Concentré de commandes Unix

Vincent LOZANO — Frédérique BARRÉ — Laurent DEFOURS

11 décembre 2010

N.B. : Dans la présentation des commandes :
- ce qui est indiqué entre crochets ⟨...⟩ doit être défini par l'utilisateur ;
- ce qui est présenté entre crochets [...] est facultatif ;
- les pointillés `...` indiquent que l'on peut répéter l'argument les précédant.

1 Aide

Pour obtenir de l'aide exhaustive sur une commande, on pourra faire appel aux pages de manuels en utilisant :

`man` ⟨*nom_commande*⟩

Pour avoir une aide plus succincte on pour essayer :

⟨*commande*⟩ `--help`

On pourra donc obtenir de l'aide avec les deux méthodes précédentes sur toutes les commandes indiquées dans ce document. On pourra également chercher de l'aide en faisant une recherche par mot clef :

`apropos` ⟨*mot_clef*⟩

2 Gestion des fichiers

Dans ce paragraphe, les références aux fichiers et aux répertoires peuvent être soit :
- une *référence absolue*, qui contient le chemin d'accès complet au fichier ou répertoire à partir de la racine de l'arborescence (elle commence toujours par `/`) ;
- une *référence relative*, qui contient éventuellement le chemin d'accès au répertoire à partir du répertoire courant, chemin pouvant faire intervenir le répertoire père noté « `..` ».

2.1 Montages des systèmes de fichiers

Pour avoir une idée des points de montages des différentes systèmes de fichiers sur l'arborescence globale, on pourra utiliser :

`mount`

ou, pour avoir une information un plus précise, et notamment les taux d'occupations de chacune des partitions :

`df -h`

Pour ce qui concerne les supports amovibles (disquettes, cédéroms, clefs Usb), on procédera en trois étapes :

1. après avoir inséré le support dans le lecteur, on monte le système de fichier avec la commande :

   `mount` ⟨*point de montage*⟩

où ⟨*point de montage*⟩ est le répertoire :
- `/Cdrom` pour un cédérom ;
- `/ClefUsb` pour une clé Usb.

2. on procède ensuite à la lecture où à l'écriture dans le répertoire correspondant ;
3. enfin on démonte le système de fichier avec la commande :

   `umount` ⟨*point de montage*⟩

   Attention : tout retrait du support avant le démontage peut entraîner des pertes de données.

Remarque : Ce qui précède est possible uniquement en salle « Dao » et exploite les informations se trouvant dans le fichier `/etc/fstab` indiquant les points de montage des supports amovibles.

2.2 Exploration de l'arborescence

`cd` ⟨*référence répertoire*⟩

Cette commande permet de changer de répertoire de travail (ou répertoire *courant*) ; celui-ci devient le répertoire indiqué par ⟨*référence répertoire*⟩ Sans argument, la commande permet de revenir à son répertoire *privé* (ou *home*, noté `~`), dont la référence absolue est rangée dans la variable **`HOME`**. On peut afficher le contenu de cette variable avec la commande « **`echo $HOME`** ».

`pwd [-P]`

Cette commande affiche la référence absolue du répertoire de travail actuel. Avec l'option **`-P`**, le chemin affiché ne contiendra pas de liens symboliques.

`ls [-`⟨*options*⟩`] [`⟨*référence répertoire*⟩`]`

Cette commande permet de lister le contenu d'un répertoire, avec les options suivantes (pouvant être combinées) :

- `-a` afficher tous les fichiers du répertoire y compris les fichiers commençant par « `.` » ;
- `-i` afficher le numéro d'index (i-noeud) de chaque fichier à gauche de son nom ;
- `-l` en plus du nom, afficher le type du fichier, les permissions d'accès, le nombre de liens physiques, le nom du propriétaire et du groupe, la taille en octets, et l'horodatage de la dernière modification ;
- `-t` trier le contenu des répertoires en fonction de la date et non pas en ordre alphabétique, les fichiers les plus récents étant présentés en premier
- `-R` afficher récursivement le contenu des sous-répertoires.

Si aucun argument n'est fourni, le contenu du répertoire courant « `.` » est affiché, trié par défaut par ordre alphabétique. La commande :

`file` ⟨*reference fichier*⟩

tente de trouver puis affiche le type de fichier en examinant son contenu.

2.3 Création et destruction de répertoires

`mkdir` ⟨*référence répertoire*⟩

Cette commande crée le répertoire indiqué par ⟨*référence répertoire*⟩, qui peut être une référence absolue ou relative.

`rmdir` ⟨*référence répertoire*⟩

Cette commande supprime, s'il est vide, le répertoire indiqué.

2.4 Création de fichiers

Pour créer des fichiers texte (que l'on pourra ensuite copier, déplacer...) on dispose de deux procédés simples :

- on peut effectuer la redirection vers un fichier de la sortie standard d'une commande (comme par exemple la commande **`echo`** qui se contente d'écrire chaque message passé en argument sur la sortie standard) ;
- on peut utiliser un éditeur de texte, comme **`emacs`**.

2.5 Visualisation du contenu d'un fichier

`cat [`⟨*référence fichier*⟩`] ...`

Cette commande affiche sur la sortie standard le contenu de chacun des fichiers indiqués (ou le contenu de l'entrée standard si aucun nom de fichier n'est fourni). On peut passer en argument plusieurs références de fichiers.

`less [`⟨*référence fichier*⟩`] ...`

Cette commande est un filtre permettant d'afficher un texte écran par écran :

- si l'argument ⟨*référence fichier*⟩ est défini, on affiche le contenu du fichier ;
- sinon, on lit sur l'entrée standard et on peut en particulier utiliser la commande **`less`** en sortie d'une autre commande, via un *tube* : ⟨*commande*⟩**`|less`**.

2.6 Manipulation des fichiers

`cp` ⟨*référence fichier*⟩ `...` ⟨*destination*⟩

Cette commande sert à copier des fichiers (et éventuellement des répertoires). :

- si ⟨*destination*⟩ est une référence de fichier, il ne peut y avoir que deux arguments et on fait une copie du premier fichier, copie dont la référence est ⟨*destination*⟩ ;
- si ⟨*destination*⟩ est une référence de répertoire, on copie dans ce répertoire chaque fichier indiqué en conservant le même nom.

`mv` ⟨*référence fichier*⟩ `...` ⟨*destination*⟩

Cette commande permet de déplacer ou renommer des fichiers :

- si ⟨*destination*⟩ est la référence d'un répertoire existant, on déplacera tous les autres fichiers dans ce répertoire ;
- sinon, s'il n'y a que deux fichiers indiqués, on déplacera le premier pour remplacer le second (si le second n'existe pas, cela revient à effectuer un renommage de fichier).

`rm [-`⟨*options*⟩`]` ⟨*référence fichier*⟩ `...`

Cette commande permet d'effacer des fichiers ou des répertoires, avec les options suivantes :

`-f` forcer : ne pas demander confimation à l'utilisateur ;

`-r` supprimer récursivement le contenu des sous-répertoires.

2.7 Exécution de fichiers

Pour exécuter un programme, il suffit de donner à l'interpréteur, la référence sur le fichier contenant l'exécutable. Cependant, on peut la plupart du temps, se passer de la référence complète, car le système cherche automatiquement les fichiers exécutables dans une liste de répertoires. Cette liste est stockée dans la variable d'environnement `PATH` dont on peut afficher le contenu avec la commande :

```
echo $PATH
```

En outre on peut interroger le système pour savoir où se trouve l'exécutable correspondant à une commande avec :

`type` ⟨*nom_commande*⟩

Cette commande affichera un message particulier si ⟨*nom_commande*⟩ est une commande interne à l'interpréteur de commande.

2.8 Liens symboliques

On peut créer dans l'arborescence, un fichier spécial appelé *lien symbolique*. Un lien symbolique possède une cible (à l'instar d'un raccourci sous Windows) et toute opération en lecture ou écriture sur le lien symbolique s'effectuera sur le fichier ou répertoire cible. Pour créer un lien symbolique on utlisera la syntaxe :

`ln -s` ⟨*reference de la cible*⟩ ⟨*nom du lien*⟩

3 Gestion des utilisateurs

`who`

Cette commande affiche les informations suivantes pour chaque utilisateur connecté :
- nom de connexion ;
- terminal ;
- heure de connexion ;
- nom d'hôte distant, ou numéro de terminal X.

`whoami`

Cette commande affiche le nom de *login* de l'utilisateur (un utilisateur peut lancer différents *shells* sous des noms de *login* différents).

`su -` ⟨*nom utilisateur*⟩

Cette commande permet à un utilisateur de se transformer temporairement en un autre utilisateur, pourvu qu'il en connaisse le mot de passe.

`chmod [-R]` ⟨*mode*⟩ ⟨*référence fichier*⟩ **`...`**

Cette commande modifie les permissions d'accès de chacun des fichiers indiqués, en suivant l'indication de mode, qui est de la forme :
- soit ⟨*type d'utilisateur*⟩**+**⟨*type d'accès*⟩ pour ajouter une autorisation ;
- soit ⟨*type d'utilisateur*⟩**−**⟨*type d'accès*⟩ pour en supprimer une ;

avec :

1. ⟨*type d'utilisateur*⟩ pouvant être **`u`** pour le propriétaire, **`g`** pour le groupe, **`o`** pour les autres ou **`a`** pour tout le monde ;
2. ⟨*type d'accès*⟩ pouvant être **`r`** pour la lecture, **`w`** pour l'écriture, **`x`** pour l'exécution (ou le parcours pour les répertoires).

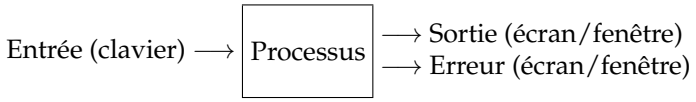
L'option **`-R`** permet de modifier récursivement les autorisations des répertoires et de leurs contenus.

4 Gestion des processus

Avant toute chose, quelques remarques générales sur la façon dont les processus sont gérés par Linux :
- un processus est *identifié à l'aide d'un numéro unique (PID)* sur une machine donnée ;
- un processus est *attaché à un terminal (TTY)* physique ou virtuel ;

– les processus sont organisés *en arborescence* : pour chaque processus, on connaît le *numéro du processus parent (PPID)*.

À chaque processus est également associé par défaut trois canaux de communication : un canal de sortie, d'entrée et d'erreur. Chacun de ces canaux est associé par défaut au terminal courant.



4.1 Commandes

Il est possible de lancer plusieurs commandes les unes après les autres avec la syntaxe :

⟨*commande*$_1$⟩ `;` ⟨*commande*$_2$⟩ `; ...`

D'autre part, la syntaxe :

⟨*commande*$_1$⟩ `&&` ⟨*commande*$_2$⟩

permet d'exécuter ⟨*commande*$_2$⟩ uniquement si l'exécution de ⟨*commande*$_1$⟩ s'est déroulé sans erreur. De manière analogue, la syntaxe :

⟨*commande*$_1$⟩ `||` ⟨*commande*$_2$⟩

exécutera ⟨*commande*$_2$⟩ uniquement si l'exécution de ⟨*commande*$_1$⟩ a échoué.

4.2 Redirections

On peut alors rediriger chacune de ces entrée/sortie standard sur un fichier physique, ainsi :

⟨*commande*⟩ `>` ⟨*fic*⟩

redirige la sortie de ⟨*commande*⟩ vers le fichier ⟨*fic*⟩. D'autre part, avec la syntaxe :

⟨*commande*⟩ `<` ⟨*fic*⟩

l'entrée de ⟨*commande*⟩ est alimentée par les données du fichier ⟨*fic*⟩. Enfin, avec la syntaxe :

⟨*commande*$_1$⟩ `|` ⟨*commande*$_2$⟩

1. ⟨*commande*$_1$⟩ et ⟨*commande*$_2$⟩ sont lancées en même temps ;
2. la sortie de ⟨*commande*$_1$⟩ est envoyée à l'entrée de ⟨*commande*$_2$⟩.

4.3 Tâches de fond ou « jobs »

Pour lancer une commande en tâche de fond, on utilise la syntaxe suivante :

⟨*nom commande*⟩ `&`

Le *shell* affiche alors une ligne du genre :

```
[1] 25647
```

Cette ligne indique que le *job* est associé au numéro 1, et que le PID du dernier processus dans le tube est 25647. On peut ensuite réaliser le *job control*, dans un terminal, à l'aide des commandes ci-après.

`jobs [-l]`

Cette commande liste tous les *jobs* associés *au terminal courant* en indiquant le numéro (1, 2...) de chaque *job*. L'option **`-l`** permet d'avoir en plus le PID de chaque job.

`Ctrl+z`

La frappe d'un caractère suspension (généralement **`Ctrl+Z`**) pendant l'exécution d'un processus permet de *stopper* ce dernier et de reprendre la main. Ce processus apparaîtra avec la mention « **`stopped`** » lorsqu'on exécute la commande **`jobs`**.

`bg [%`⟨*numéro job*⟩`]`

Cette commande permet de basculer une tâche en arrière plan (par exemple, un processus que l'on vient de stopper). On passe normalement en argument le numéro du *job* précédé du caractère « **`%`** » (**`%1`**, **`%2`** ...). Si aucun argument n'est fourni, c'est le *job courant* (indiqué par un « **`+`** » dans l'affichage réalisé par la commande **`jobs`**) qui est basculé en arrière plan.

`fg [%`⟨*numéro job*⟩`]`

Cette commande permet de faire passer l'exécution d'une tâche au premier plan : on perd alors la main au niveau du terminal associé. L'argument est défini exactement de la même façon que pour la commande **`bg`**.

`Ctrl+c`

Cette dernière séquence de touche permet de stopper la tâche lancée en avant plan dans le terminal.

4.4 Informations concernant l'ensemble des processus

`ps [-`⟨*options*⟩`]`

Cette commande permet d'afficher des informations relatives au processus en cours d'exécution (*tous* les processus et pas seulement les taches de fond). Par défaut, cette commande n'affiche que les processus attachés au terminal courant. On dispose des options suivantes :

- `-u` ⟨*user*⟩ afficher tous les processus appartenant à l'utilisateur ⟨*user*⟩ ;
- `-e` affichage de tous les processus de la machine ;
- `-f` permet d'afficher notamment le processus père du processus considéré ;
- `--forest` fait apparaître la généalogie des processus.

Pour avoir aperçu rapide de l'utilisation de la mémoire et des processus en cours d'exécution, on pourra utiliser les commandes :

`free`

et

`top`

4.5 « Tuer » un processus

`kill [-`⟨*signal*⟩`]` ⟨*PID*⟩ `...`

Cette commande permet d'envoyer un *signal* aux processus dont les PID sont indiqués. L'argument ⟨*signal*⟩ indique soit le nom soit le numéro du signal à envoyer. La liste des signaux disponibles peut être obtenue en exécutant « **`kill -l`** ».

Si on ne précise pas de signal, **`TERM`** est envoyé. Ce dernier tuera les processus qui ne l'interceptent pas. Si l'on désire tuer des processus qui interceptent **`TERM`**, il peut être necessaire d'envoyer le signal **`KILL`**, qui ne peut pas être intercepté.

On pourra également utiliser la commande **`killall`** permettant de « tuer » un processus en indiquant son nom au lieu de son pid :

`killall [-`⟨*signal*⟩`]` ⟨*nom de la commande*⟩

Par exemple, la commande :

```
killall emacs
```

stoppera toutes les instances du programme `emacs`.

5 Communication

`ssh [-X] ⟨hôte⟩ [-l ⟨login⟩]`

permet de se connecter sur la machine ⟨*hôte*⟩ en tant que l'utilisateur ⟨*login*⟩ (par défaut c'est l'utilisateur courant qui est utilisé). L'option `-X` permet de bénéficier de l'affichage des applications graphiques. Cette commande établit une connexion cryptée avec la machine ⟨*hôte*⟩ ; parmi les outils de connextion non cryptés existe la commande :

`telnet ⟨hôte⟩`

On peut envoyer un message sur un terminal où est connecté un utilisateur avec :

`write ⟨login⟩ [⟨term⟩]`

qui permet d'envoyer un message (qu'il faudra terminer par la séquence `Ctrl-d`) à l'utilisateur ⟨*login*⟩, sur le terminal ⟨*term*⟩ où il est connecté. Les terminaux où sont connectés les différents utlisateurs sont listés par la commande `who` (cf. § 3). On pourra explicitement activer ou désactiver la réception de messages en utilisant la commande :

`mesg ⟨activation⟩`

où ⟨*activation*⟩ pourra prendre les valeurs `y` ou `n` pour respectivement accepter ou refuser les messages.

Enfin, on pourra utiliser l'utilitaire `mutt` pour envoyer des messages électroniques avec la syntaxe :

`mutt [-a ⟨fichier attaché⟩] [-s ⟨sujet⟩] ⟨adresse⟩`

Par défaut cet utilitaire lancera un programme interactif de composition de message en mode texte. Il faut noter que si le sujet est composé de plusieurs mots il faudra l'entourer de guillemets. En envoyant le corps du message sur l'entrée de `mutt` on peut se passer du mode interactif :

`echo ⟨text⟩ | mutt ⟨arguments indiquées ci-dessus⟩`

ou :

`mutt ⟨arguments indiquées ci-dessus⟩ < ⟨fichier texte contenant le message⟩`

Voici un exemple :

```
echo voilà la doc. | mutt -a bidule.pdf -s "la doc du bazar" trucmuche@enise.fr
```

commande envoyant en attachement le fichier `bidule.pdf` dans un message destiné à l'adresse `trucmuche@enise.fr` dont le le sujet est « la doc du bazar », et dont le contenu et « voilà la doc ».

6 Archivage

On peut archiver le contenu d'un répertoire avec la commande suivante :

`tar cvfz ⟨archive⟩ ⟨répertoire à archiver⟩`

où ⟨*archive*⟩ doit porter l'extension `.tgz`. On pourra par exemple écrire :

```
tar cvfz mesfichiers.tgz tp-algo
```

le fichier `mesfichiers.gz` pourra être traité par un programme d'archivage sous Windows (`Winzip`, `7-zip`, ...). Un fichier archivé par `tar` pourra être décompacter avec la commande :

`tar xvfz ⟨archive⟩`

Notez qu'on peut décompacter une archive Zip, avec la commande `unzip`.

7 Personnalisation

Il est possible de personnaliser l'environnement de l'interpréteur de commande, en créant dans son répertoire privé un fichier nommé '`.bash_profile`'. Ce fichier est lu à chaque connexion, et à chaque exécution d'un terminal. Ce fichier pourra contenir des lignes du type :

`alias` ⟨*commande perso*⟩`='` ⟨*commandes*⟩`'`

Par exemple :

```
alias psu='ps -u durand'
```

fournira une nouvelle commande `psu`, affichant tous les processus de l'utilisateur `durand` (cf. 4.4).

8 Outils basiques

Unix dispose de nombreux outils élémentaires permettant de réaliser des opérations diverses et variées sur les fichiers. Parmi ceux-ci :

`grep` ⟨*motif*⟩ ⟨*fichier*⟩

affiche toutes les lignes de ⟨*fichier*⟩ contenant la chaîne de caractères ⟨*motif*⟩.

`wc -l` ⟨*fichier*⟩

affiche le nombre de lignes de ⟨*fichier*⟩..

`sort` ⟨*fichier*⟩

tri un le fichier.

Remarque : toutes ces commandes peuvent s'utiliser avec un pipe à la place d'un fichier en argument. Par exemple : `ps -ef | grep emacs`.