Introduction aux systèmes d'exploitation (IS1)
TP n° 9 : 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 (puis 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. 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.

Les variables du shell

La personnalisation du comportement des différentes commandes utilise la définition de *variables d'environnement*. Nous avons déjà constaté l'impact de certaines d'entre elles :

- `PAGER` : indique quel exécutable est utilisé par « `man` » pour adapter l'affichage à la géométrie du terminal ;
- `LC_COLLATE` : définit l'ordre utilisé pour les caractères.

Nous allons en voir quelques autres, et notamment la variable `PATH` dont le rôle est primordial pour déterminer quelle commande doit être exécutée.

En bash, les variables possèdent un *identificateur*, qui peut être n'importe quelle suite de caractères commençant par une lettre ou '_' et ne contenant que des lettres, des chiffres ou '_'. Elles n'ont qu'un type possible, chaîne de caractères (mais l'évaluation numérique est possible lorsqu'elle a du sens).

La déclaration et l'affectation se font simultanément par :

`var=`*valeur* ***sans espace avant ni après le signe '='.***

On accède à la valeur d'une variable `var` par `$var` ou `${var}`.

« `env` » fournit la liste des variables d'environnement.

Exercice 1 – *quelques variables d'environnement*

1. Afficher la liste des variables d'environnement. Repérer `PAGER` et `LC_COLLATE`, mais aussi `SHELL`, `TERM`, `USER`, `PWD`...
2. ✍ Écrire une ligne de commande permettant d'obtenir l'affichage suivant :
   `Bonjour` *utilisateur*`, vous êtes actuellement dans le répertoire` *rep*.
   où *utilisateur* est le nom de l'utilisateur courant, et *rep* est un chemin vers le répertoire courant.

3. Que représente la variable `HOME` ? Modifier sa valeur, puis exécuter la commande « `cd` ». ✍ Que pouvez-vous en conclure ?

4. Afficher le contenu de la variable `OLDPWD`, puis exécuter la commande « `cd -` ». ✍ Que pouvez-vous en conclure ?

5. La commande « `locale -a` » indique toutes les configurations linguistiques disponibles. Modifier la valeur de la variable `LANG` puis observer comment cela affecte le comportement de commandes telles « `date` », « `cal` », « `man` »... ou encore les modes interactifs (« `rm -i` », par exemple).

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 2 – *la variable d'environnement* `PATH`

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

2. Créer dans votre répertoire `.../TP9` un petit *script* [1] nommé `fic.sh`, contenant les deux lignes suivantes :

```
#! /usr/bin/env bash
echo "ceci est le résultat de l'exécution du fichier fic.sh."
```

(note : le rôle de la première ligne est simplement de lever toute ambiguïté quant au programme susceptible d'interpréter la suite du script)

3. ✍ Après lui avoir donné les droits nécessaires, essayer de l'exécuter depuis votre répertoire `.../IS1` (en utilisant une référence valide, relative ou absolue). Recommencer depuis votre répertoire `.../TP9` 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 `.../TP9` (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`.

1. c'est-à-dire un programme écrit en bash, dans un fichier exécutable

Variables simples *vs.* variables d'environnement

Toutes les variables du shell ne sont pas des variables d'environnement – en particulier, définir une variable par « `var=valeur` » n'en fait pas une variable d'environnement. La différence majeure entre variable simple et variable d'environnement concerne leur *portée*.

« `set` » utilisée sans argument, affiche la liste de toutes les variables déclarées dans le shell courant.

Exercice 3 – *portée des variables par défaut*

1. Tout d'abord, exécuter la commande « `set -o posix` » pour obtenir le comportement normal de « `set` » (son comportement par défaut au `SCRIPT` ne respecte pas la norme POSIX, et l'affichage inclut, en plus des variables, toutes les *fonctions* définies ; l'affichage est à peu près illisible).

2. Dans le même terminal, afficher avec « `set` » toutes les variables définies dans le shell courant, et comparer avec la liste fournie par « `env` ».

3. ✍ Toujours dans ce terminal, déclarer une variable shell `NOM` contenant votre prénom. Vérifier que la variable `NOM` apparaît bien dans la liste obtenue par « `set` ».

4. Dans un autre terminal, afficher la valeur de la variable `NOM`. Que constatez-vous ?

5. ✍ Écrire un script `hello.sh` qui affiche `Bonjour,` suivi de la valeur de la variable `NOM`, puis modifie la valeur de la variable `NOM` à `Tartempion` et réitère son salut. Quel affichage obtient-on ? Quelle est maintenant la valeur de `NOM` dans le shell initial ?

« `export` » permet de transformer une variable en variable d'environnement.

6. ✍ Transformer `NOM` en variable d'environnement, puis réitérer les tests des questions 4 et 5.

7. ✍ Conclure quant à la portée des deux types de variables.

Exercice 4 (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 `~/Test/Je/Vais/Tous/Les/Jours/Ici` et `~/Test/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 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`.

Paramètres d'une commande

Comme vous le savez, la plupart des commandes acceptent des paramètres et/ou des options, qui lui sont transmis par le shell après interprétation et découpage de la ligne de commande. Lors de l'écriture d'un programme (en Java, C, bash, ou tout autre langage), il est possible de manipuler ces paramètres (qui seront transmis au programme lors de son exécution). En bash, cela se fait *via* certaines variables spéciales :

Les paramètres d'un script

- `$#` est le nombre de paramètres passés au script,
- `$0` est le nom du programme en cours d'exécution,
- pour chaque entier `i` entre 1 et `$#`, `$i` contient le i^e paramètre,
- `$@` contient la liste de tous les paramètres. Attention, en général il faut protéger cette variable avec des guillemets (`"$@"`) (à cause des espaces).

Exercice 5

✍ Écrire un script `bonjour.sh` qui affiche `Je dois saluer ... personnes` (en remplaçant `...` par la bonne valeur), puis sur une deuxième ligne : `Bonjour`, suivi de la liste de tous ses paramètres.

(Bien entendu, un traitement satisfaisant des paramètres n'est pas vraiment envisageable sans tests ni boucles ; de telles structures existent en bash, mais nous ne verrons pas cela durant cette séance)

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 »[2] pour affecter les variables représentant les différents arguments (`$0`, `$1`,..., la liste `$@` et `$#`) ; ainsi la ligne :

`cat fic1 fic2 fic3`

est découpée en une liste de quatre chaînes de caractères :

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

et donc `$#=3`, `$0=cat`, `$1=fic1`, `$2=fic2` et `$3=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**
`{ }`	`ba{ba,bu}`	`baba` `babu`	énumération
`$...` ou `${...}`	`$HOME`, `${HOME}`	`/home/roger`	valeur de variable
`$(...)`, `` `...` ``	`$(which true)`, `` `which true` ``	`/bin/true`	sortie d'une commande
`$((...))`	`$((12 + 4 * 2))`	`20`	évaluation numérique

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

Exercice 6

✍ 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 la 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.
```

Exercice 7 – *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.

2. 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.

1. `${#MARY}`
2. `${MARY:9}`
3. `${MARY:9:11}`
4. `${MARY#super}`
5. `${MARY%docious}`
6. `${MARY/expiali/...}`

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 8

✍ Écrire un script `echappement.sh` qui affiche dans le terminal les lignes suivantes :

```
       A     B
L'ensemble {1, 2, 3} est inclus dans N.
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 9

✍ 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***.

```
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 !
```

Instructions de rendu

Faites une archive `archive_tp9.tar` contenant votre fichier de réponses et les différents scripts demandés, puis déposez-la sur Moodle.