Formation Linux: les bases du système

Julien De Bona <julien@bawet.org>

Introduction

Ce document essaie de présenter les concepts fondamentaux gouvernant un système Linux et de donner les bases pour poursuivre son apprentissage seul, notamment en lisant les pages de manuel et en expérimentant. Il suppose d'avoir un peu manipulé Linux car les concepts abordés ont de multiples connexions entre eux et seront inévitablement évoqués avant d'avoir pu être détaillés. Il s'agit donc de replacer les connaissances acquises dans leur contexte et de les formuler de manière plus générale. Il est aussi probable que ce document ne sera pas autosuffisant.

Principe fondamental: tout est fichier

Sous Linux et les autres systèmes UNIX, tout est fichier. Il ne faut donc pas se limiter au concept de fichier sur disque, mais plutôt l'étendre à une interface derrière laquelle peut se cacher n'importe quoi. Cette interface consiste en les opérations suivantes:

- ouverture: il ne s'agit pas d'ouvrir un document avec un traitement de texte mais de déclarer au système d'exploitation qu'on désire travailler avec le fichier et d'obtenir un "handle" qui servira à manipuler le fichier dans la suite du programme; l'ouverture positionne également un curseur de lecture/écriture (voir suite).
- lecture d'un ou plusieurs octets à partir du curseur de lecture/écriture; le curseur est déplacé au fur et à mesure de la lecture
- écriture d'un ou plusieurs octets à partir du curseur de lecture/écriture; le curseur est déplacé de la même manière
- positionnement du curseur de lecture/écriture
- ioctl: ce sont des autres opérations, définies selon le fichier, et utilisées pour piloter les périphériques
- fermeture: il s'agit d'indiquer au système d'exploitation qu'on a fini de travailler avec le fichier

Plusieurs types de fichiers existent:

- fichier normal
- répertoire
- pipe (tuyau)

- périphérique bloc ou caractère
- lien symbolique
- socket

Les fichiers normaux

Ils contiennent une suite d'octets et contiennent soit un programme exécutable, soit des données quelconques.

Les répertoires

Les répertoires sont des fichiers particuliers, qui contiennent d'autres fichiers. Chaque répertoire contient toujours au moins deux fichiers: ".", qui correspond au répertoire lui-même, et "..", qui correspond au répertoire parent, afin de pouvoir remonter dans l'arborescence. Ainsi, un répertoire est un sous-répertoire de ses sous-répertoires.

Les périphériques

Le fichier de périphérique ne contient aucune donnée. Les opérations effectuées sur lui sont passées à un pilote qui lui est associé par deux nombres (le majeur et le mineur). Un périphérique de type caractère peut être manipulé comme un fichier normal (lecture/écriture d'un nombre quelconque d'octets). Les imprimantes, ports séries et le clavier sont par exemple des périphériques caractères. Par contre, un périphérique de type bloc ne peut être lu et écrit que par blocs (typiquement 512 octets), et le système d'exploitation fait appel à des caches en mémoire pour améliorer les performances. Ce sont typiquement les disques durs, disquettes et CDs. Selon le périphérique, certaines opérations peuvent être impossibles (positionnement sur une bande magnétique, écriture sur un CD-ROM ...)

Les liens symboliques

Ce fichier contient simplement un nom de fichier à manipuler en lieu et place de lui-même. L'on peut donc manipuler le lien comme s'il était le fichier pointé.

Les pipes (tuyaux)

Le pipe est un fichier particulier: un processus peut l'ouvrir pour y écrire et un autre pour lire ce que le premier y a écrit. C'est donc un moyen de communication entre deux processus. De par sa nature, il n'est pas possible de se positionner dans un pipe.

Les sockets

Il s'agit d'un autre fichier permettant des communications inter-processus. Les sockets sont utilisées pour la programmation réseau. Les sockets ont un domaine, correspondant à un protocole, par exemple TCP, UDP ou UNIX (communications sur un même ordinateur).

L'arborescence des fichiers

La plupart des fichiers sont rangés dans une arborescence unique de répertoires. A la racine de cette arborescence, on trouve un répertoire nommé "/". Comme il n'a pas de répertoire au-dessus de lui, le répertoire ".." qu'il contient correspond à lui-même. Chaque répertoire peut contenir[1] d'autres fichiers, y-compris d'autres répertoires, ce qui crée toute l'arborescence. On référence un fichier de l'arborescence par son chemin, qui peut être absolu (à partir de la racine: /etc/fstab, /usr/bin/du) ou relatif (à partir d'un endroit de l'arborescence appelé "répertoire courant": fstab, ../../etc/fstab, ./bin/du, bin/du)

Un fichier n'apparaît pas forcément dans cette arborescence. Par exemple, un programme peut créer en mémoire un fichier normal pour son usage propre. Les pipes, d'usage très courant (le fameux "|"), sont rarements rattachés dans l'arborescence. Il en va de même des sockets: seules les sockets de type UNIX sont rattachées à l'arborescence. Quelques fichiers de périphériques ne figurent également pas dans l'arborescence, par exemple les interfaces réseaux (eth0 ...).

Les répertoires principaux

/bin: contient les commandes de base du système, nécessaires au démarrage.

/sbin: comme /bin, mais ces commandes sont en général réservées à l'administrateur et les utilisateurs normaux n'en ont pas besoin.

/etc: contient les fichiers de configuration globaux.

/etc/init.d: contient des scripts de démarrage et d'arrêt des services.

/dev, contenant les fichiers de périphériques. Il est un peu particulier car il peut être géré de 3 manières:

- non géré: il contient de nombreux fichiers, un par périphérique potentiellement connecté à l'ordinateur.
- géré dynamiquement par le kernel (devfs): quand un nouveau périphérique est détecté, le kernel crée le fichier correspondant. Cela a l'avantage de limiter les fichiers de /dev au minimum. Les fichiers gérés par le kernel n'ont cependant pas les mêmes noms que les

1 En fait, le répertoire ne contient pas de fichiers, mais des informations sur eux.

fichiers classiques, et un daemon gère la création de liens afin que les périphériques puissent être référencés par leur nom habituel. Cette méthode est désormais dépassée et remplacée par udev.

- géré par udev, un programme créant les fichiers à la demande, pour remplacer avantageusement devfs, dont il est le successeur pour linux 2.6.

/home: contient pour chaque utilisateur un répertoire lui appartenant et où il peut stocker ses fichiers.

/tmp: contient des fichiers temporaires. Il est généralement effacé au démarrage du système et son sticky bit est positionné: tout le monde peut y écrire mais pas effacer les fichiers des autres.

/lib: contient les bibliothèques nécessaires aux programmes de /bin et /sbin, ainsi que les modules du kernel (typiquement des pilotes) dans /lib/modules

/proc: liste des informations sur les processus en cours d'exécution et sur le système, par exemple; le contenu de ce répertoire n'existe pas sur le disque, mais est généré par le kernel

/sys, nouveau dans Linux 2.6: va accueillir le contenu de /proc, à l'exception des informations sur les processus

/var; contient des fichiers divers, typiquement des verrous (/var/lock), les IDs des services en cours d'exécution (/var/run), le courrier (/var/mail) ...

/usr, où sont stockés les programmes; on y retrouve des répertoires bin, sbin, et lib. /usr/share contient des fichiers partagés, /usr/share/man contient les pages de manuel, /usr/share/doc contient les autres documentations et /usr/include les headers pour la compilation de programmes; /usr n'est pas nécessaire au démarrage du système, et peut donc être situé sur une partition différente de la racine

/usr/local; joue le même rôle que /usr, à la différence que le système de packages (rpm ou dpkg) n'y touche pas; c'est donc là qu'il faut installer les programmes qu'on compile soi-même

/opt; peut être considéré comme l'équivalent du "Program Files" de Windows dans la manière de l'utiliser; rarement utilisé, on peut par exemple y extraire une archive prête à l'emploi, comme c'est pour le cas pou le JDK Java

Les principaux périphériques

Les fichiers de périphériques sont regroupés dans /dev. Les plus importants sont:

- hda, hdb, hdc, hdd, les périphériques ide (maître et esclave au premier et second contrôleur. On leur ajoute un chiffre de 1 à 4 pour désigner

leurs partitions primaires, et supérieur ou égal à 5 pour leurs partitions logiques (hda1, hdb5 ...) Le partitionnement d'un disque se fait par la commande fdisk nom_du_disque. Les commandes cfdisk ou parted remplissent la même tâche mais de manière plus évoluée.

- sda, sdb, ..., les disques SCSI. Les disques USB sont mappés sur des périphériques SCSI
- scd0, scd1 ..., les lecteurs CD SCSI. Sous Linux 2.4, on avait l'habitude d'émuler des lecteurs SCSI avec les graveurs pour pouvoir graver, et on y accédait via les périphériques SCSI. Ces périphériques sont aussi appelés sr0, sr1 ...
- lp0, le port parallèle
- ttyS0, ttyS1, les ports séries
- usb/lp0, la première imprimante USB
- psaux, la souris PS/2 sur Linux 2.4
- input/mice, les souris USB. Sur Linux 2.6, les souris PS/2 sont accessibles également sur ce périphérique et psaux est obsolète
- tty0, tty1 ..., les consoles virtuelles
- dans snd, on retrouve les périphériques liés à la carte son via alsa. Les anciens pilotes (OSS) utilisaient les fichiers dsp, mixer, midi
- fd0, fd1, les lecteurs de disquettes. Pour les formater, on peut utiliser les commandes fdformat, superformat ou mformat.
- null, un fichier toujours vide quand on le lit, et où tout ce qui est écrit est perdu
- zero, un fichier contenant une infinité d'octets nuls pouvant être lus
- console, tty, stdin, stdout, stderr, qui sont les fichiers standards du processus courant ainsi que le terminal auquel il se rattache
- kmsg, où on peut lire les messages du kernel au fur et à mesure de leur arrivée
- mem, une représentation de la RAM. Si on le lit, on peut y retrouver des chaînes de caractères contenues dans les programmes en cours d'exécution
- kmem, une représentation du kernel en mémoire

Les processus

Un processus est simplement un programme en cours d'exécution, une suite d'instructions exécutées séquentiellement. Un processus peut cependant être composé de plusieurs suites d'instructions exécutées en parallèle: les threads. Chaque processus est isolé des autres et limité par les droits accordés à celui qui l'a lancé. Il peut recevoir au démarrage des indications par l'intermédiaire de l'environnement et de la ligne de commande. Pour communiquer avec d'autres processus et

l'extérieur, un processus dispose des fichiers et des IPC (Inter Process Communication): mémoires partagées, sémaphores et signaux. Par défaut, un processus a toujours 3 fichiers ouverts: l'entrée standard, typiquement le clavier, la sortie standard, typiquement l'écran, et la sortie d'erreur, qui correspond généralement à la sortie standard. Pour terminer, chaque processus a un "répertoire courant" qui lui sert de base quand il veut référencer un fichier par un chemin relatif.

Le shell et les commandes

Lors de son lancement, un programme peut recevoir des arguments sur la ligne de commande. Ces arguments sont simplement des chaînes de caractères, dont la première (numéro 0) est le nom du programme. Le shell par défaut sur la plupart des systèmes Linux est bash. Lorsque l'on tape une commande, chaque mot séparé par un espace correspond à un argument. Si on désire mettre un espace dans un argument, il faut le précéder d'un backslash ("\"), ou entourer cet argument de guillemets ou d'apostrophes.

Conventionnellement, les arguments (on parle dans ce cas d'options) ont la forme d'une lettre précédée d'un tiret, et éventuellement suivie par un autre argument. Plusieurs options courtes n'ayant pas d'argument complémentaire peuvent être regroupées derrière un même tiret Le système GNU a également introduit des options longues (plus d'une lettre), introduites par deux tirets. Quelques exemples:

- **`ls -l`**
- **`tar -x -v -z -f fichier.tgz`**
- **`tar -xvzf fichier.tgz`**
- **`tar --extract --verbose --gunzip --file=fichier.tgz`**
- **`tar -xvzf "fichier avec espace.tgz"`**

Quant à l'environnement, il est constitué de valeurs du type **`NOM=valeur`**. Plusieurs variables d'environnements sont standardisées; par exemple, la valeur de LANG peut être utilisée par le programme pour déterminer dans quelle langue il doit réaliser ses affichages, PATH indique au shell où trouver les exécutables dont le chemin complet n'a pas été indiqué[1] et DISPLAY indique le serveur X[2] que les applications graphiques doivent

1 Par défaut, le répertoire courant n'est pas dans le PATH. Si on désire l'y ajouter, il est préférable de le metre à la fin pour éviter toute mauvaise surprise. Pour exécuter un programme se trouvant dans le répertoire courant, on le précède du chemin "./".

2 Sous UNIX, le protocole X est utilisé par les programmes graphiques pour communiquer avec l'utilisateur. Un serveur X contrôle une console (clavier, écran, souris) et peut accepter les connexions de programmes s'exécutant sur la même machine ou même sur une autre machine. L'application transmet au serveur X les instructions pour dessiner son interface, et le serveur X transmet à l'applications les pressions sur les touches du clavier, et les mouvements et clics de souris. Le serveur X traditionnel sous Linux est Xfree86, maintenant remplacé sur la plupart des

utiliser.

En référence au chapitre précédent, le shell a comme tout processus un répertoire courant. Ce répertoire courant deviendra le répertoire courant de tous les processus lancés par le shell.

Premières commandes

La commande **pwd** (Print Working Directory) affiche le répertoire courant du shell.

La commande **ls** (List Sorted) permet de lister le contenu d'un répertoire ou un fichier. Elle accepte les option -l pour produire un affichage détaillé, -a pour lister les fichiers visibles et cachés[1] y-compris "." et "..", -A, qui affiche la même chose que -a sauf "." et "..", et -d, qui affiche le répertoire lui-même au lieu des fichiers qu'il contient. Elle accepte également un ou plusieurs arguments indiquant les fichiers ou répertoires à lister. A défaut de ces arguments, le répertoire courant est listé.

Exemples:

- **ls # liste le contenu du répertoire courant**
- **ls -l / # liste les fichiers de la racine avec leurs détails**
- **ls -ld / # affiche les détails du répertoire racine**
- **ls -a ~ # liste tous les fichiers du répertoire personnel de l'utilisateur (bash remplace "~" par le répertoire personnel)**

La commande **cd** permet de changer le répertoire courant du shell. C'est une commande intégrée au shell: aucun nouveau processus n'est lancé. Elle prend en argument le nouveau répertoire courant, ou "-" pour retourner au répertoire courant précédent.

Pour créer un répertoire, on utilise la commande mkdir **nom_du_répertoire**, et **rmdir nom_du_répertoire** pour le retirer (il doit être vide. Pour effacer un fichier, on utilise la commande **rm fichier**, qui peut être utilisée avec l'option **-r** et un répertoire pour effacer récursivement le répertoire. **mv source destination** déplace un fichier. Cela correspond à un renommage si on déplace le fichier avec un autre nom dans le même répertoire. **cp source destination** copie le fichier et accepte entre autres les options **-r** pour copier récursivement les répertoires et **-a** pour effectuer des copies conformes récursivement (les fichiers copiés on le même propriétaire et les mêmes droits d'accès que les originaux).

L'environnement peut être manipulé par les commandes **export** et **unset**. Ainsi, **export VARIABLE=valeur** crée une variable d'environnement "VARIABLE" avec la valeur "valeur", et **unset VARIABLE** efface cette variable d'environnement. L'environnement est automatiquement aux processus

---

distributions par le serveur du consortium X, appelé X.org, que Debian utilisera également dans sa prochaine version (3.2 ou 4.0).

1 Un fichier caché est un fichier dont le nom commence par un point (".").

lancés par le shell. On peut aussi passer une variable d'environnement à un programme seulement en le précédant de **VARIABLE=valeur** sur la ligne de commande. La commande **env** permet d'afficher l'environnement courant.

Plus d'informations sur les fichiers

La commande **ls -**l produit un affichage de ce genre, qui appelle à de nouveaux commentaires:

```
-rw-r--r--   1 julien julien    14058 2004-02-12 14:06 fichier
```

Le premier caractère indique le type de fichier: "-" indique un fichier normal,"d" un répertoire, "b" un périphérique bloc, "c" un périphérique caractère, "s" une socket, "p" un pipe et "l" un lien symbolique. Les 9 caractères suivants, répartis en 3 groupes de 3, indiquent les droits d'accès pour, dans l'ordre le propriétaire du fichier, les membres du groupe du fichier et les autres utilisateurs. Les 3 caractères de chacun de ces groupes indiquent dans l'ordre les droits de lecture, d'écriture et d'exécution (r,w,x). Suivent ensuite le nombre de liens du fichier, le nom du propriétaire du fichier, le groupe du fichier, sa taille en octets, la date de dernière modification et le nom du fichier. S'il s'agit d'un lien symbolique, sa cible suit le nom de fichier.

Le nombre de liens indique le nombre de fois qu'un fichier apparaît dans l'arborescence. Cela signifie que plusieurs endroits de l'arborescence pointent vers un même fichier sur le disque. On crée un tel lien avec la commande **ln source destination**. L'effacement d'un fichier est souvent appelé "unlink". En effet, cette opération consiste à retirer le lien dans l'arborescence, et seulement si le nombre de liens d'un fichier tombe à zéro, l'espace occupé par le fichier sur le disque est marqué comme disponible. Deux restrictions s'appliquent à la commande ln: les deux liens doivent se trouver sur un même système de fichiers, et elle ne peut pas être appliquée à des répertoires, car le répertoire ".." serait ambigu. L'option -s (Symbolic) permet de créer un lien symbolique qui permet de dépasser ces limitations, mais si le fichier d'origine est effacé, le lien symbolique devient "dangling", c'est à dire qu'il ne pointe plus vers aucun fichier. Il est intéressant de noter qu'un répertoire a souvent plusieurs liens. Cela est dû au fait que chacun de ses sous-répertoires possède un répertoire ".." pointant vers lui.

Concernant le propriétaire et le groupe amène la notion d'utilisateurs. Linux est un système multi-utilisateurs. La gestion des utilisateurs sera étudiée plus loin. Pour le moment, il suffit de savoir qu'un utilisateur possède un groupe par défaut et peut être membre d'autres groupes. Lorsqu'un fichier est créé, son propriétaire est l'utilisateur qui l'a créé, et il en va de même pour son groupe. Pour chaque catégorie d'utilisateurs (le propriétaire, les membres du groupe et les autres), trois droits peuvent être définis: lecture, écriture et exécution. Seul l'administrateur (root) peut modifier le propriétaire d'un fichier avec la commande

"chown" (CHange OWNer), dont la syntaxe est chown utilisateur[:groupe] fichier [fichier ...]. Le propriétaire d'un fichier peut seulement en changer le groupe avec la commande chgrp (CHange GRouP): chgrp groupe fichier [fichier ...], et changer les droits d'accès avec la commande **chmod**. La méthode la plus simple pour utiliser chmod est **chmod <A><B><C>**, où il faut remplacer <A> par ceux à qui on veut appliquer les droits: u pour le propriétaire du fichier (User), g pour les membres du Groupe, o pour les autres (Others) et a pour tous (All), <B> par + pour accorder le droit qui suit, - pour le retirer, = pour donner exactement les droits qui suivent, <C> par le droit à accorder ou retirer: r pour la lecture (Read), w pour l'écriture (Write), x pour l'exécution (eXecute).

Le bit d'exécution a une signification différente selon qu'il s'agit d'un fichier normal ou d'un répertoire. Dans le premier cas, il indique que le fichier est un programme et peut être exécuté. Dans le second cas, il indique que l'on peut accéder au répertoire.

Le bit d'écriture s'applique au contenu du fichier, et non aux informations qui y sont raccrochées comme son nom et ses permissions. Ainsi, si on veut effacer un fichier, il faut et il suffit d'avoir les droits d'écriture sur le répertoire qui le contient afin d'en modifier le contenu.

Outre ces trois droits, trois autres bits sont rattachés à chaque fichier: le bit SUID (Set User ID), le bit SGID (Set Group ID) et le sticky bit. Les bits SUID et SGID s'appliquent aux exécutables. Au lieu d'être exécutés avec les droits de celui qui les lance, ils le sont respectivement avec ceux de leur propriétaire et de leur groupe. Le sticky bit s'applique aux répertoires, et raccorde le droit d'effacement au droit d'écriture des fichiers qu'il contient. Ainsi, le sticky bit de /tmp est positionné.

Les systèmes de fichiers

Linux n'utilise pas les systèmes de fichiers FAT ou NTFS, mais ext2, ext3, ReiserFS, JFS et XFS. En effet, FAT et NTFS ne permettent pas de stocker les attributs de fichiers exposés au chapitre précédent. ext2 est le système historique de Linux. ext3 est une évolution d'ext2, auquel a été ajouté un journal afin d'assurer l'intégrité des écritures sur disque. Reiserfs a été développé à partir de zéro, est journalisé et permet entre autres de stocker plusieurs petits fichiers dans un seul bloc. XFS et JFS viennent d'autres systèmes UNIX et sont rarement proposés par défaut.

Les volumes

Linux ne possède qu'une arborescence unique de fichiers. L'accès à d'autres systèmes de fichiers, comme celui d'une disquette, d'un CD ou d'une partition se fait par une opération appelée montage. Il consiste à accrocher une arborescence à un répertoire d'une autre par la commande **mount**. Sa syntaxe est **mount -t filesystem device directory**, où filesystem est le système de fichiers (vfat pour une FAT Windows, ext2, ext3, reiserfs pour les systèmes de Linux, iso9660 pour les CDs, udf pour

les DVD). Cette commande doit être tapée en tant que root (voir plus loin: les utilisateurs). Le fichier permettant d'automatiser cela est /etc/fstab, dont voici un exemple. Sur chaque ligne, on trouve dans l'ordre le périphérique à monter, l'endroit où le monter, le filesystem qui se trouve sur ce périphérique, les options de montage et deux nombres pouvant en toute sécurité être mis à zéro. On remarque qu'aucun filesystem n'est monté sur /proc: c'est le kernel qui génère automatiquement le contenu de ce répertoire. Le swap est une partition dont le rôle est d'étendre la RAM réellement disponible. Les options rw et ro indiquent de monter en lecture-écriture ou en lecture seule, l'option noauto indique de ne pas monter le périphérique au démarrage, et user permet à un utilisateur normal de monter le périphérique avec la commande **`mount directory`** ou **`mount device`**.

```
/dev/hda2       /       ext3    defaults        0 1
proc            /proc   proc    defaults        0 0
/dev/hda1       /boot   ext3    defaults        0 1
/dev/hda3       none    swap    sw              0 0
/dev/hdc1       /data   ext3    defaults        0 1
/dev/fd0        /floppy auto    rw,user,noauto  0 0
/dev/cdrom      /cdrom  auto    ro,user,noauto  0 0
```

Il est ainsi possible d'avoir un système réparti sur plusieurs partitions en toute transparence. On peut démonter un filesystem avec la commande **`umount directory`** ou **`umount device`**, à condition qu'il ne soit pas utilisé (aucun de ses fichiers ouvert, aucun processus n'ayant son répertoire courant dans un de ses répertoires[1].

A propos du formatage et des systèmes de fichiers

Par formatage, on désigne le fait de structurer un média pour permettre d'y lire et écrire (voir le 1er chapitre). Seules les disquettes peuvent nécessiter un formatage; les autres médias (disques durs) sont déjà formatés en usine. On dispose pour cela des commandes fdformat, superformat et mformat (qui crée en plus un système de fichiers FAT). La création du système de fichiers inscrit les informations permettant de stocker et de retrouver des fichiers. Quand le filesystem est créé, le lecteur peut être monté. Pour créer un système de fichier ext2, la commande est **`mke2fs /dev/disque`**, pour ext3, il suffit de rajouter l'option -J , pour reiserfs, la commande est **`mkreiserfs /dev/disque`** et pour une FAT, **`mkdosfs /dev/disque`**.

Les utilisateurs

Linux est un système multi-utilisateurs: il garantit que chaque processus et utilisateur ne puisse pas être influencé par un autre. Les utilisateurs sont recensés dans le fichier /etc/passwd[2], par exemple:

---

1 Le processus coupable d'un tel blocage n'est pas forcément lancé par explicitement par un utilisateur. Un coupable fréquent est fam (File Alteration Monitor), qu'il suffit d'arrêter (**`/etc/init.d/fam stop`**).

2 C'est le système simple de configuration des comptes. Pour être plus complet,

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
operator:x:37:37:Operator:/var:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
Debian-exim:x:102:102::/var/spool/exim4:/bin/false
julien:x:1000:1000:Julien De Bona,,,:/home/julien:/bin/bash
```

On y retrouve sur chaque ligne, séparés par des ":" le nom de l'utilisateur, son mot de passe, ou plutôt un x qui sera expliqué plus loin, le numéro d'utilisateur, le numéro de son groupe par défaut, des informations sur l'utilisateur dont son nom, son répertoire personnel et son shell par défaut. Par sécurité, les mots de passes se retrouvent cryptés dans un autre fichier, /etc/shadow, que seul root peut lire, permettant ainsi de laisser /etc/passwd lisible à tous. Les groupes, eux, sont définis dans le fichier /etc/group, par exemple:

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:
tty:x:5:
disk:x:6:
lp:x:7:julien
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
cdrom:x:24:julien,hal
floppy:x:25:julien,hal
tape:x:26:
sudo:x:27:
audio:x:29:julien,mpd
dip:x:30:julien
```

Dans l'ordre, on trouve le nom de groupe, son mot de passe (rarement

l'authentification se fait en général via PAM (Pluggable Authentication Modules), qui par défaut consulte ce fichier, mais peut consulter un annuaire LDAP ou virtuellement utiliser n'importe quelle méthode pour autoriser un utilisateur à se connecter

utilisé), son numéro et ses membres, séparés par des virgules.

On remarque qu'un utilisateur possède le numéro 0 et appartient au groupe 0. Il s'appelle toujours root et est l'administrateur de l'ordinateur. Il a tous les pouvoirs, y compris celui de lire les fichiers de chaque utilisateur. En règle générale, on n'utilise l'utilisateur root que lorsque c'est nécessaire afin de limiter les effets de fausses manoeuvres.

Commandes utiles

passwd permet à un utilisateur de changer son mot de passe. root peut y ajouter en argument le nom d'un utilisateur pour changer son mot de passe. Le bit SUID de passwd est positionné. En effet, /etc/passwd est interdit d'accès aux utilisateurs normaux pour éviter qu'il soit consulté ou modifié n'importe comment. En utilisant passwd, un utilisateur gagne le droit d'accès au fichier, mais est limité à ce que le programme lui laisse faire. **adduser** permet d'ajouter un utilisateur et **addgroup** un groupe. Invoqués sans arguments, ils sont interactifs, mais ils acceptent également des paramères sur la ligne de commande. **deluser** et **delgroup** réalisent les tâches inverses. Il peut être nécessaire d'effacer le répertoire personnel de l'utilisateur et ses fichiers de mail. Consulter la page de manuel pour les options permettant de le faire. Il existe également les commandes **useradd**, **groupadd**, **userdel** et **groupdel**, qui se comportent un peu différemment. Il est également possible de rajouter des utilisateurs et des groupes en éditant simplement les fichiers et en créant le répertoire personnel sans oublier de le "donner" à l'utilisateur et à son groupe par défaut avec la commande chown.

Scripting basique

Tout d'abord, le shell possède deux caractères jokers pour faire référence à des fichiers: * pour un nombre quelconque (y-compris 0) de caractères et ? Pour exactement un caractère. Ainsi, si un répertoire contient des les fichiers titi, toto tata, toutou et tati, ls ta* affichera tata et tati, tandis que ls t?t? listera titi, toto, tata et tati dans l'ordre alphabétique.

Ensuite, il est possible de rediriger les fichiers par défaut des programmes lancés, et ainsi de lire depuis le un fichier au lieu du clavier, ou d'écrire dans un fichier au lieu du terminal, le fichier pouvant être /dev/null si on n'a pas besoin des affichages du programme. On redirige l'entrée standard en ajoutant < fichier après le programme et ses arguments. Pour la sortie standard, on utilise > fichier (le fichier sera écrasé) ou >> fichier (la sortie sera ajoutée à la fin du fichier. Quant à la sortie d'erreur standard, on utilise 2> fichier ou 2>> fichier pour la rediriger vers un fichier ou 2>&1 pour la rediriger vers la sortie standard.

Plutôt que de rediriger une sortie vers un fichier, on peut la rediriger directement vers l'entrée standard d'un autre processus en faisant suivre le programme d'un "|" et d'une seconde commande.

On peut aussi envoyer sur la ligne de commande d'un processus la sortie d'un autre grâce aux apostrophes inverses (`): Par exemple, si fichier

contient sur chaque ligne le nom d'un fichier que l'on désire effacer, on peut taper **`rm `cat fichier`** pour obtenir l'effet souhaité[1].

On peut également conditionner l'exécution d'un programme à la réussite d'un autre: p1 && p2 lance p2 seulement si p1 a réussi, tandis que p1 || p2 lance p2 si p1 a échoué. p1;p2, en revanche lancera p2 quand p1 sera terminé[2].

De nombreuses commandes ou options anodines deviennent très utiles lorsqu'elles sont utilisées avec ce qui précède. Voici quelques commandes de bases couramment utilisées:

grep expression fichiers affiche les lignes correspondant à l'expression rationnelle passée sur la sortie standard. Si aucun fichier n'est spécifié, grep lit sur l'entrée standard. L'expression rationnelle peut être une simple chaîne de caractères, mais quelque chose de bien plus complexe. Les possibilités sont très riches. L'option -v de grep permet au contraire d'afficher les lignes qui ne correspondent pas à l'expression. Par exemple, pour afficher les sous-répertoires d'un répertoire, on peut utiliser **`ls -l|grep '^d'`** (cette expression correspond à un d en début de ligne).

cut est le complément de grep, qui permet de couper mais dans chaque ligne (par exemple, en extraire le Xème champ. Il est moins utilisé, consulter le manuel pour les détails.

awk est un interpréteur pour un langage de manipulation de chaînes de caractères, pouvant avantageusement remplacer cut et d'autres utilitaires

perl est l'interpréteur pour le langage du même nom. Son étude sort du cadre de cette formation, mais c'est un outil très puissant pour des tâches de scripting et de manipulation de chaînes[3]

cat fichiers concatène les fichiers qui lui sont passés (ou l'entrée standard si aucun fichier n'est spécifié) et les envoie sur la sortie standard. Si on a découpé un fichier en plusieurs petits fichiers pour le mettre sur des disquettes, on peut recoller le tout par un **`cat * > fichier_complet`**.

false échoue toujours, **true** réussit toujours. On peut ainsi s'assurer qu'un commande réussira toujours en tapant **`commande || true`**. Plus généralement, un programme retourne une valeur nulle s'il se termine

---

1 Attention si le fichier lu est trop long: la taille de la ligne de commande est limitée, et le comportement de la commande ne serait pas celui attendu. Une formulation du genre **`cat fichier|while read i; do rm "$i"; done`** résoudrait ce problème pour autant qu'il y ait un fichier par ligne. Noter aussi les guillemets au cas où le fichier contiendrait des espaces.

2 Il s'agit bel et bien des opérateurs logiques "ou" (||) et "et" (&&) avec une exécution conditionnelle

3 Et bien d'autres choses: Perl possède des milliers de modules pour réaliser les tâches les plus diverses, de la manipulation du calendrier de la Comté au templating de sites webs, en passant à l'accès aux bases de données et autres tâches traditionnelles.

avec succès, supérieure à zéro sinon. Cette valeur est choisie par l'auteur du programme: il ne s'agit donc pas de codes indiquant une erreur du genre "commande inconnue", mais de codes pouvant être utilisés dans la logique du script.

more et **less** sont des "pagers" permettant de visualiser un fichier, ou l'entrée standard si aucun fichier n'est passé en argument. Ainsi, si une commande produit une longue sortie, **commande | less** permettra de la consulter à son aise.

echo affiche simplement le texte en argument suivi d'un retour à la ligne. L'option -n n'ajoute pas ce retour à la ligne.

yes écrit à l'infini sur la sortie standard le texte en argument, ou simplement yes s'il n'y en a aucun. Si par exemple un programme demande de nombreux messages de confirmations, **yes|commande** fournira automatiquement les réponses.

dd (dump disk) est généralement utile pour manipuler des périphériques blocs. Il effectue des copies bloc à bloc. Par exemple, on peut faire une copie exacte de disquettes (en conservant par exemple le secteur d'amorçage) par la commande **dd if=/dev/fd0 of=fichier.tmp**.[1] Ensuite, on peut insérer une nouvelle disquette et faire un **dd if=fichier.tmp of=/dev/fd0**. Les arguments if=... et of=... désignent respectivement le fichier à lire (Input File) et à écrire (Output File). Les arguments bs=... (Block Size) et count=... sont également acceptés. Ainsi, pour sauvegarder dans un fichier le premier secteur d'une disquette, il suffit de taper **dd if=/dev/fd0 of=fichier bs=512 count=1**.

Outre les variables d'environnement, il est également possible de définir des variables pour le shell. Il suffit simplement de ne pas les exporter avec la commande **export**. On peut les utiliser en faisant précéder leur nom d'un signe "$". Si on désire éviter un tel comportement, il faut entourer le nom de variable d'apostrophes. La commande read permet de lire sur l'entrée standard (voir la note en page 13).

On peut aussi créer d'autres structures, comme des boucles, des tests de conditions avec la commande test, ... Par exemple, on peut boucler sur des mots avec par exemple: **for arg in argument1 argument2 argument3; do echo $arg; done**. Un coup d'oeil dans le répertoire /etc/init.d et dans la page de manuel de bash permettra de découvrir ces structures.

Le démarrage du système

Après son initialisaton, l'ordinateur recherche un secteur amorçable sur un disque. Quand il l'a trouvé, il l'exécute. Linux possède 2 chargeurs pouvant prendre place dans ce secteur: lilo et grub[2]. Le chargeur va charger le kernel depuis le disque dur, lui passer éventuellement des

---

1 La commande cp fonctionnerait aussi.

2 Le démarrage est une étape très dépendante de l'architecture. lilo et grub sont destinés aux PC; pour un macintosh, on utilisera quik ou yaboot, pour un alpha, on utilisera aboot ou milo

paramètres, et puis le démarrer. Cette opération se déroule en mode réel avec les interruptions définies par le BIOS[1]. Une fois démarré, le kernel installe ses pilotes et ils prennent le relais du BIOS. Ensuite, il recherche le système de fichiers à monter comme racine et lance le programme init qui s'y trouve. Init est initialisé par le fichier /etc/inittab. Il est constitué de lignes au format **identifiant:runlevel:moment:action**. Linux accepte 10 runlevels, qui correspondent à une configuration pour le système. Debian laisse l'administrateur en faire ce qu'il veut, d'autres distributions créent un runlevel pour un système en mode texte, un autre pour un système en mode graphique ... Le runlevel 0 est utilisé pour éteindre l'ordinateur, et le 6 pour le redémarrer. La ligne **l0:0:wait:/etc/init.d/rc 0** indique lorsqu'on passe au runlevel 0 de lancer et atteindre la fin (wait) la commande **/etc/init.d/rc 0**. Traditionnellement, elle va lancer dans l'ordre tous les scripts de /etc/rc0.d commençant par S à l'entrée dans le runlevel, et puis ceux commençant par K à la sortie du runlevel. La ligne **1:2345:respawn:/sbin/getty 38400 tty1**, elle va pour les runlevels 2345 lancer et relancer dès qu'elle est finie la commande qui suit. getty a pour rôle de demander le login de l'utilisateur. Ensuite, getty lance le programme login pour demander et vérifier le mot de passe. Si cela se passe bien, le shell de l'utilisateur est lancé et ses droits limités à ceux de l'utilisateur. Lorsque la session se termine ou que le login échoue, getty redémarre conformément à l'instruction respawn. Dans ce fichier, on voit qu'un getty est lancé sur les consoles virtuelles 1 à 6.

```
# /etc/inittab: init(8) configuration.
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $

# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization script.
# This is run first except when booting in emergency (-b) mode.
si::sysinit:/etc/init.d/rcS
# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes the S and K scripts upon change
# of runlevel.
#
# Runlevel 0 is halt.
# Runlevel 1 is single-user.
# Runlevels 2-5 are multi-user.
# Runlevel 6 is reboot.

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
```

---

1 Le problème des 1024 cylindres vient de là. Les anciens BIOS ne pouvaient pas accéder à plus de 1024 cylindres sur les disques durs (512 MB ou 8.4 GB en général). Dans ces cas, il faut s'assurer que le kernel soit situé dans ces 1024 cylindres en créant une partition en début de disque, et montée sur /boot.

```
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
# Normally not reached, but fallthrough in case of emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

# Action on special keypress (ALT-UpArrow).
#kb::kbrequest:/bin/echo "Keyboard Request--edit /etc/inittab to let this
work."
# What to do when the power fails/returns.
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
# Note that on most Debian systems tty7 is used by the X Window System,
# so if you want to add more getty's go ahead but skip tty7 if you run X.
#
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6

# Example how to put a getty on a serial line (for a terminal)
#
T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100

# Example how to put a getty on a modem line.
#
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3
```

Tout le reste se trouve dans /etc/rc?.d, où on peut voir quels scripts et commandes sont exécutés pour arrêter l'ordinateur ou démarrer les services.

Le ramdisk initial (initrd)

Pour faire face à la variété des configurations matérielles, linux permet de démarrer en deux phases. Il crée un disque virtuel en mémoire et le remplit avec le contenu d'un fichier contenant un filesystem avec les pilotes nécessaires, et en fait sa racine provisoire. Ce fichier est obtenu avec le BIOS. Le kernel charge les pilotes nécessaires pour trouver sa partition racine, puis la monte et en fait sa nouvelle racine. Le processus de démarrage se poursuit alors normalement.

Le système de packages

Pratiquement toutes les distributions possèdent un système de packages pour gérer l'installation des programmes. Chaque programme installable est stocké dans un fichier dont le format est défini par le système de packages. Le système de packages fournit en général une gestion des dépendances (tel package requiert l'installation d'un autre pour fonctionner, ou entre en conflit avec un autre). En général, le système comprend une commande simple permettant l'installation d'un fichier package (dpkg pour Debian, rpm pour RPM), et une commande plus évoluée fonctionnant sur base de dépôts de packages, et permettant en une ligne de télécharger un package avec toutes ses dépendances, et de l'installer, en retirant éventuellement les packages qui sont en conflit avec les packages nouvellement installés. Debian propose apt-get, tandis que le système RPM possède les outils yum, apt-get (qui a été porté depuis Debian), urpmi (Mandrake) ...

Pour information, les différences les plus visibles entre les packages Debian et RPM sont que les packages Debian ont une granularité plus fine et gèrent eux-mêmes leur configuration (d'où les questions posées à leur installation, qui peuvent être réobtenues par la commande **`dpkg-reconfigure package`**).

Les commandes relatives aux packages sous Debian

Si on dispose d'un fichier .deb localement sur l'ordinateur, on peut l'installer par la commande **`dpkg -i fichier.deb`**. Si certaines dépendances ne sont pas installées, un message d'erreur est affiché et il faut installer à part les packages manquants.

La méthode classique est d'utiliser apt-get. La première opération est de

mettre à jour la base de données locales des packages disponibles: **`apt-get update`**. Une fois la base de données mise à jour, on peut installer un package par **`apt-get install nom_du_package`**. Si on ne connaît pas le nom précis du package, on peut le trouver par **`apt-cache search mots clés`**. Lors de la mise à jour de la base de données, des packages peuvent être disponibles en une version plus récente. **`apt-get upgrade`** va mettre à jour le système avec tous ces packages en adoptant une attitude très protectrice: aucun package ne sera installé ni retiré. Si on désire faire une mise à jour d'une version de Debian à la version suivante (par exemple de 3.0 à 3.1), on peut utiliser **`apt-get dist-upgrade`**, qui adoptera une méthode moins défensive.

Pour effacer un package, on peut utiliser la commande **`dpkg -r nom_du_package`** ou **`dpkg -P nom_du_package`**. Comme pour l'installation, un problème de dépendances provoque un message d'erreur. La différence entre les deux commandes est que l'option -P efface en plus les fichiers de configuration et que le système considère ne jamais avoir installé le package.

L'autre méthode de suppression, via apt-get, est **`apt-get remove nom_du package`**, ou **`apt-get remove –purge nom_du_package`**.

Informations sur les packages

On peut lister les packages installés par la commande **`dpkg -l`**, et lister les fichiers d'un package par **`dpkg -L nom_du_package`**.

La documentation

L'essentiel de la documentation en ligne est disponible sous forme de pages de manuel, via la commande man. En général, on l'appelle avec pour seul argument le nom de la page désirée. Les sujets sont répartis dans 9 sections:

1. programmes et commandes
2. appels systèmes (pour la programmation)
3. fonctions de bibliothèques (également pour les développeurs)
4. fichiers spéciaux (notamment les fichiers de /dev)
5. fichiers de configuration (notamment les fichiers de /etc
6. jeux
7. divers
8. commandes d'administration (répertoires sbin)
9. routines du kernel

Si plusieurs sujets ont le même nom, on accède au sujet désiré en ajoutant le numéro de sa section avant son nom. Par exemple, on accède à la documentation de man par la commande "man man", la page de la commande passwd par **`man 1 passwd`** et la page du fichier /etc/passwd par **`man 5 passwd`**. Les pages de man sont un simple fichier avec un début et une fin. Il est visualisé avec un "pager", la plupart du temps less (on fait défiler avec les touches haut/bas et on quitte avec "q") ou more (on fait défiler vers le bas d'une ligne avec "enter" et d'une page avec la barre d'espace, et on quitte avec q ou à la fin du document.

Un autre système d'aide est également disponible: info. Les documents info sont principalement disponibles pour les programmes du projet GNU (soit les commandes de base) et ont une structure d'arborescence. La commande **`info info`** donnera les détails sur la visualisation des documents info.

Finalement, le répertoire /usr/share/doc contient diverses documentations. Souvent, ce ne sont que les README des programmes, mais on peut aussi y trouver des documents plus complets en HTML, texte simple ou PDF, et des exemples de configuration.

Pour une documentation plus pédagogique, Internet fournit plus de ressources, dont voici les principales:

- http://www.google.com/linux: page de recherches relatives à Linux, très pratique pour avoir des explications à propos d'un message d'erreur par exemple.
- http://www.tldp.org: The Linux Documentation Project regroupe de

nombreux guides relatifs à Linux, dont les fameux HOWTOs. Ils ciblent à chaque fois un domaine précis, relatif au système lui-même ou à une application particulière.

- http://www.lea-linux.org: un site en français avec des fiches pratiques, des forums et une liste de logiciels triés par rubrique.