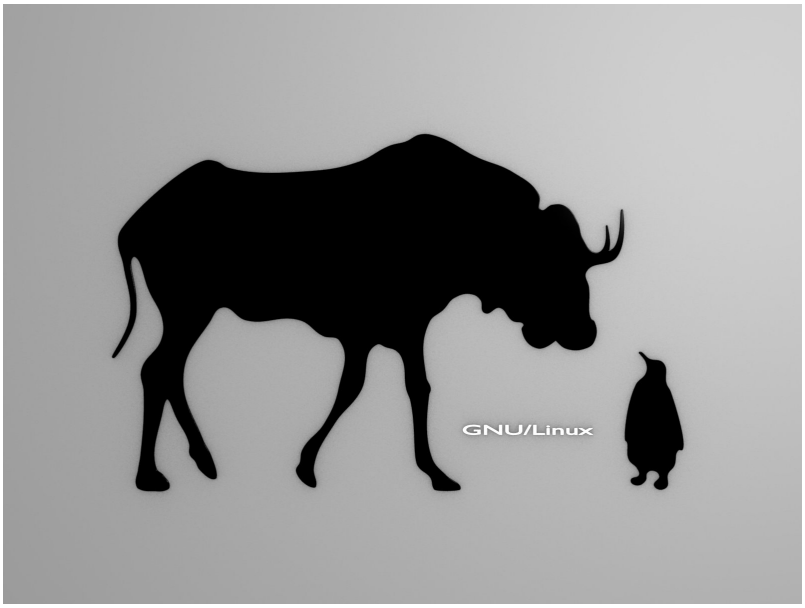
Introduction à Linux

2008-07



Gauthier CATTEAU
Armando MARTINS

Table des matières

Objectifs

Permettre à un utilisateur Linux de se familiariser à l'utilisation et la personnalisation d'un système Linux.

Introduction

Ce support ne saurait être considéré comme un travail achevé ou suffisant pour un auto-apprentissage. Il s'agit de notes organisées destinées à accompagner le cours, ainsi que les activités de TD et TP, d'**Introduction à Linux**. Il n'est certainement pas exempt d'erreurs et j'invite les lecteurs, étudiants et enseignants, à me faire part de leurs remarques pour m'aider dans ce travail sans fin d'amélioration de ce support.

Cette formation ne fera pas de vous un maître incontesté de GNU/Linux , mais vous guidera dans vos premiers pas sous Linux pour que vous sachiez l'installer, ajouter de nouveaux logiciels, et l'utiliser quotidiennement. De plus, cette formation constitue une bonne introduction à une compréhension plus avancée du système d'exploitation et vous donnera les clefs pour aller plus loin, si vous le désirez. . .

Présentation de GNU/Linux

Avant d'entrer dans le vif du sujet une présentation d'Ubuntu Linux de la philosophie en général peut sembler nécessaire. En effet, malgré la médiatisation grandissante du mouvement du Libre, de nombreuses personnes assimilent le Libre à la gratuité. Si vous pensez encore que ces deux notions sont équivalentes, vous verrez qu'à la fin de ce chapitre, votre avis aura changé et vous mesurerez plus précisément les différences entre Libre et propriétaire, ainsi que les enjeux qui en découlent.

A. Qu'est-ce que le mouvement GNU ?

Image 1 : Richard Matthew Stallman

En 1984, Richard Matthew Stallman, chercheur en informatique du MIT quitte son poste et se consacre à l'écriture d'un système d'exploitation Libre du nom de GNU . Il annonce l'année suivante la création de la FSF afin de supporter ce projet.

C'est durant ces années qu'il écrit ce qui deviendra les préceptes du Logiciel Libre. La concrétisation en est la publication en 1989 de la première version de la licence GPL qui sera alors le fondement éthique, juridique et politique du mouvement du Libre.

Complément

Plus d'information sur le mouvement GNU sur le site http://www.gnu.org/ .

B. Qu'est-ce qu'un logiciel libre ?

L'expression « Logiciel Libre » fait référence à la liberté et non pas au prix. Pour comprendre le concept, vous devez penser à la « liberté d'expression », pas à « l'entrée libre ».

L'expression « Logiciel Libre » fait référence à la liberté pour les utilisateurs d'exécuter, de copier, de distribuer, d'étudier, de modifier et d'améliorer le logiciel. Plus précisément, elle fait référence à quatre types de liberté pour l'utilisateur du logiciel :

Définition

Liberté 0

- La liberté d'exécuter le programme, pour tous les usages.

Liberté 1

- La liberté d'étudier le fonctionnement du programme, et de l'adapter à vos besoins. Pour ceci l'accès au code source est une condition requis.

Liberté 2

- La liberté de redistribuer des copies, donc d'aider votre voisin.

Liberté 3

- La liberté d'améliorer le programme et de publier vos améliorations, pour en faire profiter toute la communauté. Pour se faire, l'accès au code source est une condition requise.

Un programme est un Logiciel Libre si les utilisateurs ont toutes ces libertés .

Ainsi, vous êtes Libre de redistribuer des copies, avec ou sans modification, gratuitement ou non, à tout le monde, partout. Être Libre de faire ceci signifie – entre autre – que vous n'avez pas à demander ou à payer pour en avoir la permission. Cela permet de garantir la Liberté – savoir ce qu'il se passe sur votre ordinateur, pouvoir changer de système aisément par l'utilisation de formats ouverts –, l'Égalité – avoir accès à un logiciel à un prix bas ou gratuitement–, et à la Fraternité – avoir le droit de redistribuer légalement à ses amis, ses logiciels.

Vous devez aussi avoir la liberté de faire des modifications et de les utiliser à titre personnel dans votre travail ou vos loisirs, sans en mentionner l'existence.Si vous publiez vos modifications, vous n'êtes pas obligé de prévenir quelqu'un de particulier ou de le faire d'une manière particulière. La liberté d'utiliser un programme est la liberté pour tout type de personne ou d'organisation de l'utiliser pour tout type de système informatique, pour tout type de tâche et sans être obligé de communiquer ultérieurement avec le développeur ou tout autre entité spécifique.

C. Linux, juste le noyau !

«Au sens strict, Linux est le nom du noyau de système d'exploitation libre, multitâche, multiplate-forme et multi-utilisateur de type UNIX créé par Linus Torvalds, souvent désigné comme le noyau Linux.»

Le projet GNU arrive en 1991 avec de très nombreux outils libres, mais il lui manque un élément central : **le noyau**. Cet élément est essentiel car il gère la mémoire, le microprocesseur, les périphériques comme le clavier, la souris, les disques durs. . .

Image 2 : Linus Torvald

C'est à cette époque qu'un étudiant finlandais, Linus Torvalds, commence à développer un noyau et demande aux personnes intéressées d'y contribuer. La licence GPL a été publiée à la même époque et Linus Torvalds s'est laissé persuader de placer son noyau sous cette dernière.

Le système d'exploitation actuellement connu est donc un assemblage des outils GNU fonctionnant sur un noyau Linux, on parle donc de GNU/Linux avec le slash, « / » pour « GNU sur Linux ».

Définition

GNU/Linux est un système d'exploitation complètement Libre et performant. Il est hautement configurable. Il ne dépend pas d'une multinationale. Il est supporté par une grande communauté d'utilisateurs souvent prêts à vous aider. Quelque soit votre domaine de compétence, vous pouvez participer à l'amélioration de GNU/Linux pour que ce dernier évolue dans votre intérêt. Ce n'est pas un simple logiciel gratuit, mais un Logiciel Libre. Ce qui garantit qu'il restera accessible et gratuit pour tous, sans discrimination.

D. Qu'est ce qu'une distribution ?

En réalité, si on vous livrait le noyau Linux seul, accompagné des outils GNU de base, vous seriez bien avancé : pas d'interface graphique, juste quelques commandes, bref, votre système d'exploitation serait inexploitable, un comble,non ?

C'est pour cela qu'existe des distributions Linux qui contiennent le noyau Linux, les outils GNU, plus un ensemble de logiciels qu'elles ont choisi de supporter. Ceux-ci sont testés et compilés pour vous. La plupart d'entre elles contiennent un système d'installation de logiciel simplifié qui leur est – malheureusement – propre. Vous avez déjà dû voir qu'il existe de très nombreuses distributions : Mandriva, Red Hat Fedora, Debian, Gentoo, OpenSuse, Ubuntu ...

Alors pourquoi autant de distributions, me direz-vous ? En fait, chaque distribution a sa cible : certaines sont orientées sur la facilité d'utilisation,d'autres sont pour les véritables « geeks », certaines sont spécialisées pour l'utilisation dans le domaine scolaire ou musical, d'autres encore se veulent très légères et fonctionner sur des PC antédiluviens. . . Vous voyez qu'il peut y avoir autant de distributions que de cas d'utilisation !

E. À quoi sert vraiment un système d'exploitation ?

Il exploite ! Oui, mais « qui » allez-vous me dire ? En fait, il s'agit plutôt de « quoi » : l'OS exploite votre matériel.

Essayons d'imaginer le contraire : si le système d'exploitation n'existait pas, tous les logiciels devraient être conçus pour tous les matériels existants.

C'est à dire que chaque programmeur devrait prendre en compte l'ensemble du matériel (carte graphique, type de mémoire RAM, disque dur, processeur. . .) ainsi que tous les périphériques (clavier, souris, écran, imprimante. . .) existant ou ayant existé.

De plus, à la sortie d'un nouveau matériel, ce qui arrive par centaines quotidiennement, il faudrait alors le prendre en compte et sortir une nouvelle version de chaque logiciel !

J'ajouterai également que cela prendrait une place en mémoire non négligeable et énormément de temps puisque ce travail serait dupliqué pour chaque logiciel!

C'est donc la fonction principale d'un système d'exploitation : il offre une double interface entre ce qui est capable de dialoguer dans la même langue que le matériel et les logiciels installés sur la machine. Les logiciels installés, par conséquent « se moque complètement» du type de matériel installé de votre ordinateur : ils envoient des instructions comme « affiche-moi cela », « fais ceci » et le système d'exploitation, par le biais des drivers, fournit la bonne traduction dépendant du matériel.

Remarque

Le serveur X (Xorg ou Xfree) qui gère l'affichage de l'interface graphique est un cas particulier. Certaine personnes disent d'ailleurs que c'est un système d'exploitation ...

F. Un système d'exploitation multiplateforme

Linux est multiplateforme.

Ce terme assez barbare veut tout simplement dire que Linux est disponible sur plusieurs types de machines ou architecture de processeur. Ainsi, on va pouvoir trouver Linux sur une machine de type P.C. tel que vous connaissez et que vous utilisez certainement, mais aussi sur les Macintosh, ou encore sur les super calculateurs.

Voici les différentes architectures sur lesquels nous allons pouvoir installer notre Linux :

- I386 et x86_64 (notre P.C.)
- POWERPC (Ancien Macintosh)
- AMD64
- ARM
- HPPA
- ALPHA
- IA64
- MIPS
- MIPSEL
- Et pour finir SPARC

II

Découverte du système d'exploitation GNU/Linux

A. Ou trouver votre distribution ?

Les distributions préinstallées

Certains vendeurs équipent leurs ordinateurs directement avec l'OS GNU/Linux d'installé, on appelle ça des OEM (Original Equipment Manufacture)

Les distributions payantes en magasin

Certaines distributions de Linux peuvent être achetées en magasin (carrefour, fnac, surcouf, grosbill,...) entre autre:

- Mandriva, qui existe aussi en version gratuite et libre, mais qui propose des versions payantes intégrant des pilotes propriétaires (nvidia, ati, wifi,..) et des codecs.
- Red Hat est comme Mandriva disponible en version payante ou gratuite et libre, cette dernière s'intitulant Fedora Core
- Suse, qui a adopté la même méthode que les 2 autres: une version payante et une version gratuite et libre: Open Suse

Par téléchargement

La plupart des distributions GNU/Linux sont disponibles en libre téléchargement (sauf les versions payantes où il vous faudra évidemment mettre la main au portefeuille avant de pouvoir les télécharger légalement) Vous trouverez le plus souvent les distributions téléchargeables sur leurs sites respectifs. Vous pouvez aussi trouver les liens vers ces fichiers sur un de ces sites:

- Distrowatch http://www.distrowatch.com
- QuebecOs http://www.quebecos.com/modules/wfdownloads/

Avec de la documentation

Bien souvent, les magazines ayant pour thème Linux comportent des DVDs, comportant souvent des distributions linux et des logiciels. Ce type de magazine coûte un peu près 10€, l'installation de la distribution et les premiers pas étant

souvent expliqués dans le magazine.

B. Installation depuis un LiveCD

Maintenant que vous avez téléchargé et gravé votre distribution Linux sur un cédérom, vous pouvez "booter" votre ordinateur sur le lecteur de cédérom.

Image 3 : Écran de démarrage (en anglais !)

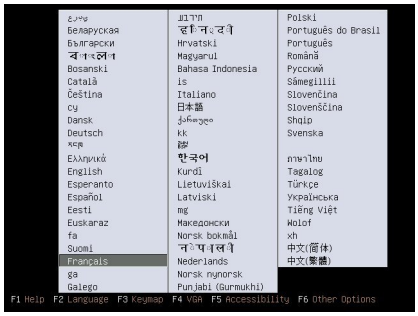


Image 4 : Choisissez ici votre langue

Image 5 : C'est mieux en français ?

Image 6 : Détection du matériel en cours

Image 7 : Lancement de l'interface graphique

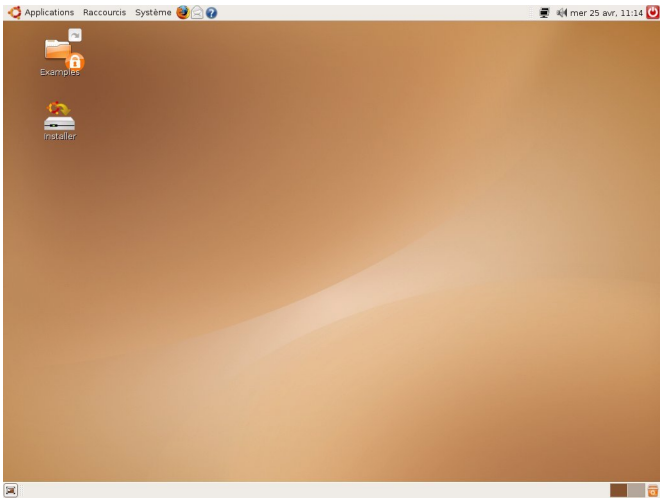


Image 8 : Lancement de l'interface graphique.

Remarque

Dans notre cas, nous allons démarrer notre machine virtuelle et "booter" sur le cd virtuel.

C. Les sessions utilisateurs

Linux est un système d'exploitation multi-utilisateurs.

Définition : Multi-utilisateurs

En informatique, un système d'exploitation comme Unix ou Linux est conçu pour que plusieurs utilisateurs puissent avoir accès au même ordinateur en même temps. Le système partage son temps de travail en plusieurs processus n'ayant aucun lien entre eux, si ce n'est le même ordinateur.

Le terme opposé est mono-utilisateur qui est utilisé lorsque l'on parle de systèmes d'exploitation utilisable par un seul utilisateur à la fois ou encore en référence à une licence de logiciel prévue pour un utilisateur.

D. Les Bureaux Virtuels

Vous aimez faire plusieurs choses en même temps sur votre ordinateur ? Par exemple, vous voulez ouvrir : The Gimp pour modifier quelques photos, une fenêtre Jabber pour discuter avec vos amis, une fenêtre IRC pour discuter avec d'autres personnes, votre navigateur web, votre client email, OpenOffice.org pour rédiger des documents. . . Ça commence vite à faire beaucoup, n'est ce pas ? Alors, trions un peu les fenêtres, séparons-les par thème. . .

Imaginez que vous ayez un bureau pour tout ce qui est internet, un autre pour la bureautique

Eh bien, c'est ce que vous permettent les bureaux virtuels.

Par défaut, sur la plupart des environnements , vous avez sur un de vos tableaux de bord un ensemble de petits carrés ; cliquez sur chacun des ces carrés, qui représentent les bureaux, pour vous faire une idée.

Définition : Bureau virtuel

Un bureau virtuel est un environnement graphique qui peut être démultiplié, afin de travailler à un seul type de tâche dans chaque bureau, au lieu d'avoir toutes les fenêtres réunies dans un seul. Par exemple, l'utilisateur choisira de réserver un bureau aux tâches de retouches photographiques, un autre à internet, un autre aux jeux, etc. Les fenêtres ne sont plus mélangées.

E. Présentation des gestionnaires de fenêtres.

La distribution Ubuntu que nous allons étudier ensemble, propose par défaut l'environnement graphique "Gnome". Mais il existe un grand nombre d'autre gestionnaire de fenêtres.

Les plus connus sont :

- **Gnome** se veut « simple d'utilisation », c'est à dire qu'une application fait une seule chose, mais la fait bien. Les options paramétrables de chaque application restent limitées afin de ne pas noyer l'utilisateur dans des réglages dont il n'aurait jamais à se soucier.

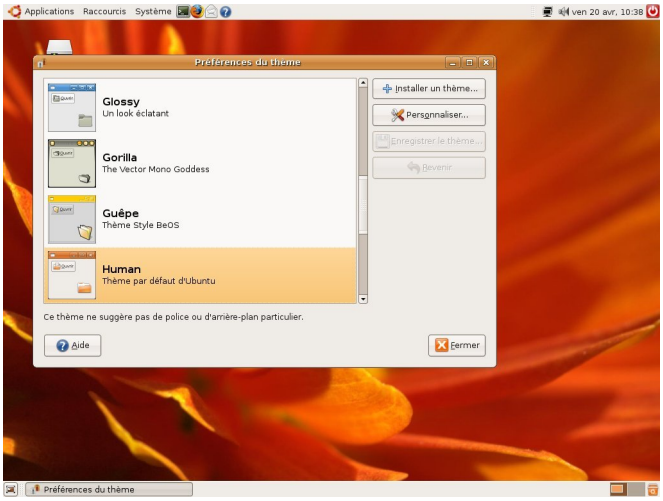


Image 9 : Gnome

- **KDE** se veut plus complet. L'application qui peut graver vos données est tout aussi capable de lire vos vidéos et musiques ainsi que de redimensionner vos images. De plus, les applications sont beaucoup plus paramétrables par le biais des menus d'options, même si vous n'aurez certainement jamais à toucher la plupart de celles-ci !

Image 10 : KDE

- **Xfce**, lui, se veut plus léger et peut tourner sur des configurations modestes comparées aux deux mastodontes précédents. FluxBox vise la même cible.

Image 11 : XFCE

Et bien d'autre encore :

WindowMaker, Enlightenment, BlackBox, Ion, FluxBox, AfterStep, IceWM, ...

Il existe des dizaines d'environnements de bureaux, et deux proposent même un environnement de bureau complet – c'est à dire des applications spécifiquement à destination de cet environnement avec une philosophie, des interactions et une présentation homogène – comme Gnome, interface par défaut d'Ubuntu. Le choix de l'une ou l'autre des interfaces graphiques disponibles dépend uniquement de vos goûts personnels. En effet, dès le système installé, vous pourrez utiliser les mêmes logiciels.

Vous pourrez également changer par la suite d'environnement graphique et en avoir plusieurs installés simultanément. C'est la liberté de choix ! À l'écran déconnexion, vous pourrez choisir votre environnement de bureau. Et tout cela repose, une fois encore, sur le serveur X – Xorg sur Ubuntu – qui se charge de l'affichage. Au démarrage de votre interface graphique, Xorg démarre – et le curseur en forme de roue qui tourne apparaît – en lisant la configuration qui se trouve dans le fichier texte brut /etc/X11/xorg.conf, puis GDM qui vous permet de choisir l'utilisateur et l'interface à utiliser. Vous pouvez alors choisir entre plusieurs environnements, si vous les avez installés sur votre machine. Enfin, l'environnement de bureau sélectionné se lance — Gnome, par exemple.

Les applications disponibles

III

A. Les application de Bureautique

OpenOffice.org, KOffice, Scribus...

B. Les outils Graphiques

The Gimp, Inkscape, F-Spot ...

C. Les outils pour internet.

Firefox, Konqueror, Thunderbird, Evolution ...

D. Les applications Multimédia.

Amarok, Totem, Xine ...

E. Les outils de virtualisation.

VirtualBox, Qemu ...

Installation des applications et mise à jour du système.

IV

A. Comment ajouter des applications ?

La procédure à suivre pour installer un logiciel sous Ubuntu et toutes les distributions dérivées de Debian, se résume en un mot : **APT**. Il s'agit d'un logiciel qui gère l'installation de tous les logiciels.

La première chose à faire avant d'installer une application est de mettre à jour la liste des paquets :

- apt-get update

Si on ne connait pas le nom exacte de l'application on peut la chercher avec apt-cache :

- apt-cache search Nom_du_paquet ou mot_clef

Ensuite pour installer l'application on tape la commande :

- apt-get install Nom_du_paquet

Exemple :

- apt-get install Inkscape

Le successeur d'apt est **aptitude**. Les commandes sont très similaires :

- aptitude update
- aptitude search Nom_du_paquet ou mot_clef
- aptitude install Nom_du_paquet

Si **aptitude** est lancé sans paramètre il propose une interface pour venir parcourir la liste des paquets.

Remarque

Pour rendre ce système – très pratique – accessible au débutant, Ubuntu propose pas moins de deux interfaces graphiques que l'on va détailler par la suite : Synaptic et « Ajouter/enlever. . . ».

B. Mise à jour du système et des applications.

Nous venons de voir les outils apt et aptitude pour installer des logiciels, nous allons utiliser les mêmes outils pour mettre à jour notre système d'exploitation :

- apt-get update && apt-get upgrade

ou

- aptitude update && aptitude upgrade

C'est tout. Avec l'une ou l'autre de ces commandes vous mettez à jour le système et l'ensemble des applications que vous avez installé.

Bien entendu il n'est pas nécessaire d'avoir accès à la ligne de commande pour pouvoir mettre à jour son système.

En effet, ne vous souciez pas des mises à jour, lorsque l'une d'elle va se présenter, votre système vous fera signe par le biais de l'applet « zone de notification » – à côté de votre horloge si vous n'avez touché à rien !

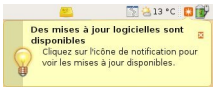


Image 12 : Mises à jour disponibles

Vous pourrez donc mettre à jour l'intégralité de votre système en cliquant simplement sur l'icône – le gestionnaire de mise à jour, également accessible par « Système => Administration => Gestionnaire de mises à jour » –, se lance puis « Installer les mises à jour ». L'application gérant les mises à jour se lance et vous découvrez en-dessous une rapide description, pour chaque élément, de la mise à jour.

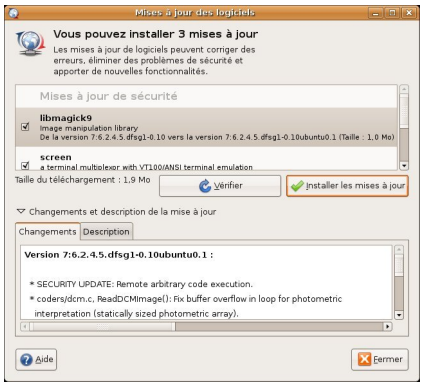


Image 13 : Choix des mises à jours

Plus rarement, le noyau sera mis à jour. C'est, par ailleurs, une des seules actions nécessitant un redémarrage complet du système.

Remarque : Mise à jour de l'intégralité du système

Vous avez dit mise à jour de l'intégralité du système ? En effet, tous les programmes seront mis à jour automatiquement. Vous aurez ainsi l'équivalent d'un énorme « Windows update » prenant en charge l'ensemble de vos logiciels, jeux et outils de sécurité et pas seulement votre système d'exploitation. Si une mise à jour majeure – changement de version, comme Windows Me vers XP – est disponible, Ubuntu vous proposera de l'installer. Vous pouvez ou non accepter sa proposition.

Plus rarement, le noyau sera mis à jour. C'est, par ailleurs, une des seules actions nécessitant un redémarrage complet du système.

C. Synaptic

Synaptic, accessible par le menu « Système => Administration => Gestionnaire de paquets Synaptic », est une interface complète pour gérer APT graphiquement. Il s'adresse principalement aux utilisateurs « avertis ». Si vous connaissez un nom de paquet, vous pouvez le rechercher directement par son nom ou par sa description.

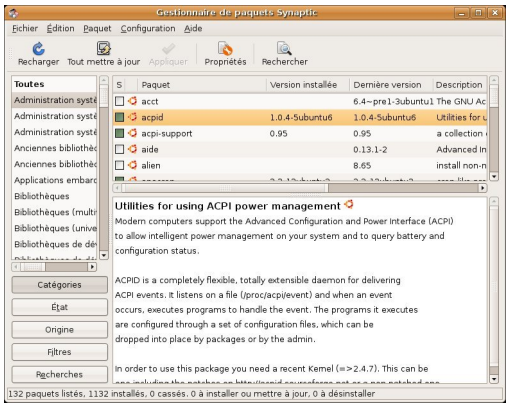


Image 14 : Synaptic

Exemple

Prenons l'exemple d'inkscape, à partir du moment où vous savez que le paquet a le même nom, l'installation est vraiment aisée : une fois **Synaptic** lancé – après avoir demandé votre mot de passe, puisque vous êtes en train d'effectuer une opération d'administration, rappelez-vous ! –, vous pouvez effectuer une recherche par le biais du menu « **Édition) Rechercher** ». Dans la zone de recherche, tapez « **inkscape** » puis cliquez sur « **Rechercher** ».

Vous pouvez choisir d'effectuer votre recherche dans les noms de paquets seulement, ou encore dans les noms et leurs descriptions dans la partie inférieure. Apparaîtra alors la ligne : inkscape.

Cliquez-droit sur celui-ci, puis « **Sélectionner pour installation** ». Acceptez les messages éventuels vous indiquant que d'autres paquets – les dépendances – doivent être installés pour que ce logiciel fonctionne correctement.

Enfin, cliquez sur « **Appliquer** ». Acceptez le message récapitulant ce qui va être fait. Attendez que la magie opère et voilà ! Un carré vert auprès du nom du paquet, prouve que votre logiciel est installé. Vous pouvez à présent fermer Synaptic.

Supprimer un logiciel n'est pas plus compliqué : cliquez-droit sur le paquet à supprimer, puis « **Sélectionner pour suppression**». Acceptez les messages éventuels vous indiquant que d'autres paquets – les dépendances – doivent être supprimés. Enfin, cliquez sur « **Appliquer** ».

Acceptez le message récapitulant ce qui va être fait, et voilà, le paquet et certaines de ses dépendances sont supprimés !

D. L'application « Ajouter/enlever. . . »

Gnome-app-install, accessible par le menu Applications => Ajouter/Enlever. . . est l'interface graphique d'APT de prédilection des débutants proposant des listes ordonnées de logiciels installables et désinstallables en un seul clic ! C'est l'interface que tout débutant devrait utiliser prioritairement.

Image 15 : Ajouter Enlever

Remarque

Par défaut, « peu » d'applications sont disponibles dans « Ajouter/enlever. . . ». En effet, seules les « applications d'Ubuntu maintenues » sont proposées. Pour avoir accès à plus de logiciels, il faudra élargir la recherche, si vous le désirer, aux autres sources de mise à jour telles « universe » ou « multiverse ». Pour cela, il vous suffit de sélectionner dans la liste déroulante supérieure « Toutes les applications disponibles ».

E. Comment sécuriser son Linux ?

Les règles de bases.

- Arrêt et suppression des services inutiles.
- Mise à jour du système.
- Configuration du Firewall avec l'un des outils suivants :
-
  a. FireStarter
  b. GuardDog
  c. Shorewall
  d. NuFW
- Installer un antivirus ? Par nostalgie...
- Et surtout, comme sur tout systèmes, **ne pas installer n'importe quoi** !

Remarque

Plus d'informations à l'url : http://www.debian.org/doc/manuals/securing-debian-howto/

Regardons un peu sous le capot.

V

Contrairement à la situation que nous avons connue il y a quelques années, il est possible aujourd'hui d'utiliser quotidiennement Linux sans en connaître son fonctionnement. Mais, dans ce cas, Linux perd un peu de son intérêt, et nous pouvons nous demander pourquoi il est nécessaire de quitter les systèmes d'exploitations propriétaires.

A. Introduction

Sous Linux et pour l'ensemble des Unix, tout est fichier. Il est donc naturel de commencer par comprendre comment sont agencés ces fichiers.

B. Systèmes de fichiers

Définition

Un système de fichiers (*file system* ou *filesystem* en anglais) ou système de gestion de fichiers (SGF) est une structure de données permettant de stocker les informations et de les organiser dans des fichiers sur ce que l'on appelle des mémoires secondaires (disque dur, disquette, CD-ROM, clé USB, disques SSD, etc.). Une telle gestion des fichiers permet de traiter, de conserver des quantités importantes de données ainsi que de les partager entre plusieurs programmes informatiques. Il offre à l'utilisateur une vue abstraite sur ses données et permet de les localiser à partir d'un chemin d'accès.

Représentation pour l'utilisateur

Pour l'utilisateur, un système de fichiers est vu comme une arborescence : les fichiers sont regroupés dans des répertoires (concept utilisé par la plupart des systèmes d'exploitation). Ces répertoires contiennent soit des fichiers, soit récursivement d'autres répertoires. Il y a donc un répertoire racine et des sous-répertoires. Une telle organisation génère une hiérarchie de répertoires et de fichiers organisés en arbre.

Les systèmes de fichiers sous Linux

Linux possède son système appelé ext2 mais peut en gérer d'autres. La liste en est donnée dans /proc/filesystems

L'utilisateur peut donc accéder sous Linux à d'autres systèmes de fichiers, comme DOS, Vfat,..provenant d'un périphérique ou importé par le réseau.

Comme pour l'utilisateur tout est fichier, tous les systèmes de fichiers quels que soient leur emplacement physique doivent être intégrés dans l'**UNIQUE** arborescence logique du système Linux.

Cette arborescence peut donc être construite (et évoluer) à partir de diverses partitions qui peuvent être situées sur plusieurs disques. Cela réalise une intégration et une abstraction plus poussée que dans le monde Windows où les partitions et lecteurs auxquels sont affectées les lettres A: C: D: ... demeurent des entités séparées. Naturellement la partition sur laquelle est situé le répertoire racine joue un rôle particulier.

Le processus de **montage**, avec sa commande **mount**, décrite plus loin, est le moyen de faire correspondre parties de l'arborescence et partitions physiques de disque. Il permet de plus d'affecter tout système extérieur (disquette, cdrom, clef usb, rép. réseau ...) à un répertoire créé pour cela dans l'arborescence.

Il suffira ensuite de se déplacer à ce répertoire, appelé point de montage, en fait un répertoire "d'accrochage", pour accéder à ses fichiers (bien sûr, conformément aux permissions que possède l'utilisateur)

C. Les différentes catégories de fichiers

fichiers normaux

- texte : courrier, sources des programmes, scripts, configuration ...
- exécutables : programmes en code binaire

fichiers répertoires

ce sont des fichiers conteneurs qui contiennent des références à d'autres fichiers.

véritable charpente de l'arborescence, ils permettent d'organiser les fichiers par catégories

fichiers spéciaux

situés dans /dev, ce sont les points d'accès préparés par le système aux périphériques. Le montage va réaliser une correspondance de ces fichiers spéciaux vers leur répertoire "point de montage".

par exemple, le fichier /dev/hda permet l'accès et le chargement du 1er disque IDE

fichiers liens symboliques

Ce sont des fichiers qui ne contiennent qu'une référence (un pointeur) à un autre

fichier.

Cela permet d'utiliser un même fichier sous plusieurs noms sans avoir à le dupliquer sur le disque.

D. Où sont mes disques ?

Les partitions sont montées dans des dossiers. Linux nomme les partitions sda1 pour la partition 1 du premier disque, sda2 pour la deuxième, sdb1 pour la première partition du deuxième disque (b),etc. Au démarrage de Linux, ce dernier ouvre un fichier texte brut nommé /etc/fstab dans lequel il trouve les correspondances entre les disques durs et les dossiers où ils doivent être montés.

Comment peut-on alors accéder à nos autres partitions, notamment les partitions Windows si vous avez gardé votre ancien système d'exploitation ? En fait, lors de l'installation, vous rappelez-vous de la partie « Point de montage » ? C'est ici où vous avez modifié ou laissé les choix par défaut concernant ce que l'on appelle le « montage » d'une partition. Monter une partition, c'est lier un dossier à une partition. Par défaut, ces dossiers se trouvent dans /media. Par exemple, si j'ai monté ma partition C:\ de Windows dans /media/windows chaque fois que j'enregistrerai dans /media/windows ou un de ses sous-dossiers, je l'enregistrerai en fait sur la partition C:\. De même avec une clef USB ou encore en lisant un lecteur de DVD/CD-ROM.

E. Explication de l'arborescence des fichiers

Un disque dur est un élément matériel qui est généralement placé à l'intérieur de l'ordinateur, c'est un périphérique de stockage magnétique qui va garder, même ordinateur éteint, tous vos documents, mais aussi le système d'exploitation et les fichiers nécessaires à la bonne marche de votre machine. L'arborescence des fichiers est leur organisation sur le disque dur. Si vous utilisez Windows, vous savez certainement que les fichiers se placent dans des répertoires.

Voici pour information quelques explications sur les différents dossiers indispensables dans / :

- **/bin** : Contient les programmes systèmes importants.
- **/boot** : Les fichiers utiles au démarrage du système
- **/dev** : Contient des fichiers factices permettant de communiquer avec vospériphériques.
- **/etc** : Ici se trouve la plupart des fichiers de configuration du système.
- **/home** : Contient les dossiers personnels des utilisateurs. Chacun y possède un dossier à son nom avec ses fichiers personnels.
- **/lib** : Contient les librairies –bibliothèques – utiles au système.
- **/media** : Les dossiers contenus correspondent aux accès de montage des périphériques de stockage.
- **/opt** : A un peu la même fonction que /usr, sauf que certains l'utilisent pour les programmes qu'ils compilent eux-même et qui ne sont logiquement pas aussi intégrés qu'un logiciel disponible dans les sources de mises à jour
- **/proc** : Ce dossier contient des fichiers et dossiers virtuels qui correspondent à l'état du système en temps réel : programmes (processus) lancés, occupation mémoire, RAM disponible, etc...

- **/root** : C'est le /home de l'administrateur ! Ce dernier est séparé pour des questions de sécurité. Cependant, si vous avez bien suivi jusqu'ici, il n'y a pas de compte root à proprement parlé sur Ubuntu. Ce répertoire existe donc seulement pour assurer la compatibilité avec les autres distributions GNU/Linux.
- **/sbin** : A un peu la même fonction que /bin, sauf que tous les programmes issus ne sont accessibles qu'à l'administrateur, ou aux amis de root sur Ubuntu.
- **/tmp** : Comme son nom l'indique, ici sont stockés les fichiers temporaires utiles aux programmes en cours d'exécution. Ce dossier est vidé à chaque redémarrage.
- **/usr** : Dossier important, contenant tous les programmes et les bibliothèques installés.
- **/var** : Dossier contenant tout ce qui est variable au système. Par exemple, les fameux fichiers « log » enregistrant ce qui se passe sur votre système, utiles quand quelque chose ne fonctionne plus par exemple – contenus dans /var/log/ .

F. Point de montage

Définition

Un point de montage est un répertoire à partir duquel sont accessibles les données se trouvant sous forme d'un système de fichiers sur une partition de disque dur ou un périphérique.

Montage et démontage sous Unix

Lorsque les données sont accessibles à partir d'un point de montage, on dit que la partition ou le périphérique sont montés. Dans les sytèmes Unix, le point de montage par défaut est /mnt ou /media. Par exemple, une disquette sera généralement montée en /mnt/fd0 et un cdrom en /mnt/cdrom ou /media/cdrom. Le point de montage par défaut des périphériques est spécifié dans un fichier de configuration système : /etc/fstab (sous Linux, /etc/vfstab sous Solaris). La commande Unix permettant de monter des répertoires est mount. La commande inverse, qui démonte, est umount (et non unmount).

Montage

La commande **mount** permet de relier une partition ou un périphérique à un répertoire, répertoire par lequel les données présentes sur la partition ou le périphérique sont accessibles.

Pour monter un périphérique ou une partition avec la commande **mount**, il faut indiquer :

le type du système de fichiers par l'option **-t**

le fichier spécial représentant le périphérique ou la partition (généralement /dev/*) ;

le répertoire de montage. Par exemple, la commande ci-dessous monte le périphérique /dev/cdrom (cédérom) sur /media/cdrom en indiquant que le système de fichier est ISO 9660.

```
mount -t iso9660 /dev/cdrom /media/cdrom
```

Certaines indications peuvent être omises lorsqu'elles sont spécifiées dans le fichier

de configuration listant les points de montage par défaut (/etc/fstab sous Linux). On peut omettre le type de système de fichiers si la version de mount utilisée est assez « intelligente ». Par contre, même en l'indiquant, on ne pourra jamais monter un système de fichiers que le noyau Unix ne sait pas gérer (parce qu'il n'a pas été configuré pour l'utiliser par exemple).

Lorsque le montage a réussi, une mise à jour est effectuée dans un fichier système recensant les montages en cours (fichier /etc/mtab sous Linux). L'option **-n** de mount permet d'éviter cette mise à jour dans des cas bien particuliers où le montage échouerait pour cette raison (si l'on travaille sur un système de fichier *chrooté* en lecture seule par exemple).

On peut également sous les Unix modernes monter des fichiers qui constituent un système de fichiers à eux-seuls (*loopback*), grâce à l'option **-loop** . Ceci est particulièrement utile dans le cas d'images représentant des disquettes, CDROMs, DVDs. Les commandes **dd** et **mkisofs** peuvent aider à fabriquer de tels fichiers.

Il est possible, sous certaines configurations, de monter (recouvrement total ou partiel) par dessus d'autres systèmes déjà montés

Démontage

Pour démonter une partition ou un périphérique, il faut utiliser la commande umount. Par exemple :

```
umount /media/cdrom
```

Le démontage ne marche que si la partition n'est pas utilisée, à savoir :

- aucun fichier n'est en train d'être lu ou écrit sur la partition ;
- aucun processus n'a son répertoire de travail sur la partition.

Si le démontage est refusé, on peut utiliser la commande fuser pour savoir quels processus l'utilisent. Par exemple (si le démontage de /media/cdrom est refusé) :

```
fuser /media/cdrom
```

Lorsque le démontage a eu lieu, le fichier /etc/mtab est mis à jour.

G. Où sont enregistrées mes préférences ?

Vous avez sûrement remarqué que 2 utilisateurs ne voudront pas du même thème de bureau. De plus, la configuration des logiciels – contacts, messagerie, navigateur, etc. – ainsi que des barres d'outils,par exemple, sera différente. . . Et pourtant, tout ce beau monde se connecte sur le même système et chacun retrouve les préférences qu'il avait paramétrées. Comment cela est-il possible ?

Lorsque vous affichez les fichiers cachés de votre dossier personnel vous retrouverez énormément de dossiers cachés, avec comme nom, celui d'un logiciel que vous utilisez. C'est là que sont stockées vos préférences. Notez également, que s'y trouvent les préférences de session Gnome. Par exemple, vous trouverez un dossier **.evolution** où sont enregistrés vos paramètres de messagerie comme ceux du logiciel Évolution.

D'autres paramètres de configuration, mais plus globaux eux, sont enregistrés dans le dossier /etc. Ceux-ci sont communs à tous les utilisateurs - par exemple, si un logiciel A a besoin d'un logiciel B, pour fonctionner, il y enregistrera le chemin vers le logiciel B pour l'exécuter.

En somme, tout ceci signifie que lorsque vous réinstallerez le système, et si vous avez laissé /home sur une partition séparée, à la réinstallation du logiciel vous

retrouverez tous vos paramètres comme avant le formatage du système,si vous gardez le même nom de compte.

H. Gestion des droits.

Linux est résolument multi-utilisateurs ; il est donc nécessaire de prévoir un système de permissions contrôlant les opérations que chacun peut faire sur les fichiers et répertoires, recouvrant toutes les ressources système (sur un système Unix, tout périphérique est représenté par un fichier ou un répertoire). Ce principe est commun à tous les Unix mais un rappel est toujours utile d'autant qu'il existe quelques usages avancés méconnus et relativement intéressant.

Chaque fichier ou répertoire dispose de permissions spécifiques pour trois catégories d'utilisateurs :

- Son propriétaire (symbolisé par la lettre "u" comme "User").
- Son groupe propriétaire (symbolisé par la lettre "g" comme "Group").
- Et les autres (symbolisé par la lettre "o" comme "Other").

Trois types de droits peuvent s'y combiner :

- Lecture (symbolisé par la lettre "r" comme "Read").
- Ecriture ou modification (symbolisé par la lettre "w" comme "Write").
- Exécution (symbolisé par la lettre "x" comme eXecute).

Dans le cas d'un fichier, ces droits sont faciles à interpréter :

- L'accès en lecture permet d'en consulter le contenu mais aussi de le copier.
- L'accès en écriture de le modifier.
- Et l'accès en exécution permet de tenter de l'exécuter (ce qui ne fonctionnera que s'il s'agit d'un programme).

Un répertoire est traité différemment :

- L'accès en lecture donne le droit de consulter la liste de son contenu.
- L'accès en en écriture celui d'y créer ou supprimer des fichiers.
- Et l'accès en exécution de le traverser (et notamment d'en faire le répertoire courant avec la commande "cd").

Trois commandes manipulent les permissions associées à un fichier :

- chown : affecte un nouveau propriétaire à un fichier ou répertoire.
- chgrp : affecte un nouveau groupe à un fichier ou répertoire.
- chmod : intervient sur les droits.

Il existe deux manières de présenter les droits ; parmi elles, la représentation symbolique, sans doute la plus simple à comprendre et mémoriser, met en jeu les lettres symboliques citées précédemment.

Pour chaque catégories d'utilisateurs(u/g/o), on peut définir les droits (=), en ajouter (+), ou en enlever (-). Ainsi, la formule "chmod u=rwx,g+rw,o-r fichier" donne au propriétaire les droits de lecture, d'écriture, et d'exécution ; ajoute au groupe propriétaire les droits de lecture et d'écriture ; et supprime le droit de lecture aux autres utilisateurs.

Les droits non concernés par les opérations d'ajout ou de retranchement restent inchangés.

La seconde méthode est la représentation numérique octale ; elle associe chaque droit à une valeur :

- 4 la lecture.

- 2 l'écriture.
- et 1 pour l'exécution.

On associe à chaque combinaison de droits la somme de ces chiffres, valeurs qu'on attribue ensuite aux différentes catégories d'utilisateurs en les mettant bout à bout dans l'ordre habituel (propriétaire, groupe, autres).

Ainsi la commande "chmod 765 fichier" mettra donc en place les droits suivant :

- Lecture, écriture, exécution pour le propriétaire (car 7=4+2+1).
- Lecture et écriture au groupe (car 6=4+2).
- Et lecture et exécution aux autres (car 5=4+1).

I. Le run level, ou niveau de fonctionnement

Définition

Le run level, ou niveau de fonctionnement, est une fonction utilisée par le système d'init – système V – notamment lors du démarrage de votre système.

Il existe différents run levels, qui correspondent chacun à un ensemble d'applications et de services à mettre en marche et à arrêter. Ainsi, quand le système est démarré, il passe par le run level 1 puis par le run level 2, et il est possible de configurer les applications et démon qui seront lancés ou arrêtés à ce moment.

Il existe en tout six run levels. Sous Ubuntu, les run levels se répartissent ainsi :

0 : Arrêt

1 : Mode mono-utilisateur

2 : Mode multi-utilisateurs (sans NFS)

3 : Mode multi-utilisateurs

4 : Inutilisé

5 : Mode multi-utilisateurs avec serveur graphique

6 : Redémarrage

Remarque

Les run levels 0, 1 et 6 sont plus ou moins standardisés et communs à tous les systèmes appliquant le système V. Lorsque vous démarrez votre système en« Safe-Mode » – deuxième ligne de Grub – vous restez en fait au run level 1. Lors d'un démarage « normal », vous passez les niveaux 1, 2, puis 5 lorsque GDM se lance.

Notons qu'il n'y a pas de hiérarchie ni de chronologie dans les run levels : il n'est pas nécessaire de passer par 3 pour aller de 2 à 5, par exemple. Les niveaux utilisés pour un fonctionnement continu de la machine – excluant donc ceux permettant de la redémarrer ou de l'arrêter – vont donc de un à cinq, et il est possible de passer de l'un à l'autre. Pour cela, une simple commande telle sudo init 3 fonctionne, mais faites attention à bien avoir enregistré vos données, car aucune confirmation n'est demandée !

J. Les Scripts systèmes

Tous les services et applications qui doivent être démarrés ou arrêtés lors d'un

passage d'un run level à un autre le sont à partir de scripts. Ceux-ci sont regroupés dans le dossier /etc/init.d/ . Ces scripts reçoivent un paramètre qui peut-être start, stop, restart, etc. . .

À chaque niveau correspond un dossier – typiquement /etc/rc.d/rc2.d pour le niveau 2 – contenant des liens symboliques vers les fichiers scripts de /etc/init.d/ . Ces liens symboliques portent des noms commençant par la lettre S ou K, suivie d'un numéro de deux chiffres.

Voici ce qu'il se passe lors d'un changement de run level dans le cas, par exemple, d'un passage du niveau 5 au niveau 3 par la commande **sudo init 3** :

- Les scripts dont le nom commence par un K situés dans le dossier correspondant au niveau de fonctionnement courant – ici 5 – sont lancés dans l'ordre décroissant des numéros avec le paramètre stop, ce qui anormalement pour effet d'arrêter le service correspondant.
- Les scripts du nouveau niveau – ici 3 – qui commencent par S sont activés dans l'ordre croissant des numéros avec le paramètre **start**.

Nous comprenons donc maintenant que les numéros correspondent à une priorité permettant de démarrer un service avant un autre — par exemple, il est préférable de démarrer Xorg avant GDM. . .

Maintenant, si vous êtes curieux, vous pouvez allez voir quels services sont démarrés à quel niveau de fonctionnement sur votre ordinateur – clic droit sur un service, puis propriétés – et désactiver ceux qui ne vous sont pas nécessaires : par exemple, le gestionnaire d'imprimante HP si vous n'avez pas d'imprimante de cette marque.

Pour s'adonner à ces optimisations, une seule direction : Système => Administration => Services.

Gestion Avancée

VI

A. Gestions des utilisateurs et des groupes.

1. Introduction

La liste des utilisateurs est habituellement stockée dans le fichier "/etc/passwd", alors que le fichier "/etc/shadow" stocke les mots de passe chiffrés. Tous deux sont de simples fichier texte, au format relativement simple, consultables et modifiables avec un éditeur de texte. Chaque utilisateur y est décrit sur une ligne par plusieurs champs séparés par des deux-points " :".

2. Liste des utilisateurs "/etc/passwd".

Voici la liste des champs du fichier "/etc/passwd" :

- identifiant (ou login) : il s'agit du nom de connection de l'utilisateur, par exemple "gauthier".
- mot de passe : il s'agit d'un mot de passe chiffré par la fonction à sens unique "crypt" ou "md5". La valeur "x" indique que le mot de passe chiffré est stocké dans "/etc/shadow".
- uid : numéro unique permettant d'identifier l'utilisateur.
- gid : numéro unique du groupe principal de l'utilisateur (Debian et Ubuntu crée par défaut un groupe spécifique pour chaque utilisateur).
- gecos : champ de renseignements qui contient habituellement le nom complet de l'utilisateur.
- répertoire de connection : attribué à l'utilisateur pour qu'il stocke ses fichiers personnels.
- programme à exécuter après la connexion : Il s'agit généralement d'un interpréteur de commandes (shell), donnant libre cours à l'utilisateur pour utiliser la machine. Si l'on précise "/bin/false", l'utilisateur ne pourra pas se connecter.

3. Le fichier des mots de passe chiffrés "/etc/shadow".

Le fichier "/etc/shadow" contient les champs suivants :

- identifiant (ou login).
- mot de passe chiffré.
- plusieurs champs de gestion de l'expiration du mot de passe.

Remarque : Sûreté du fichier "/etc/shadow".

Le fichier "/etc/shadow" contrairement à son alter ego "/etc/passwd", est inaccessible en lecture aux utilisateurs. Tout mot de passe stocké dans "/etc/passwd" est lisible par tous ; un indélicat peut alors entreprendre de le "casser" par une méthode de force brute, consistant tout simplement à chiffrer successivement tous les mot de passe simples pour tenter de le découvrir. Cette attaque, dite "du dictionnaire", qui dévoile les mots de passe mal choisis, n'est plus possible avec le fichier "/etc/shadow".

4. Modifier un compte ou mot de passe existant.

Quelques commandes permettent de modifier la plupart des informations stockées dans ces bases de données. Chaque utilisateur peut ainsi changer de mot de passe, sans doute le champ le plus variable, grâce à la commande "passwd".

"chfn" réservée au super utilisateur "root", intervient sur le champs "gecos".

"chsh" permet de changer de "shell de login", ou interpréteur de commandes, mais le choix des utilisateurs sera limité à la liste des donnée dans le fichier "/etc/shells ; alors que l'administrateur pourra saisir le programme de son choix.

Enfin, la commande "chage" donnera à l'administrateur la possibilité de modifier les conditions d'expiration du mot de passe (l'option -l utilisateur rappelant la configuration actuelle). On pourra d'ailleurs forcer l'expiration d'un mot de passe grâce à la commande "passwd -e utilisateur", qui obligera l'utilisateur à changer son mot de passe à la prochaine connexion.

Remarque

L'utilisation de la commande "usermod" permet aussi d'apporter des modifications sur un compte utilisateur.

a) Bloquer un compte.

On peut se retrouver dans l'obligation de bloquer le compte d'un utilisateur, par mesure disciplinaire ou dans le cadre d'un enquête par exemple. Il s'agit en fait de l'empêcher de se connecter à nouveau sans détruire son identité et ses fichiers. Cela s'effectue simplement par la commande "passwd -l utilisateur" (-l = lock), ou bloquer. La remise en service s'effectue de même , avec l'option "-u" pour "unlock", ou débloquer.

5. Liste des groupes "/etc/group".

La liste des groupes est stockée dans le fichier "/etc/group", simple bas de données textuelle au format comparable à celui de "/etc/passwd" et qui utilise les champs suivant :

- identifiant (le nom du groupe).

- mot de passe (facultatif) : il ne sert qu'à intégrer un groupe dont on n'est pas habituellement membre (avec la commande "newgrp" ou "sg").
- gid : numéro unique identifiant le groupe.
- liste des membres : liste des identifiants d'utilisateurs membres du groupe, séparée par des virgules.

Les commandes "groupadd" et "groupdel" permettent respectivement de créer et de supprimer un groupe.

La commande "groupmod" modifie les informations d'un groupe (son gid ou son identifiant).

La commande "passwd -g groupe" modifiera le mot de passe d'un groupe.

La commande "passwd -r -g groupe" supprimera le mot de passe.

6. Création de comptes.

L'une des premières actions de l'administrateur est de créer les comptes de ses utilisateurs, ce qui s'effectue très simplement avec la commande "adduser". Celle ci prend simplement en argument l'identifiant de l'utilisateur à créer.

"adduser" pose quelques questions avant de créer le compte, mais son déroulement offre peu de surprises. Le fichier de configuration "/etc/adduser.conf offre toutefois quelques paramétrages intéressants. On pourra ainsi prévoir automatiquement un quota à chaque nouvel utilisateur en dupliquant celui d'un modèle. On pourra aussi modifier l'emplacement du compte utilisateur, ce qui ne présente que rarement de l'utilité, c'est le cas si les utilisateurs sont si nombreux qu'il est souhaitable de répartir leurs comptes sur plusieurs disques. On pourra encore choisir un autre interpréteur de commandes par défaut.

La création du compte fabrique le répertoire personnel et y recopie le contenu du répertoire modèle "/etc/skel", afin de fournir quelques fichiers standard à l'utilisateur fraîchement créer.

Dans certains cas, il sera utile d'ajouter un utilisateur dans un groupe, en particulier pour lui conférer des droits supplémentaires. Par exemple, un utilisateur intégré au groupe "audio" pourra accéder aux périphériques son.

Pour ce faire, on procède avec la commande "adduser utilisateur groupe".

B. Planification de tâches : cron et atd.

1. Introduction

"cron" est le démon en charge d'exécuter des commandes planifiées et récurrentes (chaque jour, chaque semaine, etc.) ; "atd" est celui qui s'occupe des commandes à exécuter une seule fois, à un instant précis et futur.

2. Cron

Dans un système Unix, de nombreuses tâches sont régulièrement planifiées :

- La rotation des logs.
- La mise à jour de le base de données du programme "locate".

- Les sauvegardes.
- Des scripts d'entretien (comme le nettoyage des fichiers temporaires).

Par défaut, tous les utilisateurs peuvent planifier l'exécution de tâches. C'est pourquoi chacun dispose de sa propre "crontab", ou il peut consigner les commandes à planifier. Il peut la modifier en exécutant la commande "crontab -e" (ses informations stockées dans le fichier "/var/spool/cron/crontabs/utilisateur").

Remarque : Restreindre "cron" ou "atd".

On peut restreindre l'accès à "cron" en créant le fichier d'autorisation explicite "/etc/cron.allow", où l'on consignera les seuls utilisateurs autorisés à planifier des commandes. Tous les autres seront automatiquement dépourvus de cette fonctionnalité. Inversement pour n'en interdire qu'un ou deux trouble-fête, on écrira leur nom dans le fichier d'interdiction explicite "/etc/cron.deny". Le même mécanisme encadre "atd, avec les fichiers "/etc/at.allow" et "/etc/at.deny".

3. Les fichiers de cron

L'utilisateur root dispose de sa "crontab" personnelle, mais peut également employer le fichier "/etc/crontab" ou déposer des tâches supplémentaires dans le répertoire "/etc/cron.d". Ces deux dernières solutions ont l'avantage de pouvoir préciser l'utilisateur sous l'identité duquel exécuter les commandes.

Le paquet "cron" propose par défaut des commandes planifiées qui éxecutent :

- Une fois par heure les programmes du répertoire "/etc/cron.hourly".
- Une fois par jour les programmes du répertoire "/etc/cron.daily".
- Une fois par semaine les programmes du répertoire "/etc/cron.weekly".

De nombreux paquets profitent de ce services pour déposer dans ces répertoires des scripts de maintenance nécessaires au fonctionnement optimal de leur service.

4. Format d'un fichier "crontab".

Chaque ligne significative d'une "crontab" décrit une commande planifiée grâce aux six champs suivants :

- La conditions sur les minutes (nombres compris de 0 à 59).
- La condition sur les heures (de 0 à 23).
- La condition sur le jour du mois (de 1 à 31).
- La condition sur le mois (de 1 à 12).
- La condition sur le jour de la semaine (de 0 à 7, 0 et 7 correspondant au dimanche ; il est également possible d'employer les trois premières lettre du nom du jour en anglais comme "sun", "mon", etc...).
- La commande à exécuter (quand toutes les conditions précédentes sont remplies).

Chaque condition peut s'exprimer sous la forme d'une énumération de valeurs possibles (séparées par des virgules).

La syntaxe "a-b" décrit l'intervalle de toutes les valeurs entre "a" et "b".

La syntaxe "a-b/c décrit un intervalle avec un incrément de "c" (exemple : 0-10/2 correspond à 0,2,4,6,8,10).

Le joker * représentent toutes les valeurs possibles.

Exemple : Exemple de "crontab" :

```
#Format
#min heu jou moi jsem commande
#Télécharge les données tous les soirs à 19h25.
25 19 * * * $HOME/bin/get.pl
#La matin à 8h00, en semaine (lunid à vendredi)
00 08 * * 1-5 $HOME/bin/fait_quelquechose
#Redémarre le proxy IRC après chaque reboot
@reboot /usr/bin/dircproxy
```

Conseil : Raccourci textuels pour "cron".

Des abréviations, qui remplacent les cinq premiers champs d'une entrée de "crontab", décrivent les planifications les plus classiques.

Les voici :

- @yearly : une fois par an (le premier janvier à 0h00).
- @monthly : une fois par mois (le premiers du mois à 0h00).
- @weekly : une fois par semaine (le dimanche à 0h00).
- @daily : une fois par jour (à 0h00).
- @hourly : une fois par heure (au début de chaque heure).
- @reboot : une fois après chaque démarrage de l'ordinateur.

5. Emploi de la commande "at".

La commande "at" prévoit l'exécution d'une commande à un moment ultérieur. Elle prend en paramètre l'horaire et la date prévus et sur son entrée standard la commande à exécuter. Cette dernière sera exécutée comme si elle avait été saisie dans un interpréteur de commandes. "at" conserve d'ailleurs l'environnement courant afin de pouvoir travailler exactement dans les mêmes conditions que celles de la planification. L'horaire est indiqué en suivant les conventions habituelles : "16 :12" représente "16h12". La date peut être précisée au format "JJ.MM.AA" (27.07.08 représentent ainsi 27 juillet 2008). En son absence, la commande sera exécutée dès que l'horloge atteindra l'heure signalée (le jour même ou le lendemain).

On peut encore écrire explicitement "today" (aujourd'hui) ou "tomorrow" (demain).

Exemple : Exemple d'emploi de la commande "at".

```
$cat <<FIN | at 16 :12 27.07.07
> echo "joyeux anniversaire" | mail moi@exemple.fr
>FIN
warning : commands will be executed using /bin/sh
job 2 at 2007-07-27 16 :12
```

Une autre syntaxe permet d'exprimer une durée d'attente : "at now + nombre période". Période peut valoir minutes, hours (heures, days (jours) ou weeks (semaines). Nombre indique simplement le nombre de ces unités qui doivent s'écouler avant exécution de la commande.

On peut toujours annuler une tâche planifiée avec la commande "atrm numéro-de-tâche". Le numéro de tâche est indiqué par la commande "at" lors de la planification mais on pourra le retrouver grâce à la commande "atq", qui donne la

liste des commandes actuellement planifiées.

C. L'accès à distance par SSH.

1. Introduction

SSH signifie Secure SHell.

C'est un protocole qui permet de faire des connexions sécurisées (chiffrées) entre un serveur et un client SSH. Nous allons utiliser le programme OpenSSH, qui est la version libre du client et du serveur SSH.

2. Mise en garde sur la sécurité.

Nature du problème.

Installer un serveur SSH permet aux utilisateurs d'accéder au système à distance, en rentrant leur "login" et leur mot de passe (ou avec un mécanisme de clés). Cela signifie aussi qu'un pirate peut essayer d'avoir un compte sur le système (pour accéder à des fichiers sur le système ou pour utiliser le système comme une passerelle pour attaquer d'autres systèmes) en essayant plein de mots de passes différents pour un même login (il peut le faire de manière automatique en s'aidant d'un dictionnaire électronique). On appelle ça une attaque en force brute.

Il y a donc trois contraintes majeures pour garder un système sécurisé après avoir installé un serveur SSH :

- Avoir un serveur SSH à jour au niveau de la sécurité, ce qui doit être le cas si vous faites consciencieusement les mises à jour de sécurité en suivant la procédure de votre distribution.
- Que les mots de passes de TOUS les utilisateurs soient suffisamment complexes pour résister à une attaque en force brute.

a) Choisir des mots de passe complexes.

Un mot de passe complexe est un mot de passe qui ne veut rien dire, qui n'est pas dans le dictionnaire et qui comporte au moins 8 caractères, de préférence avec un mélange de lettres minuscules, de lettres majuscules, de chiffres et de caractères de ponctuation.

Une bonne méthode pour obtenir un mot de passe complexe et facile à retenir consiste à choisir une phrase et à prendre la première lettre de chaque mot, avec quelques complications en plus.

Exemple

la phrase "Linux, moi j'y comprends rien de rien !" donne le mot de passe **L,mj'ycr2r**

3. Installation et configuration de SSH.

a) Installation du client et du serveur SSH.

Le client SSH est disponible dans le paquet openssh-client, qui est pré installé.

Pour pouvoir vous connecter à distance, vous pouvez maintenant installer le serveur SSH :

```
# aptitude install openssh-server
```

L'installation comporte une étape de génération des clefs de cryptage. Finalement, le serveur SSH se lance.

b) Configuration du serveur SSH.

Le fichier de configuration du serveur SSH est "/etc/ssh/sshd_config". À ne pas confondre avec le fichier "/etc/ssh/ssh_config", qui est le fichier de configuration du client SSH.

Nous allons vous commenter les lignes les plus importantes de ce fichier de configuration :

- Port 22 : Signifie que le serveur SSH écoute sur le port 22, qui est le port par défaut de SSH. Vous pouvez le faire écouter sur un autre port en changeant cette ligne. Vous pouvez aussi le faire écouter sur plusieurs ports à la fois en rajoutant des lignes similaires.
- PermitRootLogin yes : Signifie que vous pouvez vous logguer en root par SSH. Vous pouvez changer et mettre "no", ce qui signifie que pour vous connecter en root à distance, vous devrez d'abord vous connecter par SSH en tant que simple utilisateur, puis utiliser la commande su pour devenir root. Sans celà, un pirate n'aurait qu'à trouver le mot de passe du compte root, alors que là, il doit trouver votre login et votre mot de passe.

Si vous avez modifié le fichier de configuration du serveur, il faut lui dire de relire son fichier de configuration :

```
# /etc/init.d/ssh reload
Reloading OpenBSD Secure Shell server's configuration
```

4. Se connecter par SSH.

a) Authentification par mot de passe.

C'est la méthode la plus simple. Depuis la machine cliente, tapez :

```
$ ssh login@nom_DNS_du_serveur_SS
```

- Si c'est la première connexion SSH depuis ce client vers ce serveur, il vous demande si le fingerprint de la clé publique présentée par le serveur est bien le bon. Pour être sûr que vous vous connectez au bon serveur, vous devez connaître de façon certaine le fingerprint de sa clé publique et la comparer à celle qu'il vous affiche. Si les deux fingerprints sont identiques, répondez yes, et la clé publique du serveur est alors rajoutée au fichier "~/.ssh/known_hosts".

Ensuite, entrez votre mot de passe... et vous verrez apparaître le prompt, comme si vous vous êtiez loggué en local sur la machine.

b) Authentification par clé.

Introduction

Au lieu de s'authentifier par mot de passe, les utilisateurs peuvent s'authentifier grâce à la cryptographie asymétrique et son couple de clés privée/publique, comme le fait le serveur SSH auprès du client SSH.

Générer ses clés.

Pour générer un couple de clés DSA, tapez :

$ ssh-keygen -t dsa

Par défaut (il demande confirmation lors du processus de création), la clé privée est stockée dans le fichier ~/.ssh/id_dsa avec les permissions 600 et la clé publique est stockée dans le fichier ~/.ssh/id_dsa.pub avec les permissions 644.

Lors de la création, il vous demande une pass phrase qui est un mot de passe pour protéger la clé privée. Cette pass phrase sert à crypter la clé privée. La pass phrase vous sera alors demandée à chaque utilisation de la clé privée, c'est à dire à chaque fois que vous vous logguerez en utilisant cette méthode d'authentification. Un mécanisme appelé ssh-agent permet de ne pas rentrer le mot de passe à chaque fois... comme nous le verrons un peu plus loin dans ce chapitre.

Remarque

Vous pouvez à tout moment changer la pass phrase qui protège votre clé privée avec la commande "ssh-keygen -p".

c) Fichier authorized_keys

Autoriser votre clé publique.

Pour cela, il suffit de copier votre clé publique dans le fichier "~/.ssh/authorized_keys" de la machine sur laquelle vous voulez vous connecter à distance. La commande suivante permet de réaliser cette opération via SSH :

$ ssh-copy-id -i ~/.ssh/id_dsa.pub login@nom_DNS_du_serveur

Une fois votre clée publique copié sur la machine distante il suffira d'établire une simple connections à la machine distante.

5. Transfert de fichiers par SSH.

Utiliser SCP.

Pour illustrer la syntaxe, je vais donner quelques exemples :

- pour transférer le fichier test1.txt situé dans le répertoire courant vers le home du compte toto de la machine ordi1.exemple.org sur laquelle tourne un serveur SSH :

  $ scp test1.txt toto@ordi1.exemple.org:
- pour récupérer le fichier test2.txt situé le home de l'utilisateur toto de la machine ordi2.exemple.org et l'écrire dans le répertoire courant :

  $ scp toto@ordi2.exemple.org:/usr/local/*.txt test-scp
- pour transférer l'intégralité du sous-répertoire test-scp du répertoire courant vers le sous répertoire incoming du home de l'utilisateur toto de la machine ordi1.exemple.org :

  $ scp -r test-scp toto@ordi1.exemple.org:incoming

6. Se connecter par SSH sans taper de mot de passe.

Le principe.

Cette section s'adresse à ceux qui utilisent un couple de clés publiques / privées, et qui ont crypté leur clé privée avec une pass phrase (c'est la configuration la plus sûre). Par conséquent, le client SSH demande la pass phrase à chaque utilisation des clés pour s'authentifier.

Pour éviter d'avoir à taper systématiquement sa pass phrase, il faut utiliser ssh-agent : ce programme tourne en tâche de fond et garde la clef en mémoire. La commande ssh-add permet de donner sa clé à ssh-agent. Ensuite, quand vous utilisez le client SSH, il contacte ssh-agent pour qu'il lui donne la clé.

a) La pratique.

Dans une console, ouvrez un screen avec ssh-agent en tâche de fond :

$ ssh-agent screen

Puis donnez votre clé à l'agent :

$ ssh-add

Il vous demande alors votre pass phrase. Maintenant que votre clé a été transmise à l'agent, vous pouvez vous connecter sans entrer de mot de passe à toutes les machines pour lesquelles vous avez mis votre clé publique dans le fichier ~/.ssh/authorized_keys.

Bash : l'interpréteur de commandes

VII

La console est une interface entre votre système et vous. Elle s'appuie sur un interpréteur de commandes (shell en anglais), sorte de langage. Il en existe plusieurs dans le monde UNIX mais le plus utilisé s'appelle Bash. Il est installé par défaut sur probablement toutes les distributions Linux et contient des centaines de commandes différentes.

A. Bash

Les consoles

Bash peut être utilisé directement en ligne de commande, c'est-à-dire en dehors de toute interface graphique (KDE, Gnome etc). Il existe aussi des interfaces graphiques. Il en existe de nombreuses : Gnome-Terminale, konsole, aterm, eterm, xterm... Leurs différences étant cosmétiques car elles utilisent finalement le même interpréteur de commande, Bash.

Vous accédez ainsi à Bash si, après le démarrage du noyau Linux, vous ne vous connectez pas à un environnement graphique (KDE, Gnome...). Sa puissance est alors à son maximum car le processeur ne s'occupe pas de gérer des fenêtres.

Pour ouvrir une console, le plus simple est d'explorer le menu de l'environnement graphique à sa recherche ou bien de cliquer l'icône probablement installée dans la barre des tâches.

Les terminaux virtuels

Que vous soyez connecté à une interface graphique ou non, il existe un moyen pratique d'ouvrir rapidement une ligne de commande supplémentaire. Le raccourci clavier est CRTL-ALT-Fx. Remplacez le x par un chiffre de 1 à 7. Exemple : CRTL-ALT-F1. Les 6 premiers ouvrent des terminaux virtuels (auxquels il faudra vous

connecter : identifiant et mot de passe) et le 7èm ramène à l'interface graphique (si ouverte). C'est surtout pratique quand aucune interface graphique (KDE, Gnome...) n'est ouverte, vous pouvez ainsi lancer en parallèle des commandes.

Le prompt

Le prompt est une invite de commande. C'est le message que la console place à chaque début de ligne, en attendant votre commande.

Exemple :

luteola@localhost ~ $

Ceci indique que le simple utilisateur (symbole $) 'luteola' travaille sur l'ordinateur 'localhost' est qu'il est dans son répertoire personnel (symbole ~) : /home/luteola

root@localhost /bin #

Dans cet exemple, l'administrateur (root et symbole #) travaille dans le dossier /bin.

Sensibilité à la casse

La console est sensible à la casse (majuscule / minuscule) ! Ainsi, **ls** est une commande alors que **LS** ne veut rien dire. Idem pour l'écriture des dossiers et fichiers. La moindre faute ou espace mal placé aura des conséquences.

Écrire des chemins d'accès

La structure des répertoires sous Linux est pyramidale, la pointe étant appelée la racine. En console, on peut bien sûr se déplacer dans ces répertoires.

Pour séparer des répertoires dans un chemin, on utilise des slashs /, comme on le ferait pour une adresse Internet en fait. N'utilisez donc pas les anti-slash, comme vous le demanderai Windows. Exemple : /home/george/musique.

Les anti-slash servent plutôt à insérer (on dit 'protéger') des caractères spéciaux ou des espaces. Créez 'à la souris' un répertoire "Nouveau Dossier" sur votre bureau (clic droit, Nouveau...) puis en console tapez cd ~/Desktop/ et utilisez la touche Tabulation (qui autocomplète un chemin). 'Nouveau' et 'dossier' sont séparés par un anti-slash.

Autocomplétion des commandes

Bash peut "deviner" la fin d'un chemin ou d'une commande. Ceci s'opère par la touche 'tabulation'. Très pratique pour accélérer la saisie de commandes ou retrouver une syntaxe ! Exemples :

- Tapez up et appuyez sur 'tabulation'. Bash affiche alors la liste des commandes commençant par 'up'.
- Tapez cd /bi puis 'tabulation'. Bash complète la fin du chemin s'il existe, cd /bin/ en l'occurrence.

Mais encore plus fort avec "bash_completion", il peut même compléter lors de l'installation de logiciels.

Par exemple sur Ubuntu ou Debian si vous tapez "source /etc/bash_completion" et qu'en suite vous tapez "aptitude install fire" puis la touche "tabulation", il va vous donné tous les logiciels disponibles qui commencent par "fire"

Historique des commandes

En utilisant les flèches haut et bas, vous parcourez l'historique des commandes tapées sous votre login.

Ou avec la commande "history" il peut vous retrouver des commandes que vous avez tapez le matin ou la veille.

Exemple : $ history | grep cat

CTRL-R permet également de faire une recherche rapide sur une commande.

Jockers

Vous pouvez utiliser des jokers dans une commande :

- ? représente n'importe quel caractère (a, Y, 3, @...).
- * représente n'importe quel chaîne de caractères (un mot, une phrase...). Exemple : ls photo* affiche la liste des fichiers ou dossiers commençant par le mot photo.
- *[a]* représente une chaîne contenant la lettre a.
- *[!a]* représente une chaîne ne contenant pas la lettre a.
- [abc]* représente une chaîne commençant par la lettre a,b ou c.
- [a-d]* représente une chaîne commençant par la lettre a, b, c ou d.

B. Les principales commandes console (BASH) sous Linux.

Voici une liste des commandes console Bash les plus utilisées en routine. Certaines plus que d'autres. N'oubliez pas que souvent des interfaces graphiques existent pour ces commandes.

Commandes de base :

- ls : Afficher la liste des dossier et fichiers d'un répertoire.
- cd : Se déplacer vers un répertoire.
- pwd : Affiche le nom du répertoire courant.
- mkdir : Créer un répertoire
- rmdir : Supprimer un répertoire.
- rm : Effacer un fichier ou un répertoire.
- cp : Copier un dossier ou un répertoire.
- mv : Déplacer (renommer) un dossier ou un répertoire.
- ln : Créer un lien vers un fichier.
- touch : Permet de changer la date d'un fichier à l'heure courante ou de créer un fichier s'il n'existe pas (touch /home/user/test).
- alias : Créer un alias d'une commande (exemple : alias rm="rm -i")

Manipulations de texte :

- cat : Afficher le contenu d'un fichier texte. Concaténer des fichiers.
- more, less : Afficher le contenu d'un fichier texte intelligemment.
- grep : Rechercher une chaîne de caractère.
- sed : Remplacer des occurrences de texte dans des fichiers.
- echo : Envoyer un message texte. Par exemple vers un fichier.
- tail : N'affiche que les dernières lignes d'un fichier (-n permet de spécifier le nombre de lignes à afficher).
- head : Comme tail, mais affiche les N premières lignes d'un fichier (N=10 par défaut).
- sort : Trier un fichier texte.
- uniq : Supprime ou compte les doublons sur un fichier trier.

- wc : Compte le nombre de lignes, mots et caractères.
- awk : Remise en forme de données texte.

Installer des logiciels :

- urpmi : Installer des paquetages RPM sous Mandriva.
- apt-get, aptitude : Installer des paquets DEB (Mepis, Ubuntu, Knoppix, Debian...).
- alien : Convertir des paquetages RPM, DEV, Tar.Gz entre eux.
- emerge : Installer des ebuilds Gentoo.
- up2date, yum: Mise à jour sur Redhat.

Recherche :

- whereis, which : Rechercher une application.
- find : Rechercher un dossier ou un fichier.
- locate : Idem mais dans une base de données. Plus rapide.
- du : Affiche la taille des répertoires.

Utilisateurs, droits et permissions :

- adduser, addgroup, passwd, usermod : Gestion des utilisateurs, des groupes et des mots de passe.
- su, sudo : Passer la main à l'administrateur (root) ou un autre utilisateur.
- chmod : Modifier les permissions sur un fichier.
- chown : Changer le propriétaire/groupe d'un fichier.
- chgroup : Changer le groupe d'un fichier.
- chroot : Modifier la racine. Parfois très utile.

Gestion des processus / tâches :

- ps & top : Voir les processus en cours.
- pidof : Affiche le Process IDentifier d'une application.
- kill, killall : Tuer un processus en cours.
- fuser : Afficher/tuer les processus accédant à un fichier/dossier.
- nice : Fixe la priorité d'exécution d'une tâche.
- renice : Modifier la priorité d'exécution d'une tâche en cours.
- init : Changer de "run level" : sortir de X, rebooter, éteindre.
- pstree : Afficher l'arbre généalogique des processus lancés.
- cron : Planifier des tâches périodiquement (outil crontab).
- lsof : listes les fichiers ouverts.

Téléchargement

- wget, curl : Outils permettant de télécharger les fichiers passés en paramêtre.
- links : Navigateur web en ligne de commande.

Archivage

- tar : Rassembler dans un fichier une liste de fichiers ou de répertoires.
- gzip, bzip2 : Compresser un fichier au format gzip ou bzip2.
- zip, unzip : Compresser un fichier au format zip.

Enchaînements de commandes :

- > >> : Redigirer une sortie de commande vers un fichier.
- ; || && : Enchaîner des commandes séquentielle.
- | : Lancer des commandes simultanément via un 'pipe'.

Variables d'environnement

- export : Charge une variable dans l'environnement courant. (export http_proxy=http://monproxy :3128/)
- unset : Supprime cette variable. (unset http_proxy)

Administration du système :

- fdisk, cfdisk : Formater et partitionner un disque.
- lsmod : Afficher les modules du noyau présents en mémoire.
- modprobe : Manipulation de modules chargeables dans le noyau.
- mount, umount : Monter/démonter des partitions dans l'arborescence de fichiers (/etc/fstab).
- lspci, lsusb : Affiche les périphériques branchés sur ports PCI, AGP ou USB.
- lshw : Liste votre matériel de manière propre.
- dmesg Affiche les messages du noyau.

C. Mon premier script.

Un script est une suite d'instructions élémentaires qui sont exécutées de façon séquentielle (les unes après les autres) par le langage de script. Dans cet article nous nous limiterons à l'utilisation du shell comme langage, et en particulier à du shell bash. Attention, n'espérez pas que le présent document constitue un manuel complet de programmation ! C'est une courte introduction qui nous l'espérons, vous permettra d'écrire de petits scripts qui vous rendront de précieux services.

Le langage Bash gère notamment :

- La gestion des entrées-sorties et de leur redirection.
- Le passage de paramètres.
- Les variables définies par le programmeur et des variables systèmes.
- Des structures conditionnelles et itératives.
- Les fonctions internes.

Pour commencer, il faut savoir qu'un script est un fichier texte standard pouvant être créé par n'importe quel éditeur : vi, emacs, kedit, gnotepad, ou autre. D'autre part, conventionnellement, un script commence par une ligne de commentaire contenant le nom du langage à utiliser pour interpréter ce script, soit dans notre cas : /bin/bash (on parle alors de "script shell").

Nous allons donc commencer par écrire un simple script que l'ont va nommer "bonjour_monde" et qui nous affichera "Bonjour monde !" lors de son éxecution :

```
#!/bin/sh
echo "Bonjour, Monde !"
echo "Un premier script est né."
```

Comment on l'exécute ? C'est simple il suffit de taper :

```
[user@becane user]$ sh bonjour_monde
```

Ou bien on peut le rendre exécutable avec la commande chmod comme ceci :

[user@becane user]$ chmod +x bonjour_monde

Des lors nous pourrons exécuter notre script de cette façon :

[user@becane user]$ ./bonjour_monde

Rappel

Résumons : un script shell commence par : #!/bin/bash, il contient des commandes du shell et est rendu exécutable par chmod +x.

D. Quelques conseils concernant les commentaires.

Dans un script shell, est considéré comme un commentaire tout ce qui suit le caractère # et ce, jusqu'à la fin de la ligne. Usez et abusez des commentaires : lorsque vous relirez un script six mois après l'avoir écrit, vous serez bien content de l'avoir documenté. Un programme n'est jamais trop documenté, en revanche il peut être mal documenté ! Un commentaire est bon lorsqu'il décrit pourquoi on fait quelque chose, pas quand il décrit ce que l'on fait.

Exemple :

```
#!/bin/sh
# pour i parcourant tous les fichiers,
for i in ./* ; do
# copier le fichier vers .bak
cp $i $i.bak
# fin pour
done
```

Que fait le script ? Les commentaires ne l'expliquent pas ! Ce sont de mauvais commentaires.

En revanche :

```
#!/bin/sh
# on veut faire une copie de tous les fichiers du
répertoire courant
for i in ./* ; do
# sous le nom *.bak
cp $i $i.bak
done
```

Là au moins, on sait ce qu'il se passe (il n'est pas encore important de connaître les commandes de ces deux fichiers).

E. Le passage de paramètres

Un script ne sera, en général, que d'une utilisation marginale si vous ne pouvez pas modifier son comportement d'une manière ou d'une autre. On obtient cet effet en « passant » un (ou plusieurs) paramètre(s) au script via la ligne de commande. Voyons comment faire cela.

Soit le script essai01 :

```
#!/bin/sh
echo le paramètre \$1 est \"$1\"
echo le paramètre \$2 est \"$2\"
echo le paramètre \$3 est \"$3\"
```

Que fait-il ? Il affiche les uns après les autres les trois premiers paramètres du script, donc si l'on tape :

```
$ ./essai01 paramètre un
le paramètre $1 est "paramètre"
le paramètre $2 est "un"
le paramètre $3 est ""
```

Donc, les variables $1, $2… $9 contiennent les « mots » numéro 1, 2… 9 de la ligne de commande.

Attention : par « mot » on entend ensemble de caractères ne contenant pas de caractères de séparations. Les caractères de séparation sont l'espace, la tabulation, le retour chariot quand c'est possible et le point virgule.

Vous avez sans doute remarqué que j'ai utilisé les caractères : \$ à la place de $ ainsi que \" à la place de " dans le script. Pour quelle raison ?

C'est simple : si l'on tape echo "essai" on obtient : essai, si l'on veut obtenir "essai" il faut dire à echo que le caractère " n'indique pas le début d'une chaîne de caractère (comme c'est le comportement par défaut) mais que ce caractère fait partie de la chaîne : on dit que l'on « échappe » ou « protège » le caractère " en tapant \". En « échappant » le caractère \ (par \\) on obtient le caractère \ sans signification particulière.

On peut dire que le caractère \ devant un autre lui fait perdre sa signification particulière s'il en a une, ne fait rien si le caractère qui suit \ n'en a pas.

F. Les variables.

La programmation sous shell nécessite naturellement des variables, pour stocker des informations temporaires, accéder à des paramètres, etc. Le contenu d'une variable est considéré comme une chaîne de caractères, sauf si on indique explicitement qu'elle doit être traité comme une variable entière, qui peut être utilisée dans des calculs arithmétiques. De plus, le shell ne permet pas de manipuler directement des données en virgule flottantes.

A la différence des langages compilés habituels, une variable n'a pas à être déclarée explicitement. Dès qu'on lui affecte une valeur, elle commence à exister. Cette affectation prend la forme 'variable=valeur' sans espace autour du signe égal.

Le message d'erreur 'i : command not found' est peut-être le plus connu des utilisateurs du shell.

Exemple :

```
$ i = 1
bash : i : command not found
```

En raison des espaces autour du signe égal, Bash a cru que l'on essayait d'invoquer la commande 'i' en lui transmettant les arguments '=' et '1'.

La bonne syntaxe est la suivante :

```
$ i=1
```

Pour accéder au contenu d'une variable, il suffit de préfixer son nom avec le caractère '$'. Il ne faut pas confondre ce préfixe des variables avec le symbole

d'invite du shell qui est généralement le même caractère '$'. La commande echo affiche simplement le contenu de la variable.

```
$ echo $i
1
```

Le nom attribué à une variable peut contenir des lettres, des chiffres, ou le caractère souligné '_'.

Voyons quelques exemples :

```
$ variable=12
$ echo $variable
12
```

Il faut donc comprendre que le shell a remplacé la chaîne $variable par sa valeur, 12, avant d'appeler la commande 'echo'. Cette dernière a donc été invoqué par la ligne de commande 'echo 12'.

```
$ variable=abc def
bash : def : command not found
```

Ici le shell n'a pas pu interpréter correctement cette ligne, car il a cru quelle se composait d'une affectation 'variable=abc', suivie d'une commande nommée 'def' (syntaxe rarement utilisé, mais autorisée). Il faut lui indiquer que les mots à droite du signe égal forment une seul chaîne de caractères. On emploie pour cela les guillemets droits :

```
$ variable="abc def"
$ echo $variable
abc def
```

Nous pouvons aussi vérifié qu'une variable qui n'a jamais été affectée est considérée comme une chaîne vide :

```
$ echo $inexistante
```

Elles sont gérées par le système et s'avèrent très utiles dans les scripts. Bien entendu, elles ne sont accessibles qu'en lecture.

Ces variables sont automatiquement affectées lors d'un appel de script suivi d'une liste de paramètres. Leurs valeurs sont récupérables dans $1, $2 ...$9.

$? : C'est la valeur de sortie de la dernière commande. Elle vaut 0 si la commande s'est déroulée sans problème.

$0 : Cette variable contient le nom du script qui a été appelé.

$1 à **$9** : Les (éventuels) premiers arguments passés à l'appel du script, comme vue précédemment.

$# : Le nombre d'arguments passés au script.

$* : La liste des arguments à partir de $1.

$$: Le numéro PID du processus courant.

$! : le n° PID du processus fils.

G. Protection des expressions.

Il peut arriver que certaines expressions possèdent des caractères qui ont une signification particulière pour le shell, et que nous ne souhaitions pas qu'elles soient interprétés par ce dernier.

Par exemple, afficher le prix américain '$5' n'est pas facile :

```
$ echo $5
$ echo "$5"
```

En effet, le shell peut en déduire que vous souhaitez afficher le contenu du cinquième paramètre positionnel, vide en l'occurrence. De même, le symbole '#' sert à introduire un commentaire qui s'étend jusqu'à la fin de la ligne, et le sens d'un message peut en être modifié :

```
$ echo en Fa # ou en Si ?
en Fa
```

Une autre surprise attend le programmeur qui veut réaliser un joli cadre autour du titre de son script :

```
$ echo ****************************
toto titi tata tete
```

Le caractère '*' remplace n'importe quelle chaîne dans un nom de fichier. La commande **echo** reçoit donc en argument la liste des fichiers du répertoire courant.

La solution adéquate consiste à protéger le caractère spécial de l'interprétation du shell. Cela peut s'effectuer de plusieurs manières. On peut ainsi utiliser le caractère backslash (barre oblique inversée) '\' , les apostrophes ' ', ou encore les guillemets " ".

Protection par le caractère backslash.

Ce caractère sert à désactiver l'interprétation du caractère qu'il précède. Ainsi, on peut écrire :

```
$ echo \$5
$5
$ echo en Fa \# ou Do \#
en Fa # ou Do #
$ echo \*\*\*
***
```

Le caractère backslash peut servir à préfixer n'importe quel caractère, y compris lui-même :

```
$ echo un \\ précède \$5
un \ précède $5
```

Lorsqu'il précède un retour chariot, le backslash a pour effet de l'éliminer et la saisie continue sur la ligne suivante.

```
$ echo début \
> et fin.
début et fin.
```

Protection par apostrophes.

La protection des expressions par un backslash qui précède chaque caractère spécial est parfois un peu fastidieuse, et on lui préfère souvent le mécanisme de protection par apostrophes, qui permet de manipuler toute l'expression en une seule fois. Entre des apostrophes tous les caractères rencontrés perdent leur signification spéciale. Cela signifie que le backslash est un caractère comme les autres, et que l'on ne peut pas l'employer pour protéger une apostrophe. Le seul caractère qui ne puisse pas être inclus dans une chaîne protéger par des apostrophes est donc l'apostrophes lui même.

```
$ echo '#\s"&>|'
```

```
#\s"&>|
```

Lorsqu'un retour chariot est présent dans une chaîne entre apostrophes, la représentation du code est inchangée :

```
$ echo 'début
> milieu
> et fin'
début
milieu
et fin
```

Protection par guillemets.

La protection par des apostrophes est totale, chaque caractère garde sa signification littérale. Il arrive pourtant que l'on préfère quelque chose d'un peu moins strict. La protection par des guillemets est plus adaptée.

Entre les guillemets, les caractères '$', apostrophes, et backslash retrouvent leur significations spéciales, alors que les autres sont réduits à leur expressions littérales. En fait, le backslash et le '$' n'ont un sens particulier que s'ils sont suivis d'un caractère pour lequel l'interprétation spéciale a un sens.

Voyons quelques exemples :

```
$ A="ABC DEF"
$ B="$A $ \$A \ \" "
$ echo $B
ABC DEF $ $A \ "
```

Dans l'affectation de la variable B, le premier '$' est suivi de A ; l'évaluation fournit le contenu de la variable A.

Le deuxième '$' est suivi par un espace, et une interprétation autre que littérale n'aurait pas de sens.

Le troisième '$' est précédé d'un backslash, il n'a donc pas de sens particulier, et l'on affiche les deux caractères $A.

Le backslash suivant précède un espace, la seule interprétation est donc littérale.

Comme le guillemet suivant est précédé d'un backslash, il est littérale et ne sert pas à fermer la chaîne ; il est donc affiché.

Et pour finir le dernier guillemet clôt l'expression.

H. Structures de contrôle

En algorithmique, l'exécution des instructions se définit classiquement selon trois types d'enchaînements :

- Séquence : les instructions sont exécutées l'une après l'autre, chacune attendant la fin de la précédente pour démarrer.
- Sélection : Certaines instructions seront exécutées ou non en fonction d'une condition.
- Itération : Une instruction est répétée plusieurs fois, en fonction d'une condition.

Nous pouvons ajouter aux possibilités des scripts shell les exécutions parallèles de tâches.

1. Sélection d'instructions.

a) Construction if-then-else.

la structure "**if-then-else**" permet l'exécution d'instruction sous certaines conditions.

Voici la syntaxe complète de cette construction :

Syntaxe

```
if [ condition_1 ] ; then
commande_1
elif [ condition_2 ] ; then
commande_2
else
commande_n
fi
```

La "condition_1" est exécutée. Il peut s'agir d'une commande composée quelconque, mais on emploie souvent la commande [ ] pour vérifier une condition. Le code de retour de "condition_1" est examiné. S'il est nul, alors la condition est considérée comme vraie. Dans ce cas, la commande composée "commande_1" est exécutée, puis le contrôle est transféré à la fin de la construction, après le "fi".

Si le code de retour de "condition_1" est non null, elle n'est pas vérifiée. Le contrôle passe alors à la deuxième partie de la sélection, l'instruction "elif" (contraction de "else if", en français "sinon si"). La "condition_2" est exécutée, puis, si elle est vérifiée, la "commande_2" est exécutée. Sinon, le contrôle passe à la dernière partie, indiquée par else (en français sinon, c'est à dire dans tous les autres cas), et "la commande_n" est exécutée.

b) Construction case-esac.

La seconde structure de sélection proposée par le shell est introduite par le mot-clé "case", et terminée par esac.

La forme générale en est la suivante :

Syntaxe

```
case expression in
motif_1 ) commande_1 ; ;
motif_2 ) commande_2 ; ;
....
esac
```

L'expression indiquée à la suite du case est évaluée puis sont résultat est comparé (en tant que chaîne de caractères) avec les différents motifs fournis ensuite.

Si la correspondance entre l'expression et le motif est réalisée, les commandes composées qui le suivent sont exécutées, puis le contrôle passe à la fin de la structure

Les motifs peuvent contenir des caractères génériques du shell comme l'astérisque qui remplace n'importe quelle séquence de caractères, le point d'interrogation qui remplace un unique caractère, ou un encadrement de caractères entre crochets indiquant une alternative ou un intervalle.

Des motifs peuvent également être juxtaposés avec un symbole "|" signifiant OU.

c) Les conditions de tests.

Dans une construction"**if-then**", la condition qui se trouve après le "if" ou le "elif" est représentée par une fonction dont le code de retour correspond à une valeur vraie (0) ou fausse (retour non nul). Dans de très nombreux cas, les conditions que nous souhaiterons vérifier consisteront en des comparaisons numériques, comparaisons de chaînes de caractères, ou consultations de l'état d'un fichier.

Pour ce faire, le shell Bash nous offre un opérateur interne nommé "test" qui accepte les arguments suivants en ligne de commande :

Option	Vraie si
-a fichier -e fichier	
-b fichier	Le fichier indiqué est un nœud spécial qui décrit un périphérique en mode bloc.
-c fichier	Le fichier indiqué est un nœud spécial qui décrit un périphérique en mode caractère.
-d répertoire	Le répertoire indiqué existe.
-f fichier	Le fichier indiqué est un fichier régulier.
-g fichier	Le bit Set-GID du fichier indiqué est positionné.
-h fichier	Le fichier indiqué est un lien symbolique.
-G fichier	Le fichier indiqué appartient au même groupe que le GID effectif du processus invoquant la commande test.
-k fichier	Le bit sticky du fichier indiqué est positionné.
-n chaîne	La longueur de chaîne indiquée est non nulle.
-N fichier	Le fichier indiqué a été modifié depuis son dernier accès en lecture.
-O fichier	
-p fichier	Le fichier indiqué est un tube nommé (fichier (FIFO).
-r fichier	Le fichier indiqué est lisible.
-s fichier	La taille du fichier indiqué est non nulle.
-S fichier	Le fichier indiqué est un socket.
-t description	La description du fichier correspond à un terminal.
-u fichier	Le bit Set-UID du fichier indiqué est positionné.
-w fichier	On peut écrire dans le fichier indiqué.
-x fichier	Le fichier indiqué est exécutable.
-z chaîne	La longueur de chaîne indiquée est nulle.
	La chaîne est non nulle.
chaîne_1 = chaîne_2	Les deux chaînes sont identiques.
chaîne_1 != chaîne_2	Les deux chaînes sont différentes.
	La première chaîne apparaît avant la seconde dans un tri lexicographique croissant.
	La première chaîne apparaît après la seconde dans un tri lexicographique croissant.
valeur_1 -eq valeur_2	Les deux valeurs arithmétiques sont égales.
valeur_1 -ge valeur_2	La première valeur est supérieure ou égale à la seconde.
valeur_1 -gt valeur_2	La première valeur est strictement supérieure à la seconde.
valeur_1 -le valeur_2	La première valeur est inférieure ou égale à la seconde.

Tableau 1 : Tableau des conditions

Les mêmes options peuvent être utilisées dans la commande interne [ ] qui est un synonyme de "test".

2. Itérations d'instructions.

a) Introduction

Les séquences d'instructions se présentent traditionnellement sous deux formes :

- Celles qui consistent en un nombre limité de boucles (répéter 10 fois la

séquence).

- Et celles qui dépendent d'une condition d'arrêt (recommencer tant que le maximum n'est pas atteint).

Les deux constructions proposées par le shell reflètent ces deux mécanismes.

b) Répétitions while-do-done et until-do-done.

Introduction

Les deux structures "while-do-done" et "until-do-done" servent à répéter une séquence d'instructions jusqu'à ce qu'une condition soit vérifiée.

Syntaxe : Instruction while-do-done

```
while [ condition  ]; do
commandes
done
```

Exemple

```
#!/bin/bash
echo "Entrez un nom de fichier"
read fich
while [ -z "$fich" ] ; do
echo "Saisie à recommencer"
read fich
done
```

Dans cet exemple le script nous demande de rentrer un nom de fichier et si nous ne rentrons rien, la boucle recommence jusqu'à ce que la chaîne de caractère ne soit pas vide.

Syntaxe : Instruction until-do-done

```
until [ condition ] ; do
commandes
done
```

Exemple

```
#!/bin/bash
until [ "$var1" = fin ] ; do
echo "Variable d'entrée"
echo "(fin pour sortir)"
read var1
echo "variable = $var1"
done
```

Ici la boucle continuera jusqu'à ce que nous tapions le mot fin.

Conclusion

La premières construction répète les commandes composées qui se trouvent dans le corps de la boucle tant que (while) la condition est vérifiée.

La seconde les répète jusqu'à ce que (until) la condition devienne vraie ; en d'autres thermes elle les répète tant que la condition est fausse.

Le code de retour général est celui de la dernière commande exécutée.

c) Construction for-do-done.

Introduction

La boucle "for" est un grand classique que l'on rencontre dans la plupart des langages de programmation. Hélas, son fonctionnement dans les scripts shell est totalement différent.

Une boucle "for-do-done" avec Bash se représente ainsi :

Syntaxe : Instruction for-do-done

```
for variable in liste_de_mots ou commande ; do
commandes
done
```

La variable va prendre successivement comme valeur touts les mots de la liste ou de la sortie de la commande suivant le in, et le corps de la boucle sera répété pour chacune de ces valeurs.

Exemple

Voyons un exemple avec une liste de mots:

```
# !/bin/bash
for i in 1 2 3 5 7 11 13 ; do
echo "$i² = $((i * i))"
done
```

Dont l'exécution affiche les carrés des premiers entiers indiqués :

```
$ ./exemple_for_1.sh
1² = 1
2² = 4
3² = 9
5² = 25
7² = 49
11² = 121
13² = 169
```

Conclusion

VIII

Merci de votre attention.