Introduction aux systèmes d'exploitation (IS1)

TP 7 – `bash` avancé

Expansions

Quand bash reçoit une ligne de commande, il fait certains pré-traitements avant de l 'exécuter. bash découpe normalement les lignes de commande aux caractères « espace ». Ainsi la ligne

```
cat fic1 fic2 fic3
```

est découpée en

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

Un certain nombre d'expressions utilisant des caractères spéciaux sont *expansés* en bash :

Syntaxe	Expression	Expansion	Explication
{ }	`ba{ba,bu}`	`baba` `babu`	énumération
\	`\"\'\$\{\ \A B`	`"'${ A` `B`	échappement
$	`$HOME`	`/home/roger`	valeur de variable
${...}	`${HOME}`	`/home/roger`	valeur de variable, voir aussi ex 2.2
$(...)	`$(which true)`	`/bin/true`	sortie d'une commande
`...`	`` `which true` ``	`/bin/true`	sortie d'une commande
$((...))	`$((12 + 4 * 2))`	`20`	valeur d'une expression
"..."	`" a $HOME $((1+1)) ' "`	`a /home/roger 2 '`	$ expansés
'...'	`' a $HOME $((1+1)) " '`	`a $HOME $((1+1)) "`	telle quelle

D'autres expressions ne prennent de signification que lorsque des fichiers portent les noms correspondants. Dans un répertoire contenant des fichiers nommés `abx`, `abd`, `abcde`, `adx` et `bdx`, les expansions suivantes ont lieu :

Caractères	Expression	Expansion
~	`~roger`	`/home/roger`
*	`ab*`	`abx` `abd` `abcde`
?	`ab?`	`abx` `abd`
[...]	`a[abcd]x`	`abx` `adx`

Les autres caractères tels que `!`, `&`, `<`, `>`, `|` et `;` qui jouent un rôle spécial lors de l'interprétation des commandes ont déjà été vu au cours des TP précédents.

Exercice 1 – *Premières escapades*

1. Placez-vous dans le répertoire `tp7` du compte `monitIS1` en utilisant un raccourci avec `~`.
2. Affichez le contenu du fichier `premier exercice`.
3. Utilisez la commande `echo` pour écrire les lignes suivantes en échappant avec \ tous les caractères qui poseraient problème :

```
A           B
L'ensemble {1, 2, 3} est inclus dans N
Elle s'écria : "Ciel mon mari !"
$HOME = votre répertoire maison
19 * 216 = le produit
```

bien sûr que bash doit afficher le vrai nom de votre répertoire maison et la valeur du produit.

Exercice 2 – *Expressions avant expansion*

1. Dans le répertoire `tp7` quelle *expression*[1] chois0.084781Tf71(i)2.848(s)2.517(o)-3.51(p)-1eT4.2808f48-3.803(n

pour tester lorsque le fichier `fic` existe (`-e`) mais (`-a` pour dire « et ») n'est pas exécutable (`-x` pour exécutable, `!` pour la négation, les parenthèses pour donner une sous-expression).

La commande `test` est si utile qu'il existe une notation abrégée plus élégante :

```
if [ -e fic -a \( ! -x fic \) ]; then echo Victoire; fi
```

Pour la même raison `test` fait partie des commandes qui sont directement interprétées par bash[2]. La syntaxe des opérations possibles est donc donnée dans la page de manuel de bash dans les explications de `test` mais aussi dans l'explication sur les expressions de conditions (« Conditional expressions »).

Exercice 3 – *Premier test*

1. Affichez une variable quelconque qui n'existe pas, par exemple `IMPROBABLE`.
2. Déduisez-en un test avec `if [`... permettant de savoir si une variable est définie ou pas. Suivant le cas, affichez la valeur de cette variable ou un message indiquant qu'elle aura désormais une valeur par défaut de votre choix.
3. Donnez la valeur `2 * 10` à votre variable. Comment faites-vous ? Est-ce que votre test fonctionne toujours ? Affichez la valeur de votre variable avec `echo` pour mieux comprendre ce qui se passe. Avez-vous une idée pour afficher directement la valeur de la variable, à savoir `2 * 10` ?

Exercice 4 – *Tests de comparaison*

1. Refaites l'exercice 5 du TP 4 qui se fixait pour objectif de comparer deux fichiers et d'afficher un message indiquant le résultat de la comparaison (identiques ou différents) avec un `if then else`.
2. En utilisant la primitive de test, améliorez pour que le message indique une erreur lorsqu'un des fichiers n'existe pas.
3. On veut désormais rediriger le résultat de ce test dans un fichier. Effectuez un test pour vérifier qu'il est possible d'écrire dans le fichier avant de le faire.

Exercice 5 – *Application : fichier de configuration*

Les commandes bash complexes sont

1. Comment afficher l'heure qu'il est sans la date ni les minutes.
2. Écrivez une commande qui salue l'utilisateur en fonction de l'heure qu' il est : "Bonjour" avant 18 heures et bonsoir après.
3. Ajoutez cette salutation personnalisée à votre fichier de configuration `.bashrc`. Testez le résultat en ouvrant des nouveaux terminaux.

Exercice 6 – *Opérons*

Les fichiers `q`*xy* contiennent des *questions* arithmétiques simples, les fichiers `r`*xy* contiennent normalement les *réponses*.

1. En utilisant la commande `read`, écrire sur une seule ligne la suite de commandes qui demandent à l'utilisateur un nom de fichier de question comme `q63` et affiche :

   ```
   3859 + 2773 = 6632
   ```

   si `q63` contient `3859 + 2773` et `r63` contient `6632`.
2. Demandez à bash de faire le calcul.
3. Quelle est la question parmi les 100 dont la réponse est incorrecte ? Utilisez un test pour répondre à cette question ?

[2] Beaucoup de commandes très courantes sont directement intégrées à bash pour accélérer leur exécution. Ces différentes commandes sont donc appelées en anglais *builtin commands* (commandes intégrées). La page de manuel de bash contient toute une section « Shell builtin commands ».