Formation LLT
2epartie

Renaud Michel

Mars 2002

Table des matières

1 Système de fichier

1.1 Tout est fichier

Une des bases du fonctionnement d'un système Unix est que tout dans l'ordinateur est associé à un fichier et que tous ces fichiers sont localisés dans une arborescence unique, dont l'élément racine (root) est sobrement noté /.

On y trouve bien sur les fichiers normaux, ceux qui sont enregistrés dans les secteurs des disques durs.
On peut créer un fichier vide à l'aide, par exemple, de la commande touch.
Les répertoires sont aussi des fichiers, le contenu d'un fichier répertoire est en fait une liste d'autres fichiers, ce sont les répertoires qui permettent de construire l'arborescence.
Les répertoires se créent à l'aide de la commande mkdir.
Il faut noter que, sur les types de partitions le permettant, un même fichier "physique" (l'ensemble de secteurs sur le disque dur) peut être associé à plusieurs noms dans l'arborescence, appelés "liens physiques". Un fichier n'est effectivement supprimé du disque que lorsque plus aucun lien physique ne pointe vers lui.
Les liens physiques supplémentaires sont créés à l'aide de la commande ln, un lien physique ne peut pointer que vers un fichier se trouvant sur la même partition que lui.

On trouve également des fichiers dit "de périphériques", placés dans /dev par convention (mais ce n'est pas obligatoire), et qui permettent, à l'aide des méthodes habituelles de travail sur des fichiers, d'accéder aux périphériques de la machine (disques durs, carte graphique, lecteurs divers, cartes d'acquisitions en tous genres, ports d'entrée sortie réels ou virtuels, ...).
Ces fichiers sont créés à l'aide de la commande mknod.

Il existe encore d'autres types de fichiers plus exotiques, les tubes nommés. Ce type de fichier est lié aux fonctionnalités de redirections d'entrées sorties particulièrement utiles en ligne de commande (à l'aide des caractère | < >) mais également utilisés en interne (caché à l'utilisateur, par opposition aux redirections du shell qui elles sont insérées spécifiquement par l'utilisateur) par beaucoup de programmes.
Ces fichiers sont créés à l'aide de la commande mkfifo.

Il y a enfin des fichiers "virtuels" qui permettent d'obtenir des informations sur le système (charge CPU, mémoire libre, uptime, ...).
Ces fichiers sont créés automatiquement par le noyau et différents drivers et sont localisés, par convention, dans /proc.

Il existe un type particulier de fichiers, appelés "liens symboliques", qui ne sont en fait que des "raccourcis" vers d'autres fichier de l'arborescence (quel qu'il soit). Un lien symbolique se comporte exactement comme le fichier vers lequel il pointe, ses permissions sont celles du fichier pointé, lorsqu'on le visualise ou l'édite c'est en fait au contenu du fichier pointé qu'on accède, par contre la suppression d'un lien symbolique ne supprime que le lien, le fichier pointé et son contenu sont toujours là.
Les liens symboliques sont créés (sur les partitions les supportant) à l'aide de la commande ln -s. les liens symboliques, contrairement aux liens physiques, ne sont pas limités par les partitions.

Il y a une petite subtilité concernant les noms de fichier, un fichier dont le nom commence par un point est considéré comme caché (il faut par exemple utiliser l'option -a de la commande ls pour les afficher)

1.2 Partitions

1.2.1 Convention de nommage, ancienne et nouvelle.

Comme écrit à la section 1.1, les disques durs et les partitions qui y sont définies sont représentés par des fichiers localisés dans /dev.

L'ancienne convention de nommage, encore systématiquement utilisées, est de placer tous les fichiers dans /dev.
Les disques/partitions/lecteurs se trouvant sur interface (E)IDE sont préfixés par hd, ceux sur interface SCSI sont préfixés par sd. Les différents disques lecteurs sont ensuite numérotés à l'aide des lettres de l'alphabet, pour l'interface (E)IDE les lettres a et b représentent respectivement les disques maître et esclave de la première nappe IDE, c et d ceux de la deuxième nappe et ainsi de suite. Les disques sur interface SCSI sont également représentés par des lettres qui sont ici associées au numéro du périphérique sur l'interface SCSI, ainsi a est le disque numéro 1 de la première interface SCSI et ainsi de suite.

La nouvelle convention de nommage utilise, quand à elle, beaucoup plus de sous-répertoires, ainsi les périphériques (E)IDE sont dans /dev/ide et les périphériques SCSI dans /dev/scsi, cela ne s'arrête pas là avec des sous-répertoires host$n$[1], des sous-sous-répertoires bus$n$, des sous-sous-sous-répertoires target$n$ et des sous^4-répertoires lun$n$ qui contiennent enfin les fichiers de périphériques. cette convention est la même pour les deux types d'interface.

Il est de coutume de créer des liens symboliques tels que /dev/cdrom, /dev/mouse, /dev/modem ou /dev/scanner vers les périphériques correspondants, pour une plus grande facilité d'utilisation.

1.2.2 Types de partitions.

Linux supporte un très grand nombre de types de partitions d'origine, et encore plus avec des patchs supplémentaires.

Le plus utilisé actuellement est le type ext2fs (second extended filesystem), qui supporte toutes les possibilités d'un système Unix (permissions, liens physiques et symboliques, fichiers spéciaux). Ce système de fichier est utilisé depuis longtemps, stable, bien connu et supporté par plusieurs autres OS, nativement ou à l'aide d'extensions. Ce système est une évolution de l'ext1 qui n'est plus utilisé.

Le fs [v]fat{12,16,32} habituel du monde windows est bien évidement supporté, mais ce système de fichier n'apporte pas toutes les possibilités nécessaires à un système Unix.
Le système ntfs, utilisé par WindowsNT est également supporté, mais le support en écriture est encore considéré comme dangereux.

Le système utilisé sur les CD-ROM, l'iso9660, est supporté, ainsi que les extensions joliet propres à MS ainsi que les extensions RockRidge qui ajoutent à l'iso9660 les possibilités des fs Unix (noms de fichiers longs, permissions, ...).
Le système UFS utilisé pour les DVD et les CD-RW est quand à lui supporté en lecture, le support en écriture n'est malheureusement pas encore fonctionnel.

Depuis quelque temps on voit également l'émergence sous Linux de systèmes de fichiers journalisés, ces systèmes assurent la cohérence du fs, permettant ainsi une réduction considérable

[1]les n sont des numéros représentants les périphériques successifs.

du temps de vérification en cas d'arrêt inapproprié de la machine (ils n'assurent cependant pas la pérennité des données contenues dans les fichiers). On dénombre les systèmes suivants

ReiserFS le premier à avoir été intégré dans le noyau Linux.

Ext3FS c'est une évolution de l'ext2, avec lequel il est entièrement compatible, les partitions ext3 étant simplement des ext2 auxquelles on a ajouté un journal. Cela en fait un système de choix pour la migration simple d'un système Linux existant basé sur ext2 (comme cela est commun depuis plusieurs années). De plus il supporte plus de modes de journalisation que Reiserfs.

JFS ce système est l'oeuvre d'IBM, qui s'est récemment chargé de le porter sous Linux. Ce système, bien que nouveau sous Linux, à déjà été éprouvé sur les machine Unix du constructeur.

XFS produit de SGI, ce système est également bien connu des professionnels.

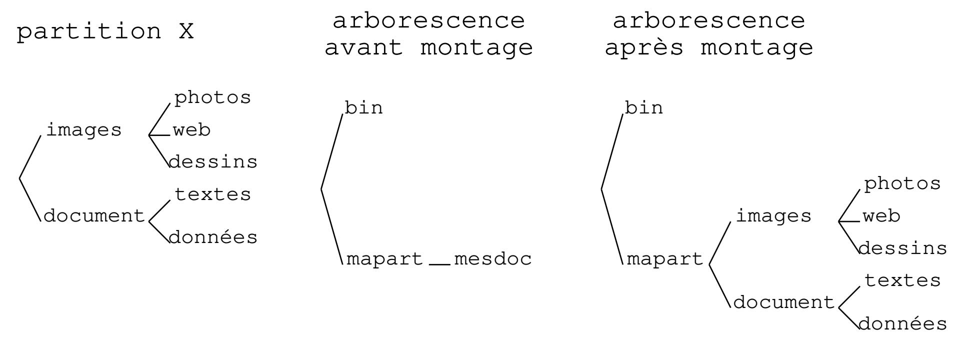
Linux supporte également les partitions réseau, telles que NFS ou SMB.

On peut également citer des systèmes de fichiers virtuels, tel procfs déjà évoqué à la section 1.1 ou encore devfs qui est utilisé pour créer et supprimer automatiquement les fichier de périphérique de /dev, rendant ainsi ce répertoire un reflet du matériel de la machine (ou plutôt du matériel effectivement supporté par Linux).

1.2.3 Montage dans le système de fichier.

Comme il a déjà été dit, le système de fichier sous un système Unix est constitué d'une seule arborescence, mais cela n'empêche pas d'utiliser plusieurs partitions, celles-ci sont "montées" dans l'arborescence.

Monter une partition dans un répertoire /mapartition signifie que, une fois le montage réalisé, les fichiers et répertoires qui seront disponibles dans /mapartition seront ceux se trouvant à la racine de la partition montée. Et bien sur si avant le montage le répertoire /mapartition contenait déjà des fichiers, ceux-ci seront inaccessibles jusqu'au "démontage" de la partition.



Le montage d'une partition se réalise grâce à la commande mount, cette commande possède des options générales et d'autres spécifiques à un type de partition, comme le propriétaire et les permissions par défaut pour les systèmes qui ne les supportent pas.

Il faut bien sur qu'il y ait une partition montée en /, la racine de l'arborescence, on l'appelle la partition racine (root). Cela est nécessaire car il faut au moins que les répertoires dans lesquels seront montées les autres partitions soient enregistrés quelque part.

Les partitions nécessaires au bon fonctionnement du système sont montées au démarrage en lisant le contenu du fichier /etc/fstab, le répertoire /etc doit donc se trouver sur la partition racine, et pas une autre partition montée à cet endroit car dans ce cas le système ne trouverait pas la liste des partitions à monter. Il y a d'autres répertoires devant se trouver sur la partitions racine, ceux et les autres sont décrits à la section 1.3.

1.3 Les répertoires standard

Cette section passe en revue les répertoires standards d'un système Unix.

1.3.1 Le répertoire racine /

/boot contient les fichiers nécessaires au chargement du noyau, donc celui-ci en particulier mais également les fichiers utilisés par le chargeur de boot.

/sbin contient les programmes nécessaires au démarrage et au bon fonctionnement du système, en particulier le père de tous les processus, init.

Ce répertoire doit se trouver sur la partition racine.

/bin contient les programmes standards du système n'étant pas nécessaire à son démarrage

/lib contient les bibliothèques standards du système (entre autres la libc) partagées par tous les autres programmes, il contient aussi dans le sous-répertoire modules les parties du noyau ayant été compilées en tant que modules.

/etc contient toute la configuration générale du système, stockée dans un grand nombre de fichiers textes prévus pour être lisibles et éditables avec un simple éditeur de texte. conserver une sauvegarde de ce répertoire est une bonne idée, c'est d'ailleurs tout l'intérêt de tout regrouper ici.

Ce répertoire doit se trouver sur la partition racine.

/dev contient tous les fichiers de périphériques.

Ce répertoire doit se trouver sur la partition racine, sauf si l'on utilisé devfs, il faut alors que le noyau monte celui-ci dès son chargement avec l'option devfs=mount.

/var contient des données versatiles telles que des boîtes e-mail, des files d'impressions ou des logs.

/proc ce répertoire, déjà évoqué, contient un système de fichier virtuel dont les fichiers permettent d'obtenir des information sur le système. Les fichiers contenus dans ce répertoire sont, pour la plupart, des fichiers textes directement consultables.

Il contient en particulier un certain nombre de répertoires numériques, ces répertoires contiennent des informations sur les processus en cours d'exécution sur le système (le nom du répertoire est en fait le PID du processus), tous les processus, sans exception, y sont représentés. On peut donner comme exemples

- **/proc/meminfo** qui fourni des information sur l'utilisation de la mémoire.
- **/proc/cpuinfo** donne toutes les information détectées par le noyau concernant le(s) processeur(s) de la machine.
- **/proc/mounts** donne la liste des partitions montées.
- **/proc/uptime** donne le temps (en secondes) pendant lequel la machine a fonctionné.

/proc/version contient la version du noyau en cours d'exécution.

/proc/kcore permet d'accéder à la totalité de la mémoire, ce fichier a exactement la taille de la mémoire vive embarquée dans la machine.

/proc/sys/kernel/hostname le nom de la machine

Les fichiers de /proc permettent aussi d'interagir avec le noyau, par exemple le fichier /proc/sys/dev/cdrom/autoclose contient 1 ou 0, suivant qu'il faut ou non fermer le chariot du lecteur CD lors d'une tentative de montage de celui-ci, modifier le contenu du fichier changera ce comportement. De même, modifier /proc/sys/kernel/hostname changera le nom sous lequel la machine s'identifie, jusqu'au prochain démarrage. tous ces fichier ne sont bien sur modifiables (s'il le sont, regardez les permissions) que par l'administrateur.

/tmp le répertoire temporaire général du système, la plupart des programmes l'utilisent pour stocker temporairement des données. Les fichiers de ce répertoire on souvent une durée de vie très courte.

C'est une bonne idée que ce répertoire soit une partition indépendante, cela évite qu'un trop grand nombre de fichiers temporaires ne remplissent complètement la partition racine.

/usr ce répertoire, dont la sous-arborescence reflète partiellement celle de /, contient généralement un grand nombre de programmes supplémentaires.

Il est fréquent de créer une partition pour ce répertoire.

/mnt est destiné à contenir les points de montage de partitions ne s'insérant pas dans l'arborescence standard, comme par exemple des supports amovibles (disquettes, CD-ROMs, ...), des partitions d'un autre système (au hasard, windows) ou des partitions partagées en réseau.

/root le répertoire home de l'administrateur

Ce répertoire doit se trouver sur la partition racine.

/home contient les répertoires personnels des utilisateurs du système.

C'est une bonne idée de créer une partition pour recevoir ce répertoire, cela facilite une réinstallation éventuelle du système sans devoir se soucier des données des utilisateurs, il suffit de ne pas toucher à cette partition.

/opt ce répertoire est utilisé pour contenir des programmes qui ne sont pas prévus pour s'intégrer dans l'arborescence standard, comme certains jeux ou programmes commerciaux.

comme pour /usr, il peut être intéressant de créer une partition pour ce répertoire.

1.3.2 Le répertoire /usr

/usr/bin contient les exécutables des programmes installés dans l'arborescence.

/usr/sbin contient les programmes réservés à l'administrateur.

/usr/games ce répertoire, assez peu utilisé, est destiné à recevoir les exécutables de jeux.

/usr/man contient les pages de manuel.

/usr/info comme /usr/man mais pour les pages info.

/usr/share ce répertoire est destiné à accueillir tous les fichiers, nécessaires ou superflus, utilisés par les programmes de /usr/bin

/usr/doc et /usr/share/doc comme leur nom l'indique, ils sont destinés à contenir la documentation des programmes

/usr/lib contient les bibliothèques non nécessaires au bon fonctionnement du système de base (i.e. les programmes dans /bin et /sbin).

/usr/src est destiné à contenir les sources des programmes, contient souvent les sources du noyau dans le sous-répertoire linux.

/usr/include contient les fichiers nécessaires à la compilation de programmes à partir des sources

/usr/local identique à /usr mais pour des programmes de moindre importance ou dont les données pourraient interagir avec des programmes déjà installés. Ce répertoire est fréquemment utilisé pour installer des programmes compilés "maison" et n'étant pas fournis avec la distribution de base.

Il arrive que, à l'instar de /usr, on associe une partition à /usr/local, cela n'est cependant utile que sur des systèmes demandant beaucoup de rigueur et d'organisation. Ce répertoire trouve en fait tout son intérêt dans de grands systèmes ou les répertoires sont partagés en réseau.

/usr/X11R6 ce répertoire, à l'arborescence semblable à /usr, est destiné à contenir tout ce qui est relié à l'interface graphique X-Window.

2 Organisation des programmes

Dans un système Unix, les programmes ne sont pas installés chacun dans un répertoire propre, mais leurs fichiers sont distribués dans les répertoires standards de l'arborescence telle qu'elle a été décrite à la section 1.3.

Tous les exécutables sont placés dans les différents répertoires bin et sbin (et quelques logiciels de divertissement dans les répertoires games de /usr et /usr/local). Cette organisation présente un gros avantage, il suffit de placer ces quelques répertoires dans la variable $PATH pour pouvoir invoquer les programmes simplement par leur nom, sans devoir préciser leur chemin complet.

Si le programme fournit une documentation sous format de page de manuel (la forme la plus classique sous Unix) ou de page info, ces fichiers seront installés dans les répertoires correspondants de /usr, /usr/local ou /usr/X11R6, de manière à ce que man ou info puissent les trouver.

La documentation supplémentaire, comme les classiques fichiers README ou Changelog, seront placés dans les répertoires doc.

Si le programme a besoin de fichiers supplémentaires (des données, des images pour son interface graphique, ...), ceux-ci seront placés dans un sous-répertoire portant le nom du programme (et éventuellement son numéro de version) dans les répertoires share. Il peut aussi se trouver dans le répertoire share des sous-répertoires partagés par plusieurs programmes ou à vocation plus générale, comme /usr/share/icons.

Si le programme à besoin d'une configuration globale celle-ci devra aller dans /etc si c'est un programme important pour le système (comme un serveur), ou sinon dans le répertoire du programme dans share. S'il faut enregistrer les paramètres des utilisateurs, cela ne peut se faire que dans le répertoire de l'utilisateur concerné, généralement dans un fichier ou un répertoire caché (voir section 1.1).

3 Gestion des utilisateurs, droits d'accès

Les systèmes Unix ont été conçus pour être des systèmes multi-utilisateurs (plusieurs utilisateurs peuvent utiliser simultanément le système), il fallait donc définir comment ces différents utilisateurs allaient interagir entre eux, que ce soit au niveau des programmes ou des fichiers.

À chaque utilisateur est associé un nombre, unique sur le système, appelé UID (User ID).

Parmi tous les utilisateurs du système, il y en a un à part, celui ayant l'UID 0, généralement appelé root, c'est l'administrateur du système. Root a tous les droits sur le système, et s'il y en a qu'il ne possède pas il peut se les octroyer. On comprend donc que seules les opérations nécessitant les permissions d'administrateur doivent être faites en tant que root, toute autre opération "normale" doit se faire avec un ID d'utilisateur non privilégié, même si l'on est root sur sa machine.

Les programmes exécutés sont associés au nom de l'utilisateur qui les a démarrés (il y a cependant des subtilités) et seul le propriétaire (et l'administrateur, qui peut tout faire) du processus peut le modifier.

On utilise également la notion de groupe, un groupe peut contenir un nombre quelconque d'utilisateurs, tout utilisateur doit être au moins membre d'un groupe (son groupe par défaut) et peut appartenir à autant de groupes que l'administrateur voudra. Les groupes sont repérés par un GID (Groupe ID) auquel est associé un nom de groupe.
La commande groups vous donne la liste de tous les groupes auxquels vous appartenez.

3.1 Les permissions des fichiers

Fort de ce qui précède, on comprend facilement qu'à chaque fichier va être associé un UID et un GID représentant le propriétaire du fichier et le groupe auquel le fichier appartient.

Dans le standard Unix, un certain nombre d'attributs sont associés aux fichiers, ces attributs sont des bits, indiquant si le fichier a cet attribut ou non.

– Il y a d'abord les permissions du fichier, elles vont par trois et représentent les permissions en lecture, écriture et exécution, il y en a trois sets associés au propriétaire, au groupe et à tous les autres. On peut ainsi accorder à tous les membres du groupe auquel appartient le fichier les droits de lire et d'écrire dans le fichier tout en refusant ce droit aux personnes n'appartenant pas à ce groupe.

  exemple si un fichier a les permissions rwxr—–, les trois premiers caractères représentent les permissions du propriétaire, il a les droits de lecture (r) et d'écriture (w) et d'exécution (x), les trois suivants sont pour les membres du groupe du fichier, ceux-ci n'ont que les droits de lecture (r–), enfin les autres n'ont aucun droit sur le fichier (—).

  Lorsque vous créez un fichier (quelle que soit la méthode) vous ne pouvez l'associer qu'à un groupe dont vous êtes membre.

Dans le cas des répertoires, les trois bits x ne représentent pas le droit d'exécution (sans objet pour ce type de fichier) mais le droit de lister le contenu du répertoire. Si le fichier de l'exemple précédent est un répertoire alors les membres du groupe ont le droit de lecture mais pas de listage, ils peuvent donc accéder aux fichiers contenus dans le répertoire (suivant leurs propres permissions) à condition de connaître leur nom ! Il est rarement utile de donner à un répertoire les droits en lecture sans donner les droits de listage.

- Il y a d'autres attributs plus étranges, appelés SUID et SGID, qui sont utilisés pour les fichiers exécutables (binaires ou scripts possédant l'attribut x), ces deux attributs indiquent que lors de l'exécution du fichier, celui-ci possédera les privilèges de son propriétaire (SUID) ou groupe (SGID) plutôt que ceux de celui qui l'a exécuté.
  Ce mécanisme est souvent utilisé pour que certains programmes puissent avoir des privilèges root sans devoir accorder ceux-ci aux utilisateurs.
- Les autres types de fichier (tubes nommés, fichiers périphériques, liens symboliques) sont également repérés à l'aide d'attributs de fichier.

Note : les attributs de fichiers sont différents suivant le système de fichier, il est préférable pour le bon fonctionnement de Linux d'utiliser un système de fichier compatible Unix (comme ext2), le système de fichier fat du monde windows est par exemple un très mauvais choix (à beaucoup de points de vue).

4 Système de packages

4.1 Qu'est-ce qu'un package

Un package est une archive contenant des données et/ou des programmes ainsi que les informations nécessaires à une installation correcte de ceux-ci sur le système. Un package est constitué d'un et un seul fichier, ce n'est pas un exécutable, il est pris en charge par un programme dédié, le gestionnaire de package.

Il existe plusieurs types différents de package

RPM pour Redhat Package Manager, c'est le type de package le plus utilisé sous Linux, ces packages gèrent les dépendances et les scripts d'installation/désinstallation.

DEB le type de package utilisé par la distribution Debian et ses dérivés, il gère les scripts et les dépendances de façon plus fine que RPM.

tar.gz utilisé par les distributions du type slackware, il s'agit d'une simple archive contenant également les scripts d'installation.

pkg cette extension peut désigner plusieurs types de package différents utilisés par des Unix propriétaires comme solaris ou QNX.

Il faut noter que les différentes saveurs libres de BSD utilisent un système différent, appelé ports, ou les programmes sont systématiquement recompilés.

4.2 Les dépendances

La gestion des dépendances est un des gros avantages des systèmes de packages.

Chaque package contient une liste des fonctionnalités qu'il fournit, cela commence par le nom du package lui-même, cela peut également être un programme particulier, une bibliothèque générique ou dans une version plus particulière ou encore une fonctionnalité plus "diffuse" telle que "serveur ftp" pour apache ou "gestionnaire de téléchargement" pour wget.

Ensuite les packages vont contenir une liste des fonctionnalités qui leurs sont nécessaires pour être fonctionnels[2].

Lors de l'installation, le gestionnaire de packages va vérifier que toutes les dépendances sont vérifiées et que l'installation du nouveau package ne va pas écraser des fichiers d'autres packages, sinon il refusera d'installer, il reste possible de forcer l'installation à ses risques et périls.

De même, à la désinstallation d'un package, il vérifie que la suppression de ce(s) package(s) ne gène pas le fonctionnement d'autres packages, une fois encore on peut ne pas tenir compte des dépendances mais cela peut s'avérer assez grave (forcer la désinstallation de la glibc est le parfait exemple de ce qu'il faut faire si on veut foutre en l'air son système).

4.3 Les scripts

Si pour installer une documentation il est suffisant d'en décompresser les fichiers, il n'en va pas forcément de même pour des programmes et encore moins pour des bibliothèques dont l'installation nécessitera d'effectuer des opérations sur le systèmes pour que le contenu du package soit pleinement fonctionnel.

C'est à cela que servent les scripts d'installation/désinstallation qui se présentent sous la forme de simples scripts shell (au standard sh). Ils sont au nombre de quatre

le script de pré-installation il effectue toutes les opérations préparatoires à l'installation.

le script de post-installation s'occupe de toute la configuration post-installation, comme par exemple enregistrer de nouvelles bibliothèques ou générer une configuration en accord avec la machine sur laquelle il est installé.

le script de pré-désinstallation s'occupe des opérations à effectuer avant de désinstaller le package, par exemple s'il s'agit d'un serveur quelconque il faut arrêter celui-ci avant de le désinstaller.

le script de post-désinstallation est chargé des opérations à effectuer après désinstallation.

4.4 Précisions sur les packages

Beaucoup de distributions utilisent le système RPM, l'organisation des packages n'est pas cohérente entre les différentes distributions (par exemple entre RedHat et SuSe), il faut donc être prudent si l'on veut installer un package n'ayant pas été prévu spécifiquement pour sa distribution, il convient alors de vérifier si les dépendances correspondent, si les lieux d'installation sont adéquats et si les scripts sont adaptés au système.
Si les dépendances ne sont pas nommées de la même façon mais sont présentes on peut forcer

[2]pour les package deb ces dépendances sont séparées en dépendances nécessaires, sans lesquelles le contenu du package ne sera pas (complètement) fonctionnel, et en dépendances conseillées, dont la présence sera un plus dans l'utilisation du contenu du package mais qui ne sont pas nécessaires à son bon fonctionnement.

l'installation sans vérification des dépendances (en les ayant d'abord toutes vérifiées).
Si l'emplacement des fichiers est inadéquat, certains packages (suivant la façon dont ils ont été construits) permettent la relocalisation de ses fichiers.
Si les scripts ne sont pas adaptés on peut effectuer l'installation sans exécuter les scripts puis faire toutes les démarches à la main.

Il existe des outils permettant de convertir des packages d'un format vers un autre (par exemple alien qui fait le lien entre les types rpm et deb), ces outils doivent être pris comme des pis-aller car, les différents types ne gérant pas les choses de la même façon (voir carrément pas les mêmes choses), le package converti ne sera jamais aussi bon qu'un package natif.

4.5 Exemple de création d'un package RPM

Nous allons suivre les étapes de la réalisation d'un package RPM simple (une application), l'exemple retenu est le programme ksetiwatch (http ://ksetiwatch.sourceforge.net/), il va de soi qu'avant d'entreprendre la réalisation du package il faut vérifier que toutes les bibliothèques et programmes nécessaires au programme sont bien présents sur le système et que le programme se compile correctement, cela permet également de repérer des options de compilation utiles.

Maintenant que nous savons que notre programme compile, nous pouvons envisager de la packager.

4.5.1 configuration de rpm

Vérifions tout d'abord la configuration de rpm à l'aide de la commande

```
rpm --showrc
```

Cette commande affichera toute la configuration de rpm (la configuration globale et la configuration éventuellement définie dans les fichiers ~/.rpmmacros et ~/.rpmrc) l'élément le plus important est l'endroit ou doivent se trouver les fichiers sur lesquels nous allons travailler, cela est défini par la macro %_topdir, pour y voir plus clair utilisons la commande

```
rpm --showrc | grep _topdir
```

Si cette configuration ne convient pas on peut la modifier en plaçant dans ~/.rpmmacros la ligne

```
%_topdir /home/moi/mespackages
```

Ce nouveau répertoire devra contenir l'arborescence archive, tmp, RPM/BUILD, RPM/RPMS/i[3456]86, RPM/RPMS/k6, RPM/RPMS/noarch, RPM/SOURCES, RPM/SPECS, RPM/SRPMS.

À partir d'ici les répertoires relatifs sont des sous-répertoires de %_topdir.

4.5.2 création du specfile

Plaçons l'archive dans RPM/SOURCES et créons un fichier ksetiwatch.spec dans RPM/SPECS, ce fichier spec va contenir toutes les informations de création du package.

1. Quelques définitions

   On commence par définir quelques macros qui simplifierons l'écriture de la suite du specfile et la mise à jour du package vers une version plus récente

```
%define ver 2.2.5
%define rel 1mdk
%define nam ksetiwatch
%define prefix /usr
```

2. Les informations du package

```
Summary: Monitoring tool for the setiathome client
Name: %{nam}
Version: %ver
Release: %rel
Copyright: GPL
Group: Applications/Scientific
Source: %{nam}-%ver.tar.gz
URL: http://ksetiwatch.sourceforge.net/
Packager: Renaud Michel <renaud.michel@student.ulg.ac.be>
Buildroot: /tmp/ksetiwatch-%ver
%description
This is Ksetiwatch 2.2.5, a monitoring tool and work unit manager for the SETI@home project. It is designed for KDE 2.x and Qt 2.2.x or higher. Ksetiwatch provides the look-and-feel of the popular SETIWatch program for Windows95/98/NT, written by Mark Loukko (http://members.home.net/mloukko/SETI.html).
```

   On voit ici que les macros définies précédemment s'utilisent en les préfixant du signe % et en les entourant de {} s'il y a risque de confusion.

   L'entrée Source spécifie le nom de l' (des) archive(s) ou se trouvent les sources du programme, celle(s)-ci sera décompressée dans RPM/BUILD.

   L'entrée Buildroot indique ou le programme devra être temporairement installé avant de réunir ses fichiers dans le package, on utilise ici un sous-répertoire du répertoire temporaire du système, /tmp.

   Les autres entrées sont auto-explicites.

3. Préparation, configuration et compilation du programme

```
%prep
%setup -n ksetiwatch-%{ver}
%build
./configure --prefix=$RPM_BUILD_ROOT%{prefix}
make all
```

   %prep est une étape chargée de préparer les sources à la compilation (généralement décompresser les archives et appliquer les patchs éventuels), on utilise ici la macro %setup prédéfinie dans rpm.

   %build est l'étape à laquelle le programme va être compilé, on effectue d'abord le classique ./configure en utilisant le répertoire temporaire précédemment défini ainsi qu'un préfixe (ici /usr), puis le make all qui lance la compilation.

4. Installation du programme

```
%install
rm -rf $RPM_BUILD_ROOT
make install
mkdir -p $RPM_BUILD_ROOT%{prefix}/lib/menu
cat > $RPM_BUILD_ROOT%{prefix}/lib/menu/%{nam}.menu <<EOF
?package(%{nam}): \
command="%{prefix}/bin/%{nam}" \
title="%{nam}" \
longtitle="%{nam}" \
needs="x11" \
section="Applications/Sciences" \
icon="%{nam}.xpm"
EOF
```

L'étape %install se charge de... l'installation, une petite subtilité, la création du fichier menu spécifique à la mandrake (que j'utilise), ceci explique également les scripts qui suivent

5. Les scripts

```
%post
if [ -x /usr/bin/update-menus ]; then /usr/bin/update-menus || true ; fi
%postun
if [ -x /usr/bin/update-menus ]; then /usr/bin/update-menus || true ; fi
```

Les scripts du package, ici les scripts de post-installation et de post-désinstallation. Ces deux scripts ont pour but de mettre à jour les menus du système.

6. Nettoyage

```
%clean
rm -rf $RPM_BUILD_ROOT
```

Une fois le package terminé il convient de supprimer le répertoire ou s'est effectué la compilation, les fichiers intermédiaires de compilation peuvent prendre beaucoup de place.
Lors de la mise au point d'un fichier spec il peut être préférable de ne pas mettre cette ligne pour pouvoir reprendre le programme déjà compilé au fil des essais, mais il ne faut pas l'oublier si l'on distribue un package source.

7. Les fichiers

```
%files
/usr
```

Cette section contient la liste de tous les fichiers à inclure dans le package, j'utilise ici un truc un peu dangereux, lui dire de prendre tous les fichiers dans /usr (relativement à Buildroot), c'est dangereux car des fichiers non désirés pourraient avoir été copié là pendant la création du package, je me permet généralement de le faire car je sais être le seul à utiliser ma machine, mais dans le doute il est préférable de regarder quels sont les fichiers effectivement installés par le make install (et ceux créés à l'étape %install) et en fournir la liste complète.

8. Le changelog

```
%changelog
* Fri Jan 25 2002 Renaud Michel <renaud.michel@student.ulg.ac.be>
2.2.5-1mdk - Version 2.2.5 - Added menu file
* Mon Sep 3 2001 Renaud Michel <renaud.michel@student.ulg.ac.be>
2.2.4-1mdk - Version 2.2.4
```

Comme le nom l'indique, notez le format des dates qui doit être respecté.

4.5.3 Création du package

La compilation s'effectue simplement avec

```
rpm -bb ksetiwatch.spec
```

Il est possible de n'effectuer que certaines étapes et de passer des étapes (utile pour la mise au point du specfile), voir à ce sujet la manpage de rpm à la section build.

Un package source se créera simplement avec la commande

```
rpm -bs ksetiwatch.spec
```

Le package compilé se trouve dans le sous-répertoire de RPM/RPMS correspondant à l'architecture pour laquelle il a été compilé. Le package source se trouve dans RPM/SRPMS.