# Commandes Unix : compléments

Juliusz Chroboczek

15 Septembre 2009

## 1 Expansion de paramètres

Un paramètre de ligne de commande qui contient l'un des caractères « * » ou « ? » est un *motif* pour l'ensemble de tous les fichiers qui peuvent être obtenus en remplaçant « * » par une suite quelconque de caractères, et « ? » par un seul caractère. Par exemple, le motif « `ala*` » représente tous les fichiers qui commencent par « `ala` », et « `*.tex` » représente tous les fichiers qui se terminent par « `.tex` ».

Lorsqu'un tel motif apparaît dans une ligne de commande, il est remplacé par les noms de tous les fichiers présents sur le système de fichiers qui correspondent à ce motif. Ainsi,

```
$ echo *.tex
```

pourra par exemple afficher

```
tp0.tex tp1.tex guide-unix.tex
```

et on pourra copier tous ces fichiers en exécutant une commande de la forme

```
$ cp *.tex ~/archives/
```

## 2 Permissions et privilèges

### 2.1 Utilisateurs et groupes

Chaque utilisateur d'un système Unix a un *nom d'utilisateur*, unique sur le système, et appartient à un *groupe*. Par exemple, l'utilisateur `jch` des machines de l'UFR d'informatique appartient au groupe `enseignants`. À chaque nom d'utilisateur et groupe sont associés un certain nombre de *privilèges* ou *droits*, comme par exemple celui de modifier un fichier donné ou d'accéder au CD-ROM.

Il existe un compte distingué, appelé *root*, qui a tous les privilèges possible. Comme ce compte est très puissant, et permet très facilement de casser le système, on évite normalement de passer du temps en étant *root* ; les commandes `su` et `sudo` permettent de devenir *root* de façon temporaire.

### 2.2 Permissions de fichier

Chaque fichier dans le système de fichiers appartient à un utilisateur, son *propriétaire*, et à un groupe, son *groupe propriétaire*. Il est aussi accompagné d'un ensemble de *permissions*, qui indiquent quels utilisateurs ont le droit de le manipuler.

Les permissions d'un fichier consistent de neufs bits. Ce sont les droit d'exécution[1], d'écriture et de lecture pour le propriétaire du fichier, les autres membres du groupe, et les autres utilisateurs.

Les permissions sont affichées par la commande `ls -l`, et changées par la commande `chmod`. Le propriétaire peut être changé à l'aide de la commande `chown`.

## 3 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`.

### 3.1 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 | grep EDITOR
```

### 3.2 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[2]

```
$ 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
```

1 Vous remarquerez la différence avec Windows, où le droit d'exécution est encodé dans le nom du fichier — un fichier sous Windows est exécutable si son nom se termine par `.exe` ou `.com`.

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

### 3.3 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
```

### 3.4 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 « `:` ».

## 4 Redirections, tubes

L'endroit d'où un processus Unix obtient ses entrées s'appelle son *entrée standard* ; l'endroit où il écrit sa sortie s'appelle sa *sortie standard* ; et l'endroit où il affiche ses messages d'erreur s'appelle sa *sortie d'erreur*.

Normalement, l'entrée standard est connectée au clavier, tandis que la sortie et la sortie d'erreur sont connectées à l'écran. Il est possible de connecter ces *descripteurs* à des fichiers, ou à d'autres processus.

### 4.1 Redirections

On connecte la sortie standard d'un processus à un fichier en faisant suivre l'invocation d'un programme d'un signe supérieur « `>` » et du nom de fichier vers lequel on veut la rediriger. Par exemple, la commande suivante stocke la chaîne « `Ala ma kota` » dans un fichier nommé `ala.text` :

```
$ echo Ala ma kota > ala.text
```

De même, l'entrée standard peut être redirigée en faisant suivre l'invocation d'un programme d'un signe inférieur « `<` » :

```
$ sort < ala.text
```

Normalement, la sortie d'erreur n'est pas redirigée : même si la sortie standard l'est, les messages d'erreur apparaissent sur le terminal. Cependant, on peut rediriger la sortie d'erreur en faisant suivre l'invocation d'un programme de la syntaxe « `2>` » :

```
$ grep -r Ala ~/enseignement 2> erreurs.log
```

Rediriger la sortie d'erreur est particulièrement utile lorsqu'un processus s'exécute en tâche de fond et on ne veut pas qu'il nous dérange :

```
$ grep -r Ala ~/enseignement > resultats.log 2> erreurs.log
```

### 4.2 Tubes

On peut aussi connecter la sortie standard d'un processus à l'entrée standard d'un autre. Pour cela, on sépare les invocations de deux programmes par le symbole « | ». Par exemple, la ligne de commande suivante connecte la sortie du processus qui exécute `ls` à l'entrée du processus qui exécute `grep` :

```
$ ls | grep ala
```

## 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 .
```

La commande `scp` est à connaître, car elle est installée sur toutes les machines Unix récentes. Il existe cependant des commandes plus puissantes de copie de fichiers entre machines ; personnellement, j'utilise souvent `rsync` et `unison`.

**Utilisation depuis Windows** Le programme `putty` est une version pour Windows de `ssh`, et permet donc d'utiliser une machine Unix depuis une machine Windows. Vous le trouverez sur `http://www.chiark.greenend.org.uk/~sgtatham/putty/`.