `isbuiltin` teste si son argument correspond à une telle commande. On suppose que le fichier désigné par la variable `$BUILTIN` contient la liste des commandes internes du shell.

Question 11.4 (La commande `whereareinpath`) Cette commande réalise le même traitement que `whereisinpath` pour chacun des arguments qui lui sont donnés. Cependant elle vérifie préalablement si la commande correspond à une commande interne du shell. L'option `-u` limite l'affichage, pour chacun des arguments à la première solution.

Exercice 12 (Vérification du chemin)
Lorsqu'une commande prend en argument un chemin est que celui-ci n'existe pas, un message d'erreur est affiché, par exemple :

```
% ls /usr/lib/thissoft/config/Makefile
/usr/lib/thissoft/config/Makefile: No such file or directory.
```

La commande `verif` affichera la partie gauche du chemin qui est valide et la première entrée inexistante :

```
% verif /usr/lib/thissoft/config/Makefile
/usr/lib/thissoft: No such directory.
```

Exercice 13 (Instructions pushd/popd)
Le C-shell fournit deux commandes internes `pushd` et `popd` facilitant la gestion des changemenst de répertoire à l'aide d'une pile. La commande `pushd` est équivalente à la commande `cd` mais elle mémorise (empile) le répertoire courant. La commande `popd` change de répertoire pour le répertoire qui se trouve en sommet de pile et supprime ce répertoire de la pile.

On peut mémoriser la pile de répertoire dans un fichier.

Exercice 14 (Gestionnaire de mail)
Écrire une programme qui regarde si il y a du courrier pour un utilisateur dont le nom est passé en argument ; par défaut l'utilisateur courant. Le mail à destination de `$USER` est déposé par le système de gestion des mails dans le fichier `/var/spool/mail/$USER`. On affichera un des messages suivants :

```
pas de messages pour <user>
courrier du jour pour <user>
courrier en retard pour <user>
```

Exercice 15
Développer une commande `numerique` qui vérifie que son argument est un entier.

Exercice 16
Développer la commande

`extract` [*filename*] *`lineno1`* *`lineno2`*

qui affiche les lignes *`lineno1`* à *`lineno2`* du fichier passé en paramètre (par défaut, l'entrée standard).

Exercice 17
Développer la commande `tree` qui affiche l'arborescence dont la racine est passée en paramètre.