Introduction aux systèmes d'exploitation (IS1)

# TP 6 – bash avancé

Semaine du 23 octobre 2006

## Notions

### Interprétation d'une ligne de commande

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 de caractères sont spéciaux pour bash :

| Caractères | Exemple |
|---|---|
| { } | `ba{ba,bu}` → `baba` `babu` |
| \ | `\"\'\$\{\ \A B` → `"'${ A` `B` |
| $ | `$HOME` → `/home/roger` |
| ${…} | `${HOME}` → `/home/roger` |
| | `${HOME#/home}` → `/roger` |
| | `${HOME%ger}` → `/home/ro` |
| $(…) | `$(which true)` → `/bin/true` |
| `…` | `` `which true` `` → `/bin/true` |
| $((…)) | `$((12 + 4 * 2))` → `20` |
| "…" | `" a $HOME $((1+1)) ' "` → ` a /home/roger 2 '` |
| '…' | `' a $HOME $((1+1)) " '` → ` a $HOME $((1+1)) "` |

D'autres caractères 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 | Exemple |
|---|---|
| ~ | `~roger` → `/home/roger` |
| * | `ab*` → `abx` `abd` `abcde` |
| ? | `ab?` → `abx` `abd` |
| [...] | `a[abcd]x` → `abx` `adx` |

Qui plus est, on peut aussi activer des expansions supplémentaires avec la commande « `shopt -s extglob` ». Par exemple `?(...)` représente optionnellement le contenu des parenthèses : `?(a)bd?(x)` deviendra ainsi `abd` `bdx`. Ces extensions sont documentées dans la page de manuel de bash. *Vous pouvez ignorer ces extensions, au moins dans un premier temps.*

Les autres caractères spéciaux `!`, `&`, `<`, `>`, `|` et `;` ont déjà été vu au cours des TP précédents.

### Tests

Le TP3 introduisait l'utilisation de `&&` et `||` pour enchaîner des commandes de façon conditionnelle. De façon plus générale, on peut écrire :

`if` *cmd* `;` `then` *cmd1* `;` *cmd2* `;` `...` `;` `else` *cmd3* `;` *cmd4* `;` `...` `;` `fi`

qui exécute la commande « `cmd` » puis :

- exécute *cmd1* `;` *cmd2* `;` `...` `;` si la valeur de retour de « `cmd` » est 0 ;
- exécute *cmd3* `;` *cmd4* `;` `...` `;` dans les autres cas.

Il existe trois commandes très particulières que l'on peut utiliser comme « `cmd` » :

`true` cette commande réussit toujours (donc renvoie la valeur 0) ;

`false` cette commande échoue toujours (donc renvoie la valeur 0).

`test` cette commande permet d'effectuer de nombreux tests ; cela pourra prendre par exemple la forme :

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

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[1]. 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 »).

[1] 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 ».

## Exercices

**Exercice 1** – *Premières escapades*

1. Placez-vous dans le répertoire `tp6` du compte `monitIS1` en utilisant un raccourci avec `~`.
2. Affichez le contenu du fichier `premier exercice`. Pourquoi est-il heureux de penser à la touche « tab » ?
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 = /home/roger
10 * 2 = 20
```

   où `/home/roger` est la valeur de la variable `HOME` et `20` est le résultat de l'évaluation de la multiplication.
4. Répétez la question précédente en utilisant `"` pour échapper l'expression entière, dans les cas où cela est possible.
5. Répétez la question précédente en utilisant `'` pour échapper l'expression entière, dans les cas où cela est possible.

**Exercice 2** – *Expressions avant expansion*

1. Dans le répertoire `tp6` quelle *expression*[2] choisir pour que

   `cat` *expression*

   n'affiche que le contenu des fichiers `marin`, `marine`, `Martin` et `Martine`.
2. Quelle expression peut-on donner pour que cette concaténation soit dans le bon ordre, c'est-à-dire donne `1234` ?
3. En activant les expansions étendues de bash, quelle expression donner pour désigner les fichiers `a1.txt` et `a111.txt` mais pas `a11.txt` ?

**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.

[2] par *expression* on entend bien entendu un seul « mot » : `te?st` par exemple, et non simplement `marin marine Martin Martine`.

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 9 du TP 3 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** – *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 ?