Compléments Unix

Juliusz Chroboczek

11 Septembre 2006

1 Interpréteurs de commandes

Il existe aujourd'hui de nombreux interpréteurs de commandes (*shells*), tous subtilement différents. *Unix 7th Edition* utilisait le *Bourne shell* écrit par Steve Bourne, qui se trouve à `/bin/sh`. Il existe de nombreuses versions améliorées du *Bourne shell*, notamment *Bash* (`/bin/bash`, `/usr/bin/bash` ou `/usr/local/bin/bash` selon les systèmes) et le *Korn Shell* dû à David Korn (`/bin/ksh` ou `/usr/bin/ksh`).

Il existe aussi un shell incompatible avec le Bourne Shell, le *C shell* (`/bin/csh` ou sa version améliorée `/usr/bin/tcsh`). Le C shell est habituellement signalé par une invite qui se termine par un signe pourcent « `%` » au lieu du signe dollar « `$` » habituel.

Lorsqu'on se trouve sur une machine qui utilise le C shell par défaut, il suffit, pour lancer un shell plus usuel, de taper `bash` ou `ksh` à l'invite `%`.

2 Variables d'environnement

Lorsque le shell exécute un programme (une application, une commande), il lui passe un *environnement*, soit un ensemble de chaînes de la forme

```
NOM=valeur
```

par exemple

```
EDITOR=emacs
```

Une application peut varier son comportement selon les valeurs des variables d'environnement ; par example, utiliser l'éditeur de texte spécifié par `EDITOR`.

2.1 Visualisation des variables d'environnement

Dans une commande shell, la notation `${EDITOR}` est remplacée par la valeur de la variable `EDITOR` ; ainsi, on pourra vérifier quel éditeur est utilisé par défaut en tapant

```
$ echo ${EDITOR}
```

La commande `set` donne la liste de toutes les variables d'environnement et de leurs valeurs.

```
$ set | fgrep EDITOR
```

2.2 Affectation des variables d'environnement

Il est possible de changer ou d'ajouter des variables d'environnement au shell courant à l'aide de la commande `export` ; par exemple, on pourra changer la valeur de la variable `EDITOR` en tapant[1]

```
$ export EDITOR=emacs
```

Pour supprimer la définition d'une variable d'environnement, on peut utiliser la commande `unset`, qui prend un seul argument, le nom de la variable d'environnement à supprimer.

```
$ unset EDITOR
```

2.3 Affectation temporaire des variables d'environnement

Pour affecter une variable d'environnement uniquement pour la durée de l'exécution d'une commande, on fait précéder l'invocation de la commande par l'affectation. Par exemple,

```
$ EDITOR=emacs mutt
```

2.4 Quelques variables d'environnement

`EDITOR` : l'éditeur par défaut.
`PAGER` : le paginateur par défaut (`more`, `less`, *etc.*).
`HOME` : le répertoire *home* de l'utilisateur.
`PATH` : la suite de répertoires où les commandes sont recherchées, séparés par des deux-points « `:` ».

3 Tâches de fond

Par défaut, le shell attend la fin de l'exécution d'un processus qu'il a lancé — on dit alors que le processus est une tâche *en avant-plan* (*foreground*). En conséquence, le shell est occuppé même si le processus n'utilise jamais son entrée ni sa sortie.

Le shell supporte aussi une notion de tâche en *arrière-plan* (*background*) ou *tâche de fond*. Une tâche en arrière-plan n'occupe pas de terminal et peut soit cesser de s'exécuter (dans quel cas on dit qu'elle est *suspendue*) soit continuer à s'exécuter.

3.1 Invocation en tâche de fond

Pour exécuter un programme en tâche de fond, on fait suivre son invocation par « `&` ».

Par exemple, si l'invocation de `fgrep` sur un fichier prend un temps important, on peut libérer le terminal pour d'autres commandes en l'invocant comme ceci :

[1] Certaines versions très anciennes du Bourne Shell ne comprennent pas cette syntaxe, et il faudra alors taper `EDITOR=emacs` puis `export EMACS`.

```
$ fgrep archives </var/log/http-access.log >http-archives.log &
```

3.2 Tâches suspendues

Il est possible de faire passer une tâche de l'avant-plan vers l'arrière-plan en tapant la touche `^Z` (Control-Z)[2]. La tâche en avant-plan est alors suspendue et ne s'exécute plus jusqu'à ce qu'on la réveille à l'aide d'une des commandes `fg` ou `bg` (voir ci-dessous).

3.3 Commandes liées aux tâches

`jobs` : donne la liste des tâches du shell courant. Avec l'option `-l`, elle donne aussi les identificateurs de processus *pid* associés aux tâches.
`fg` : fait passer en avant-plan (*foreground*) une commande qui est en arrière-plan ; si la tâche est suspendue, son exécution recommence. Cette commande prend un seul argument, qui est le signe « `%` » suivi du numéro de la tâche (tel qu'il est donné par `jobs`).
`bg` : réveille une tâche suspendue. La tâche se remet à s'exécuter mais reste en arrière-plan (*background*). La syntaxe de cette commande est analogue à celle de la commande `fg`.

3.4 Application aux commandes graphiques

Lorsqu'on travaille sous un système de fenêtres, la commande `emacs` n'interagit pas avec le terminal. Il est donc possible de l'exécuter en arrière-plan, et ainsi faire l'économie d'une fenêtre de terminal, en invoquant Emacs de la façon suivante :

```
$ emacs &
```

Il arrive cependant que l'on oublie de taper le symbole `&` lorsqu'on invoque Emacs. Il est alors possible de corriger la situation *a posteriori* de la façon suivante :

```
$ emacs
^Z
[1]+  Stopped                 emacs
$ bg
[1]+ emacs &
$
```

La même chose s'applique évidemment aux autres commandes graphiques (`xdvi`, `gv` et `ghostview`, *etc.*).

4 Réduction de la priorité

Par défaut, Unix essaie de partager le temps du processeur entre processus de façon juste (*fair*). Lorsqu'on lance un programme dont l'exécution risque de durer longtemps, il est gentil (*nice*) de réduire sa priorité afin d'éviter de gêner les autres utilisateurs.

[2]Vous remarquerez la différence entre la notation shell « `^Z` » et la notation Emacs, « `C-z` ».

La commande `nice` permet d'exécuter un programme avec une priorité diminuée. Elle prend en arguments le nom et les paramètres du programme à exécuter.

```
$ nice fgrep archives </var/log/http-access.log >http-archives.log &
```

En pratique, la commande `nice` n'est jamais utilisée.

5 Accès à distance

La commande `ssh` permet d'accéder à une machine à distance. Elle prend un seul argument, qui spécifie le nom de l'utilisateur et la machine à laquelle on veut accéder.

```
$ ssh jch@machine.exemple.fr
Password:
```

La commande `scp` permet de faire une copie d'un fichier entre machines. Sa syntaxe est analogue à celle de la commande `cp`, sauf qu'au moins un des fichiers doit contenir une spécification d'utilisateur et de machine.

```
$ scp jch@machine.exemple.fr:enseignement/bioinfo/tp1.tex .
```

ou bien

```
$ scp jch@machine.exemple.fr:/home/jch/enseignement/bioinfo/tp1.tex .
```