Introduction aux commandes du shell Unix

Tahina Ramananandro
ramanana@clipper.ens.fr
http://www.eleves.ens.fr/~ramanana/enseigne/ifips

IFIPS – TP 1 et 2
4 et 5 mars 2008

Cette séance de mardi (ainsi peut-être que celle de mercredi) utilisera beaucoup de commandes UNIX. L'objet de cette partie est de comprendre leur fonctionnement global.

Il est vivement recommandé de faire tourner « en vrai » les exemples indiqués dans cette fiche ainsi que dans les fiches de TP.

Prenez votre temps. La machine est rapide, ne vous adaptez pas à sa vitesse. N'hésitez pas à me poser des questions.

1 Code de retour

Chaque commande produit un entier, appelé *code de retour*, qui n'est pas affiché. On utilise la convention suivante :

- 0 pour une commande qui termine sans erreur
- un nombre différent de zéro, pour une commande qui termine avec erreur

Le code de sortie d'une commande interrompue dépend de cette commande. En général il est non nul mais il n'y a pas de convention. Pour afficher le code de sortie de la dernière commande exécutée, on utilise la commande `echo $?`

Exemples :

- le programme `true` termine immédiatement avec le code 0 (termine toujours sans erreur)
- le programme `false` termine immédiatement avec le code 1 (termine toujours avec erreur)
- le programme `yes` affiche indéfiniment la ligne de texte `y`. Il ne termine jamais. Pour l'interrompre, l'utilisateur doit frapper CTRL+C. Alors, le programme sort avec le code 130.

Ces codes de sortie peuvent être utilisés pour composer les commandes entre elles de façon conditionnelle :

- A `&&` B
  Exécute d'abord A, puis n'exécute B que si A termine sans erreur (i.e. avec le code 0). Le code de retour est alors celui de B, sauf si A termine avec erreur auquel cas c'est celui de A.
- A `||` B
  Exécute d'abord A, puis n'exécute B que si A termine avec erreur (i.e. avec un code non nul). Le code de retour est alors celui de B, sauf si A termine sans erreur auquel cas c'est 0.
- `if` C `;` `then` A `;` `else` B `;` `fi`
  C'est le test conditionnel général. Il faut d'abord que C termine. Alors, en fonction du code de sortie de C, A (respectivement B) est exécuté si ce code est nul (resp. non nul).

2 Enchaînement

On peut enchaîner deux commandes du shell avec l'opérateur `;`

A `;` B

Alors, A est d'abord exécuté, puis, dès que A termine ou est interrompu, B est exécuté. Le code de retour est alors celui de B.

Cet opérateur est associatif.

3 Entrées-sorties

Les commandes UNIX dialoguent avec l'utilisateur.

- Elles produisent des lignes de texte et les affichent à l'écran : la *sortie standard*
- Souvent (mais pas toujours), elles lisent du texte entré par l'utilisateur, ligne par ligne : l'*entrée standard*

Exemples :

- la commande `cat` lit chaque ligne de texte au clavier et l'affiche à l'écran. C'est la « machine à écrire ». Ce programme termine lorsqu'il n'a plus d'entrées. L'utilisateur doit explicitement signifier qu'il a fini de taper ses entrées : il frappe alors CTRL+D, et dans ce cas `cat` termine et sort avec 0. **Attention,** ce concept est différent de CTRL+C qui interrompt l'exécution de `cat` qui sort avec 130 dans ce cas.
  - `cat`
    `Je suis une fougère`
    `Je suis une fougère`
    CTRL+D

    `echo $?`
    `0`

    `cat`
    `Ceci n'est pas une ligne de texte`
    `Ceci n'est pas une ligne de texte`
    CTRL+C

    `echo $?`
    `130`
- la commande `head -n` *quant* lit *quant* lignes de texte au clavier, les affiche et termine.
- la commande `tail -n` *quant* lit un nombre quelconque de lignes de texte au clavier mais n'affiche que les *quant* dernières. Au contraire de `head`, qui s'arrête dès qu'il a lu *quant* lignes, `tail` doit attendre, comme `cat`, que l'utilisateur ait fini ses entrées par CTRL+D. L'interruption par CTRL+C ne fait rien afficher à `tail`.
- Il convient de ne pas confondre la commande `cat`, avec la commande `echo` *message*, qui affiche directement le *message* sur une ligne sans lire de lignes de texte au clavier.
  - `cat`
    `Bonjour`
    `Bonjour`
    `Je suis une fougère`
    `Je suis une fougère`
    CTRL+D

    `echo Bonjour`
    `Bonjour`
  - la commande `echo -n` *message* est similaire à `echo` *message* sauf qu'elle ne revient pas à la ligne après avoir affiché *message*. Ceci est utile pour enchaîner un message et la sortie d'une autre commande sur la même ligne.

3.1 Pipe

Il est possible de faire interagir deux commandes A et B entre elles de façon à « brancher la sortie standard de A sur l'entrée standard de B ». Pour ce faire, on écrit alors :

A | B

Ainsi, les lignes de texte produites par A ne sont pas affichées à l'écran mais envoyées directement à B comme si l'utilisateur les avait tapées.

Dès que B termine, alors A est interrompu et le code de sortie de A | B est le code de sortie de B.

Exemple :

- `head -n 5 | tail -n 2`
  Affiche « les deux dernières des cinq premières », c'est-à-dire les quatrième et cinquième lignes tapées par l'utilisateur, et seulement celles-là.
- `yes | head -n 1`
  Affiche simplement `y`. Noter que `head` termine dès qu'il a reçu la première ligne produite par `yes`, qui est alors interrompu. Le code de sortie est 0.

L'opérateur | appelé *pipe* peut être enchaîné. On peut montrer qu'il est associatif.

3.2 Redirections de et vers des fichiers

3.2.1 Lecture d'un fichier

L'utilisateur peut lancer une commande A en lui faisant lire un *fichier* au lieu du clavier. Pour cela, il utilise :

A < *fichier*

Cette commande est équivalente à :

`cat` *fichier* | A

En effet, `cat` admet une autre utilisation : `cat` *fichier1 fichier2...* affiche les fichiers les uns à la suite des autres, sans attendre d'entrée de l'utilisateur.

3.2.2 Écriture d'un fichier

L'utilisateur peut lancer une commande B en lui faisant écrire le texte produit dans un *fichier* au lieu de l'écran. Pour cela, il utilise :

B > *fichier*

Mais on peut aussi utiliser la commande `tee` :

B | `tee` *fichier1 fichier2...*

Dans ce cas, chacun des fichiers recevra le texte produit par B, mais aussi l'écran.

4 Structures de contrôle

4.1 Utiliser le texte produit dans une commande

Il est possible d'utiliser le texte produit par une commande comme s'il avait été tapé en dur dans une commande. Dans ce cas, le texte produit est remis sur une seule ligne, les sauts de ligne étant remplacés par des espaces.

Si P est la commande productrice et U une commande qui voudra l'utiliser, alors la commande aura la forme U `$(P)`

Exemple : que fait la commande suivante ?

```
echo $(echo a ; echo b)
```

4.2 Boucle `for`

for *variable* in *liste* ; do *commande* ; done

Exécute la *commande* pour chaque valeur de *variable* comprise dans la *liste* . La liste est simplement une liste de mots séparés par des espaces. Typiquement, cette liste sera le texte produit par une commande et inclus par la construction $(...) précédente.

Pour chaque exécution de la *commande*, la valeur en cours de `variable` est accessible par la construction `$variable`

Exemple : que fait la commande suivante ?

```
for i in $(echo a ; echo b) ; do echo $i ; done
```

5 Aide sur les commandes UNIX

`man` *programme*

où *programme* est le nom de la commande sans ses paramètres.

Ceci ne marche pas pour `if`, ni pour `for` qui sont des commandes *internes* du shell. Pour ces commande (ainsi que `then else fi do done`), il faut lire `man bash` (`bash` est le nom du shell).