Architecture d'un OS
Introduction à Linux

Cédric Meuter Eythan Levy

1 Le *shell*

1.1 Introduction

Le *shell* est l'interpréteur de commandes que vous utilisez lorsque vous lancez un terminal. Dans la plupart des cas, sous Linux, le *shell* par défaut est `bash` (bourne again shell). Il en existe bien d'autres (`sh, csh, zsh, ksh, tcsh, ...`). Un intepréteur de commande, comme son nom l'indique, attend que vous introduisiez une commande au clavier, et lorsque vous appuyez sur `[Enter]`, il interprète cette commande. `bash` offre, en plus, toute une série de petites fonctionalités bien utile comme la complétion automatique, l'historique des commandes, la redirection d'I/O, etc. Nous passerons tout cela en revue par la suite.

1.2 Variables d'environnement

Avant de détailler toutes ces fonctionalités, il nous faut nous attarder sur la notion de variables d'environnement. Ce sont des variables maintenues par le *shell*. En `bash`, on peut modifier l'état de ces variables d'environnement en utilisant l'opérateur `=`. Ensuite pour consulter la valeur de la variable, on peut utiliser la commande `echo`. Ainsi, par exemple, la commande `MYVAR=toto` assignera la valeur `toto` à la variable `MYVAR`. Si ensuite on en consulte sa valeur avec la commande `echo $MYVAR`, le *shell* affichera `toto`. Parmis toutes ces variables, on peut distinguer `HOME` qui contient le chemin de votre home directory, `PWD` qui contient le chemin du répertoire courrant et `PATH` qui contient une liste de chemins de répertoire dans lesquels le *shell* va chercher les exécutables lorsque vous tapez une commande. Nous y reviendrons plus tard.

1.3 Lancer une commande

Lorsque vous tapez une commande (`ls` par exemple), la première chose que fait le *shell* est d'identifier l'exécutable correspondant à la commande. Deux cas sont possibles:

1. Soit vous précisez le chemin de l'exécutable dans la commande (`/bin/ls -l -a`), auquel cas, le *shell* n'a plus rien à faire. Vous pouvez précisez un chemin absolu (à partir du `/`, comme `/bin/ls`) ou relatif au répertoire courant (à partir du `./` comme `./a.out`).

2. Soit vous ne précisez pas. Dans ce cas, le *shell* cherche si la commande se trouve dans un répertoire référencé dans la variable d'environnement `PATH`. Par exemple, si j'exécute la commande `ls -l -a`, le *shell* va parcourir tous les répertoires référencés dans le `PATH` jusqu'à en trouver un contenant l'exécutable (`ls` dans notre cas).

Une fois, l'exécutable identifié, il est lancé par le *shell* (`/bin/ls` dans notre exemple) en lui passant les paramètre adéquat (`-l -a` dans notre exemple). Par défaut, *le shell* attend la fin de l'exécution avant de reprendre la main pour accepter de nouvelle commande. Vous pouvez

remédier à cela en ajoutant un `&` à la fin de votre commande. Cela permet à la commande de s'exécuter en *background*. Le *shell* n'attend pas la fin de l'exécution de la commande et est directement prêt à recevoir d'autres commandes. Cela peut être utile lorque vous lancer un éditeur graphique à partir du *shell*. Si vous avez oublié le `&`, vous pouvez néanmoins vous en sortir en mettant le programme sur pause en appuyant sur `[Ctrl+z]`. Ensuite, vous récupérez le contrôle de `bash`. Pour que la commande mis en pause reprenne en background, vous pouvez utiliser la commande `bg` (*background*. Si vous voulez que l'exécution reprenne simplement (vous perderez à nouveau le contrôle du *shell* dans ce cas), vous pouvez utiliser la commande `fg` (*foreground*). Notez également, que vous pouvez également arrêter abruptement l'exécution d'une commande en *foreground* en appuyant sur `[Ctrl+c]`

1.4 Quelques racourcis

Le *shell* offre également une série de racourci vous permettant de dénommer plusieurs fichiers à la fois. Le caractère `*` permet de faire du *pattern matching*. Par exemple `/home/cmeuter/archi/*.c` dénote tous les fichiers se terminant par `.c` qui se trouve dans le répertoire `/home/cmeuter/archi`. Si aucun répertoire n'est précisé, le pattern matching se fera dans le répertoire courant. Ainsi, `*.h` dénote tous les fichiers se terminant par `.h` du répertoire courant.

`bash` en particulier offre également une série de racourcis bien utiles. Par exemple, la touche `[Tab]` permet de faire de la complétion automatique. Si plusieurs choix existent, `bash` vous les proposera tous.

De plus `bash` maintient un historique de toutes les commandes que vous tapez dans un terminal. Vous pouvez ensuite, vous balader dans cet historique en utilisant les touches `[↑]` et `[↓]`. Si vous voulez consulter toute l'historique, vous pouvez utilisez la commande `history`. Vous pourrez alors remarquer que chaque commande identifiée par un nombre. Si vous tapez `!<nombre>`, vous exécuterez la commande identifiée par `<nombre>`.

2 L'arborescence des fichiers

2.1 Introduction

Sous UNIX en général, et Linux en particulier, la notion de fichier est primordiale. Elle englobe tant le concept classique de *fichier de données* et de répertoires (*directories*) que celui, plus abstrait, de *fichiers device*, sortes d'interfaces permettant à un programme d'accéder à un driver de périphérique en utilisant les mêmes méthodes que pour l'écriture et la lecture dans un fichier. D'autres types de fichiers existent encore, dont nous vous parlerons lors d'un TP ultérieur.

Sous les sytèmes UNIX, l'ensemble des fichiers appartient à une hiérarchie unique de répertoires, dont l'ancêtre général est le répertoire "/". Il existe également une notion de *home directory*, qui est le répertoire dans laquelle un utilisateur est automatiquement parachuté lorsqu'il lance un *shell* (interpréteur de commandes).

2.2 Structure de l'arborescence

Sous linux, les premiers répertoires dans l'arborescence sont entre autres:

1. `/bin`: contient les fichiers exécutables des programmes utilitaires les plus standards.
2. `/dev`: contient les fichiers device associés aux différents périphériques.
3. `/home`: contient les *home directories* de tous les utilisateurs. La home directory de l'utilisateur `username` sera donc `/home/username`.

4. `/mnt`: contient les points de montage (cette notion sera vue en détail au cours théorique) des différents périphériques. Par exemple, les fichiers présents sur une disquettes seront disponibles dans l'arborescence globale à partir du fichier `/mnt/floppy`.

5. `/root`: un répertoire spécialement dédié au super-user de la machine.

6. `/tmp`: ce répertoire, dans lequel tous les utilisateurs de la machine ont le droit d'écrire, est utilisé pour stocker des fichiers utilisés temporairement, et dont la perte ultérieure ne porte pas à conséquence. N'y stockez donc jamais de fichiers importants dont vous pourriez avoir à nouveau besoin par la suite.

7. `/usr` : ce répertoire contient principalement des programmes à l'attention des utilisateurs du système.

2.3 Consultation de l'arborescence

2.3.1 `pwd`

La commande `pwd` permet, à tout moment, de savoir dans quel repértoire vous vous trouvez (dite *directory courante*). Elle affiche simplement le nom de ce répertoire à l'écran.

2.3.2 `ls`

La commande `ls` suivie d'un nom de répertoire est l'une des plus utilisées. Elle permet de *lister* le contenu du répertoire, ç.-à-d. d'afficher à l'écran la liste des fichiers contenus dans ce répertoire. Lorsque la commande est utilisée sans aucun nom de répertoire, elle affiche le contenu du répertoire courant.

La variante `ls -a` permet d'afficher *tous* (`-a` = all) les fichiers de la directory, ç.-à-d. également les fichiers cachés, qui sont des fichiers identifiés par le caractère " . " qui précède leur nom. Lors de l'utilisation de `ls -a`, le fichier appelé " . " désigne le répertoire courant, tandis que le fichier " . . " désigne le répertoire parent du répertoire courant.

2.3.3 `cd`

La commande `cd` suivie du nom d'un directory permet de changer de directory courante. Utilisée sans paramètre, elle replace l'utilisateur dans sa home directory.

2.3.4 `cat`

La commande `cat` permet d'afficher à l'écran le contenu d'un fichier. Elle est peu pratique, car une fois le fichier affiché, elle ne permet pas de naviguer dans celui-ci.

2.3.5 `less`

La commande `less`, plus puissante que `cat`, affiche le contenu du fichier, et permet ensuite de s'y déplacer pour le consulter en détail en utilisant les touches PgUp, PgDn, Home, End, espace (passage à la page suivante), ainsi que les flêches. Elle permet également d'effectuer des recherches dans le fichier. Si, par exemple, vous voulez rechercher "mot" dans votre fichier, vous pouvez le faire en tapant "`/mot`". Ensuite, pour trouver les occurences suivantes, vous pouvez utilise la touche "n". Finalement, la touche "q" vous permet de quitter less.

2.3.6 `grep`

La command `grep` permet d'effectuer des recherche dans des fichier. Entre autre, `grep mot fichier1 fichier2 ...` permet de rechercher toutes les lignes dans les fichiers spécifiés en paramètres ou apparait le mot donné en premier paramètre. Utilisé avec l'option `-n`, `grep` affiche également les numéros de ligne où le mot apparait.

2.3.7 `find`

La command `find` permet de rechercher un ou plusieurs fichier suivant certains critère (nom, date, taille, ...). Par exemple, la commande `find repertoire -name nom_du_fichier` permet de rechercher si un répertoire donné, ainsi que tous ses sous répertoires dans l'arborescence, contient un certain fichier, et affiche les résultats trouvés.

2.3.8 Exercices

1. Inspectez en détail votre home directory (en incluant les fichiers cachés). Reconnaissez-vous tous les fichiers ?
2. Recherchez dans quel répertoire se trouve le fichier exécutable du programme `ls` (ce fichier s'appelle `ls` tout simplement), grâce à `find`. Vous déplacer dans le répertoire trouvé, et vérifier que le fichier s'y trouve réellement.
3. Explorez le répertoire `/usr/bin`. Y reconnaissez-vous certains programmes ?
4. Lancez l'utilitaire *nautilus* et testez-le. Retrouvez ensuite le fichier `ls` identifié précédemment.

2.4 Modification de l'arborescence

2.4.1 `cp`

Le copiage d'un fichier d'un répertoire vers un autre se fait par la commande `cp source destination`. Utilisée avec l'option `-r` la commande permet également de copier des répertoires (et tout leur contenu, récursivement).

2.4.2 `mv`

Le déplacement d'un fichier vers un autre répertoire (ou alternativement son renommage dans le même répertoire) se fait à l'aide de la commande `mv source destination`.

2.4.3 `rm`

La suppression de fichiers classiques se fait par la commande `rm fichier1 fichirer2...` L'option `-f` permet d'éliminer les demandes de confirmation de suppression des fichiers

2.4.4 `mkdir`

La commande `mkdir repertoire` permet de créer un répertoire appelé `repertoire` dans la directory courante.

2.4.5 `rmdir`

La commande `rmdir` permet quant à elle de supprimer un répertoire vide. La suppression d'un répertoire non-vide et de tout son contenu peut se faire par la commande `rm -r repertoire`.

2.5 Outils graphiques

Il existe également des outils vous permettant d'explorer la hiérarchie des fichiers en mode gaphique, tels que *Konqueror* sous KDE ou *Nautilus* sous Gnome.

2.5.1 Exercices

1. Téléchargez dans votre home directory les fichiers `list.c`, `list.h`, `main.c` et `Makefile` disponibles sur la page web du TP[1].
2. Créez un répertoire `tparchi` dans votre home directory et copiez-y ces fichiers.
3. Recherchez dans les trois fichiers C toutes les lignes contenant `prepend_list`.
4. Renommez le fichier `Makefile` en `makefile`.
5. Compilez le programme en utilisant `make` et testez-le.
6. Arrangez vous pour que ce programme puissent être lancé sans le "./"
7. Pour terminer, supprimez tous les fichiers, ainsi que le répetoire `tparchi`.

[1] `http://www.ulb.ac.be/di/ssd/cmeuter/archi/`