Initiation aux systèmes d'exploitation (IS1)
TP n° 10 : interprétation de la ligne de commande

Le but de ce TP est de mieux comprendre les différentes opérations effectuées par le shell pour interpréter une ligne de commande *avant de* l'exécuter. Pour rappel, une ligne de commande « simple » est toujours constituée d'un premier « mot » – la commande – puis éventuellement d'autres – les options et les paramètres :

$$\begin{array}{cccccccc} \text{cmd} & \text{opt}_1 & \dots & \text{opt}_n & \text{par}_1 & \dots & \text{par}_m \\ \downarrow & \downarrow & & \downarrow & \downarrow & & \downarrow \\ \text{mot}_0 & \text{mot}_1 & \dots & \text{mot}_n & \text{mot}_{n+1} & \dots & \text{mot}_{n+m} \end{array}$$

Effectuer ce découpage, puis interpréter les différents mots, et notamment la commande, utilise d'une part les différents mécanismes regroupés sous le nom d'*expansion*, et d'autre part les variables du shell.

Expansion

Quand un processus shell reçoit une ligne de commande, il pré-traite cette chaîne de caractères avant de l'exécuter. En particulier, il découpe cette chaîne de caractères selon le séparateur « espace »[1] ; ainsi la ligne :

`cat fic1 fic2 fic3`

est découpée en une liste de quatre chaînes de caractères :

`cat` `fic1` `fic2` `fic3` .

Par ailleurs, certains caractères spéciaux permettent de demander au shell de faire des pré-traitements sur la chaîne de caractères avant de la séparer en arguments et de l'exécuter ; on dit que le shell procède à l'*expansion* de cette chaîne de caractères. Nous avons déjà vu que le shell remplace les mots contenant des caractères joker (`*`, `?`, `[`, `]`) par la liste des noms de fichiers correspondant au motif décrit. Le tableau ci-dessous récapitule les autres cas d'expansion les plus fréquents :

Syntaxe	Exemple	Résultat	Interprétation
`{ }`	`bjr ba{li,lo}t sava`	`bjr balit balot sava`	énumération
	`ba{1..3}`	`ba1` `ba2` `ba3`	
`$...` ou `${...}`	`$HOME`, `${HOME}`	`/home/roger`	valeur de variable
`$(...)`, `` `...` ``	`$(tty)`, `` `tty` ``	`/dev/pts/0`	sortie d'une commande
`$((...))`	`$((12 + 4 * 2))`	`20`	évaluation numérique

⚠ Attention, dans le tableau ci-dessus, ` est un accent grave et non une apostrophe.

1. plus précisément, selon les paquets de caractères listés dans la variable `IFS` (pour *Input Field Separator*) : par défaut l'espace, la tabulation (`\t`) et la fin de ligne (`\n`). Il n'est pas facile d'afficher la valeur de cette variable... « `echo $IFS | od -c` » devrait convenir.

Instructions de rendu

À la fin de la séance, faites une archive `archive_tp10.tar` contenant votre fichier de réponses et les différents scripts demandés, puis déposez-la sur Moodle. (Il est expliqué dans les premiers TP comment créer une archive).

Exercice 1

✍ Écrire un script `expansion.sh` qui affiche dans le terminal les lignes suivantes ; utiliser le mécanisme d'expansion pour compléter les premières lignes en remplaçant les « . . . » par les valeurs correctes, et éviter de taper les deux dernière ligne en entier.

```
Le chemin absolu de mon répertoire courant est ...        (valeur déterminée par le shell)
Le résultat de la commande whoami est ...                 (résultat)
Le produit de 33 par 17 vaut ...                          (résultat)
Il a agi anticonstitutionnellement anticonventionnellement antisocialement.
Les caractères latins sont a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s,
t, u, v, w, x, y, z.
```

Exercice 2 (optionnel) – *manipulation de chaînes de caractères*

Définir une variable `MARY` ayant la valeur `supercalifragilisticexpialidocious`, puis afficher les expansions des expressions suivantes pour en déterminer le sens. Il est suggéré de mettre `echo` avant chaque expansion afin de ne pas avoir de message d'erreur.

1. `${#MARY}`
2. `${MARY:9}`
3. `${MARY:9:6}`
4. `${MARY#super}`
5. `${MARY#cali}`
6. `${MARY%docious}`
7. `${MARY/expiali/...}`
8. `${MARY/expialif/...}`

Échappement

De même que certains caractères indiquent au shell qu'il doit opérer une expansion, d'autres (tels que « & », « ! », « < », « > », « | » et « ; ») jouent un rôle spécial lors de l'interprétation des commandes. Le mécanisme dit *d'échappement* permet de *déspécialiser* ces caractères spéciaux pour les utiliser tels quels.

Syntaxe	Expression	Résultat	Explication
`\`	`\"\'\$\{ A\ B C`	`"'${` `A B` `C`	échappement d'un caractère
`"..."`	`" a $HOME $((1+1)) ' "`	`a /home/roger 2 '`	échappe tout sauf `$`, `\`, `!` et `"`
`'...'`	`' a $HOME $((1+1)) " '`	`a $HOME $((1+1)) "`	échappe tout sauf `\` et `'`

⚠ Attention, dans le tableau ci-dessus, ’ est une apostrophe et pas un accent grave.

Exercice 3

✍ Écrire un script `echappement.sh` qui affiche dans le terminal les lignes suivantes (exactement, y compris les espaces initiaux. Et pour la dernière ligne, vous pouvez essayer de factoriser par `hu`.) :

```
       A     B
L'ensemble { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 } est
inclus dans N.
Les singletons de cet ensemble sont {1}, {2}, {3}, {4}, {5}, {6}, {7}, {8},
{9}, {10}, {11}, {12}, {13}, {14}, {15}, {16}, {17}.
Elle demanda d'une voix *forte* : "Qui est-ce ?"
$HOME = ${HOME} = (valeur donnée par le shell).
19 * 216 = (valeur calculée par le shell)
Ajoutez un peu d'huile d'olive, d'humeur joyeuse, d'humus, et remuez bien !
```

Exercice 4

✍ Écrire un script `echappement_bis.sh` définissant une variable `TOTO` égale à `whoami`, puis affichant le texte ci-dessous en utilisant la variable `TOTO` ***pour chaque ligne*** (oui, même la dernière), mais jamais explicitement le mot `whoami`.

```
J'aime bien utiliser la commande "whoami".
Ma variable $TOTO contient 'whoami'.
Mon login est (login donné par le shell).
La commande d'aujourd'hui est \whoami\.
whoareyou, c'est une commande imaginaire !
```

La dernière ligne nécessite l'utilisation d'un mécanisme vu à l'exercice 2.

> **Trouver le bon chemin** La variable d'environnement `PATH` joue un rôle primordial : c'est elle qui contient la liste de tous les chemins vers les répertoires dans lesquels les fichiers exécutables doivent être recherchés. C'est grâce à elle qu'il est possible d'utiliser des noms de commandes simples tels que « `bash` », « `grep` » ou « `ls` » en lieu et place du chemin complet vers les exécutables concernés.
>
> Le caractère ':' sert de séparateur entre les différents chemins contenus dans la valeur de `PATH`.

Exercice 5 – *la variable d'environnement* `PATH`

1. Afficher la valeur de la variable d'environnement `PATH`.

2. Créer dans votre répertoire `[...]/TP10` un petit *script* nommé `fic.sh`, contenant la ligne suivante :

```
echo "ceci est le résultat de l'exécution du fichier fic.sh."
```

3. ✍ Après lui avoir donné les droits nécessaires, essayez de l'exécuter depuis votre répertoire `[...]/IS1` (en utilisant une référence valide, relative ou absolue). Recommencer depuis votre répertoire `[...]/TP10` par la commande « `./fic.sh` » puis par « `fic.sh` ». Que se passe-t-il dans chaque cas ? Pourquoi ?

4. ✍ Modifier `PATH` pour que la commande « `fic.sh` » s'exécute sans erreur depuis le répertoire `[...]/TP10` (et seulement depuis celui-ci).

5. ✍ Créer un répertoire `~/mybin`, puis déplacer `fic.sh` dans ce répertoire. Essayer ensuite de l'exécuter par la simple ligne de commande « `fic.sh` », depuis le répertoire `~/mybin`, puis depuis un autre répertoire. ✍ Comment faire en sorte que l'exécution s'effectue sans erreur dans tous les cas ?

Si vous le souhaitez, vous pouvez rendre pérenne(s) l'un et/ou l'autre de ces comportements en modifiant votre fichier `~/.bashrc`.

Exercice 6 (optionnel) – *autres variables d'environnement intéressantes*

1. D'autres variables s'utilisent de manière analogue à `PATH`, dans des contextes différents, par exemple `MANPATH`, `INFOPATH` ou `CDPATH`. Ce dernier permet de définir des répertoires par défaut dans lesquels chercher le nom passé en paramètre à « `cd` ».

   a. Créer dans votre répertoire personnel une arborescence contenant les répertoires `~/Je/Vais/Tous/Les/Jours/Ici` et `~/Je/Consulte/Sans/Arret/Celui/La`.

   b. Affecter la variable `CDPATH` de manière à pouvoir aller dans ces répertoires depuis n'importe quel répertoire par les commandes « `cd Ici` » et « `cd La` ». Vérifier que vous pouvez toujours aller d'un répertoire à un de ses fils de nom *Fils* par la commande « `cd Fils` »...

2. La variable `PS1` permet de personnaliser le prompt de `bash` : nature des informations, couleurs... Voir la section `PROMPTING` de « `man bash` ».

3. La variable `LS_COLORS` permet de choisir les couleurs utilisées par « `ls -G` » (ou « `ls --color` »). La syntaxe est cependant un peu barbare ; la commande « `dircolors` » permet d'obtenir, à partir d'un fichier relativement lisible, une affectation « `LS_COLORS=...` » au format exigé par `bash`.

   L'option « `-p` » permet de visualiser la configuration par défaut. Celle-ci peut être copiée dans un fichier `~/.dir_colors`, qui peut être modifié selon vos goûts puis passé en paramètre à « `dircolors` ».

   L'affectation « `LS_COLORS=...` » et la commande « `export LS_COLORS` » peuvent être exécutées à la main, ou via « `eval ‘dircolors‘` », ou être copiées dans le `.bashrc`.

Instructions de rendu

Faites une archive `archive_tp10.tar` contenant votre fichier de réponses et les différents scripts demandés, puis déposez-la sur Moodle. (Il est expliqué dans les premiers TP comment créer une archive).