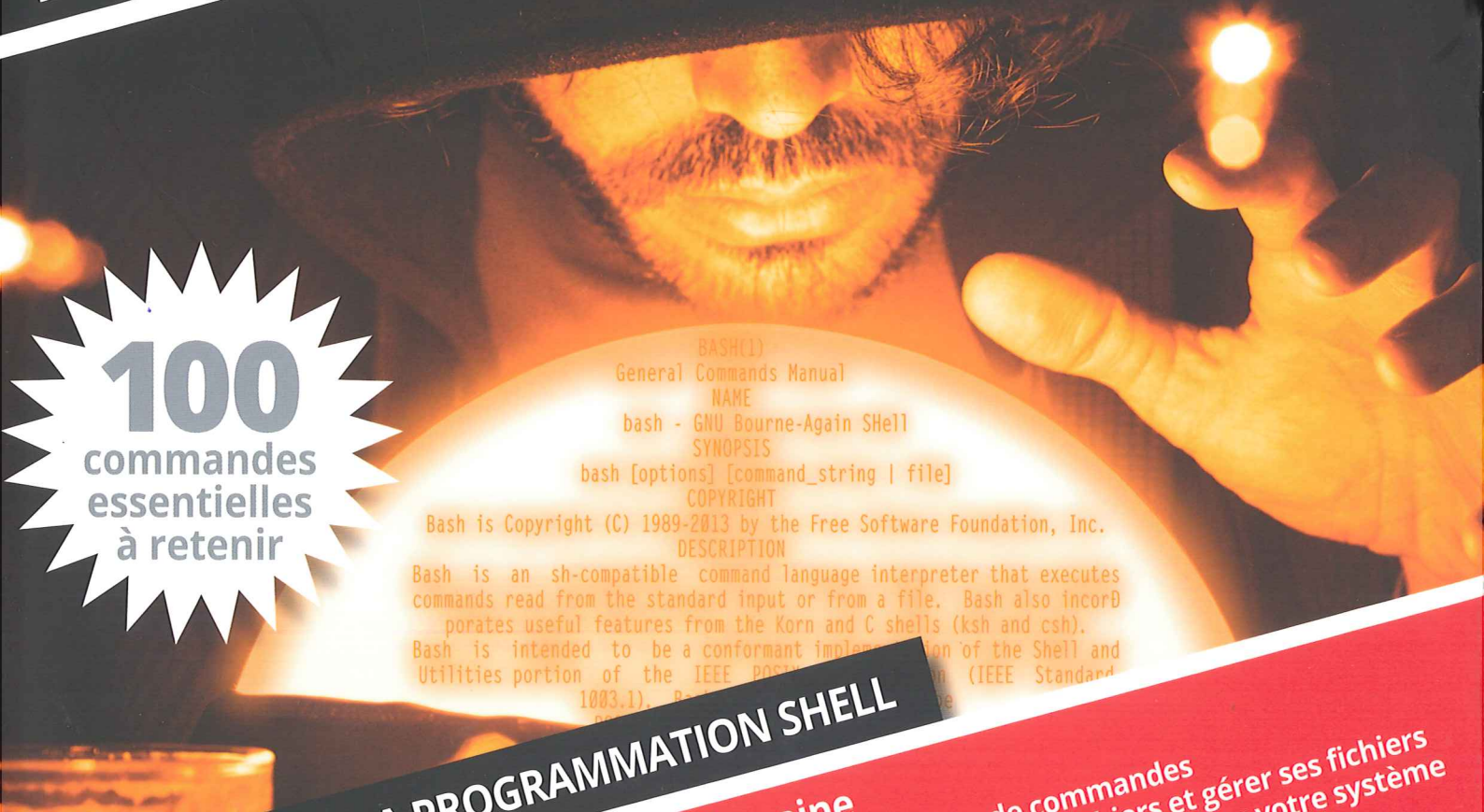
LES GUIDES DE GNU LINUX PRATIQUE
DÉCOUVRIR, COMPRENDRE ET UTILISER LINUX
HORS-SÉRIE N°32
France METRO : 12.90 € — CH : 18,00 CHF — BEL/PORT.CONT : 13,90 € — DOM TOM : 13,90 € — CAN : 18,00 $ cad — MAR : 130 MAD

LIGNE DE COMMANDES

LE GUIDE POUR APPRENDRE LE SHELL EN 7 JOURS !

COMPATIBLE WINDOWS - MAC OS X - LINUX



100 commandes essentielles à retenir

+ INITIATION À LA PROGRAMMATION SHELL

Introduction

Ligne de commandes et système de fichiers : un peu de théorie

Avant de débuter

Utiliser la ligne de commandes sur Linux, Mac OS X et Windows

Votre semaine

Jour 1 : Débuter avec la ligne de commandes
Jour 2 : Connaître son système de fichiers et gérer ses fichiers
Jour 3 : Afficher et filtrer des informations dans votre système
Jour 4 : Gérer et surveiller votre système
Jour 5 : Configurer et utiliser vos connexions réseau
Jour 6 : Faites vos premiers pas en programmation shell
Jour 7 : Gagnez en efficacité avec les scripts shell

Édité par Les Éditions Diamond
L 12225 - 32 H - F: 12,90 € - RD
www.ed-diamond.com

Jour 1

Débuter avec la ligne de commandes

Jour 2

Connaître son système de fichiers et gérer ses fichiers

Jour 3

Afficher et filtrer des informations dans votre système

Jour 4

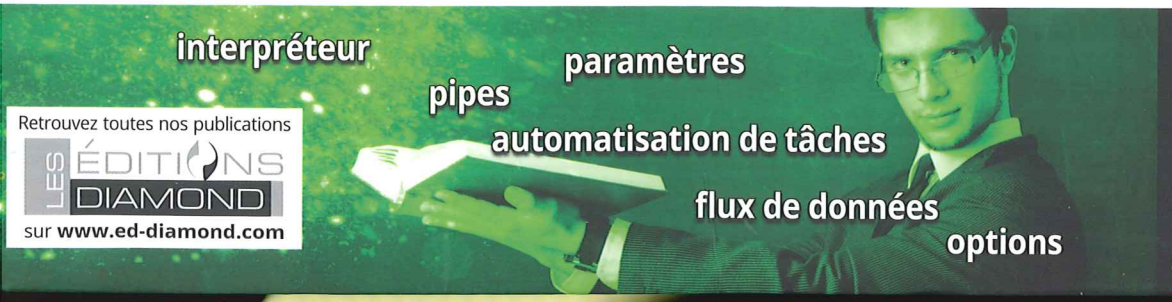
Gérer et surveiller votre système

Jour 5

Configurer et utiliser vos connexions réseau

Jour 6

Faites vos premiers pas en programmation shell



Jour 7

Gagnez en efficacité avec les scripts shell

Retrouvez toutes nos publications
LES ÉDITIONS DIAMOND
sur **boutique.ed-diamond.com**

Linux Pratique Hors-Série
est édité par **Les Éditions Diamond**

B.P. 20142 / 67603 Sélestat Cedex

Tél. : 03 67 10 00 20 / **Fax** : 03 67 10 00 21

E-mail : cial@ed-diamond.com
lecteurs@linux-pratique.com

Service commercial : abo@linux-pratique.com

Sites : www.linux-pratique.com
www.ed-diamond.com

Directeur de publication : Arnaud Metzler

Chef des rédactions : Denis Bodor

Rédactrice en chef : Aline Hof

Conception graphique : Kathrin Scali

Responsable publicité : Tél. : 03 67 10 00 27

Service abonnement : Tél. : 03 67 10 00 20

Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Illustrations/photos : www.fotolia.com

Distribution France :
(uniquement pour les dépositaires de presse)

MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes :
Distri-médias : Tél. : 05 34 52 34 01

IMPRIMÉ en Allemagne - PRINTED in Germany

Dépôt légal : A parution

N° ISSN : 2101-6836

Commission Paritaire : K78 990

Périodicité : Bimestrielle

Prix de vente : 12,90 Euros

PRÉFACE

18 ans ! Heu non, ce n'est malheureusement pas mon âge (!), mais celui de ma première installation d'une distribution Linux sur une bête de course : un 486 DX 4/100. Il s'agissait d'une distribution Slackware francisée : « Linux Kheops » éditée par les « logiciels du soleil ». On est en 1996 et elle embarque un noyau 2.0 et Xfree86. À cette époque, après avoir laborieusement créé les disquettes d'installation de 1,44Mo, l'installation se déroulait exclusivement en mode texte. Je me souviens avoir ensuite passé une bonne partie de la nuit avant d'arriver à configurer convenablement le serveur X à l'aide de la bible de l'époque : le Guide du Rootard.

Bon je sais, ça fait un peu « Moi d'mon temps tout n'était pas si simple, on savait utiliser la ligne de commandes et on comprenait ce qu'on faisait ». Maintenant, les installations des systèmes Linux sont ultra simplifiées et s'opèrent en mode graphique en quelques clics de souris. C'est incontestablement un plus, car il est évident que cela contribue à la diffusion au plus grand nombre de notre système préféré. Cependant, ces automatisations et dissimulations de la complexité du système d'exploitation ont des effets pervers : l'utilisateur ne maîtrise pas, voire ne comprend pas comment cela fonctionne. L'ordinateur devient alors un outil magique ! J'exagère à peine. Faites un petit sondage autour de vous en demandant que l'on vous explique la différence entre Word et Windows, ou encore entre un fichier DOC et un répertoire ?

Je suis de ceux qui considèrent que certaines notions de base doivent être acquises pour permettre une utilisation optimale et raisonnée de l'outil informatique : un peu comme quand vous devez prendre des leçons de code de la route avant d'avoir vos premières leçons de conduite. Sans ces connaissances « théoriques », l'usage de l'ordinateur peut même devenir contre-productif. Mais force est de constater que les éditeurs de système d'exploitation ont tout fait pour tenter de masquer la complexité en y parvenant plus ou moins bien : « *The instruction at 0x00aed157 referenced memory at 0x00000000. The memory could not be read. Clic on o to terminate the program* », « *The application XXXX quit unexpectedly. Macosx and other applications are not affected* », « *Sorry, Ubunu 12,04 has experienced an internal error. If you notice further problems, try restarting the computer* ».

La plupart du temps, ceci oblige ainsi les utilisateurs à payer du support ou de la maintenance (ou bien de solliciter l'informaticien de la famille tous les 4 matins !) lorsque des problèmes mineurs, mais ennuyeux surviennent alors qu'avec quelques notions en matériel et système d'exploitation ces soucis pourraient être corrigés facilement. Mais effectivement, généralement ces opérations demandent d'utiliser la ligne de commandes, ce qui peut rebuter les plus motivés.

Ce hors-série est donc fait pour ceux qui utilisent exclusivement des interfaces graphiques, mais qui désirent comprendre plus en détail le fonctionnement de leur système d'exploitation. À la fin des différentes parties de ce mook, ils seront capables de réaliser des tâches courantes d'administration système, des automatisations pour gagner en productivité, d'être autonomes dans la recherche de problèmes et de pannes et ainsi de maîtriser et d'utiliser de manière optimale leur outil informatique. Vous êtes prêts, allons-y !

Yann Morère

Sommaire

Linux Pratique Hors-Série N°32

LIGNE DE COMMANDES

INTRODUCTION

LIGNE DE COMMANDES ET SYSTÈME DE FICHIERS : UN PEU DE THÉORIE

Yann Morère

Avant de nous lancer dans l'utilisation des commandes Unix, nous allons présenter quelques notions importantes pour une meilleure compréhension de votre système et par la même occasion, une meilleure utilisation de celui-ci.

1 INTRODUCTION

Si les interfaces graphiques sont omniprésentes sur les machines de bureau et indispensables pour la création à l'aide des logiciels AO (PAO, CAO, FAO, PréAO, etc.), elles peuvent se révéler totalement inadaptées pour des tâches répétitives, de maintenance du système (configuration et réparation) ou encore sur les systèmes embarqués. La ligne de commandes est utilisée pour la majorité des opérations qui nécessitent un accès aux fonctionnalités primaires du système d'exploitation (gestion du système de fichiers, gestion mémoire, gestion des droits et des priorités, sécurité, réseau, etc.). Oui c'est vrai, ça s'appelle de l'administration système... le mot est lâché. Mais à notre niveau, on parlera plutôt de gestion et maintenance de l'ordinateur, le métier d'administrateur système étant bien plus vaste et complexe (réseau, sécurité, etc.). On utilise alors des programmes et des scripts en ligne de commandes qui nous permettent d'interagir directement avec le système d'exploitation (noyau). Certains diront que le vrai Linux passe obligatoirement par le mode texte et la ligne de commandes, puisqu'étymologiquement, Linux désigne uniquement le noyau et donc les fonctions de base du système.

La philosophie Unix, dont est issu Linux, veut que toute action puisse être réalisée en ligne de commandes avant d'être utilisée via un programme utilisant une interface graphique. La plupart du temps, ces applications ne sont que des frontaux graphiques permettant de construire la bonne ligne de commandes ! Alors, pourquoi ne pas utiliser le programme original en mode texte ?

La connaissance des commandes shell devient indispensable si vous désirez maîtriser le système d'exploitation. Par exemple, elles vous permettront de réparer une interface graphique qui ne se lance plus après une mise à jour qui s'est mal passée.

L'accès aux programmes console vous permettra d'optimiser votre configuration, de vous connecter à distance sur votre machine à l'aide d'un shell sécurisé (SSH), d'étudier les journaux système en cas de fonctionnement étrange, d'automatiser des tâches (renommage en masse), de rechercher rapidement des informations/fichiers, d'utiliser des systèmes embarqués pour lesquels une interface graphique n'est pas adaptée ou n'existe pas (serveur, Raspberry Pi, Armadeus)... Pour résumer, cela vous permettra de gagner beaucoup de temps sur des tâches qui ne sont pas productives.

Si vous n'êtes pas encore convaincu, voici 10 raisons supplémentaires d'utiliser la ligne de commandes plutôt qu'une interface graphique :

- le clavier est plus rapide que la souris : à votre avis, pourquoi dans les menus graphiques de vos applications vous retrouvez des « raccourcis clavier » ?
- elle permet d'obtenir de l'aide et d'aider plus facilement : avez-vous remarqué que sur les forums d'entraide on retrouve la plupart du temps les solutions aux problèmes sous la forme de ligne de commandes ? C'est bien plus simple et rapide à expliquer qu'un cheminement complet à travers un logiciel graphique (des captures d'écran sont alors obligatoires). De plus, on peut rapidement faire un copier/coller de la commande pour régler le problème ;
- les commandes des systèmes Unix sont standardisées alors que les interfaces graphiques ne le sont pas encore. Mis à part quelques commandes spécifiques, l'utilisation de la console est identique, quelle que soit la distribution utilisée, alors que les interfaces graphiques peuvent être très différentes : Unity, GNOME, KDE, etc.
- l'utilisation de la ligne de commandes est plus puissante et plus flexible ! Eh oui, le frontal graphique reprenant le programme console ne possède peut-être pas toutes les options disponibles en ligne de commandes ;

Illustration basée sur le GKND Creator (https://framalab.org/gknd-cre). Œuvre originale de Simon « Gee » Giraudot. Licence Creative Commons By-Sa.

- les commandes peuvent être automatisées et réutilisées dans des scripts. Dans le cas d'une interface graphique, il faudrait un système de macro afin d'enregistrer les actions répétitives. En ligne de commandes, un simple script suffit ;
- il est plus facile de se souvenir des commandes. Le cerveau mémorise mieux les mots que les déplacements dans le plan. Les enchaînements de menus et de clics ne sont pas aisés à retenir ;
- les interfaces graphiques changent en permanence : vous commenciez à être à l'aise avec GNOME 3 et hop, Ubuntu passe sur Unity... Tout est différent, il faut réapprendre à utiliser l'interface ;
- les erreurs sont moins fréquentes en ligne de commandes. Il arrive bien plus souvent de cliquer sur le bouton ***Ok*** d'une boite de dialogue sans en lire l'intitulé que de lancer une commande dangereuse du type `rm -rf /`. Avez-vous souvent lâché au mauvais endroit des fichiers que vous étiez en train de déplacer ? La souris requiert une coordination et une concentration intenses pour permettre d'atteindre une cible large de quelques pixels ;
- la ligne de commandes peut être utilisée sur n'importe quel ordinateur, quel que soit le système Unix/Linux/BSD que vous serez amené à utiliser. Eh oui même sur votre Mac, système graphique depuis ses débuts, elle est présente... Quand on vous dit que la ligne de commandes c'est l'avenir ;)
- faites un petit test : débranchez votre souris, vous pouvez toujours travailler à minima. Maintenant, rebranchez la souris et débranchez votre clavier... Alors ?

Avant d'entrer directement dans le vif du sujet et taper des commandes dans un terminal, nous allons voir quelques notions relatives au système d'exploitation et à son utilisation.

2 QUELQUES DÉFINITIONS

Si vous lisez un peu les forums relatifs à Linux, la ligne de commandes est nommée de plusieurs manières. On va retrouver les termes suivants : console, shell, bash, terminal, tty... Il s'agit d'un abus de langage, car il s'agit de choses différentes même si elles sont liées étroitement.

Dans la terminologie Unix, le terminal ou tty, pour *teletypewriter*, est un type particulier de fichier de périphérique qui met en œuvre un certain nombre de commandes (lecture, écriture, etc.). Certains terminaux sont fournis par le noyau au nom d'un dispositif matériel (clavier, écran, ligne série). D'autres terminaux, parfois nommés pseudo-ttys, sont fournis par des programmes appelés émulateurs de terminal, comme « gnome-terminal ».

Le nom « terminal » vient d'un point de vue électronique du concept, alors que le nom « console » d'un point de vue du « matériel/meuble ». La console est généralement un terminal dans le sens physique : l'écran et le clavier. Sur des systèmes Linux et BSD, la console donne accès à plusieurs terminaux (on bascule de l'un à l'autre par une combinaison de touches). Cependant, on utilisera couramment les termes terminal et console pour désigner la même chose : l'écran affichant une interface texte pour interagir avec le système. Usuellement, sur les distributions Linux, il y a 6 consoles textes disponibles (tty1 - tty6), elles sont accessibles via les raccourcis clavier [Ctrl] + [Alt] + [Fx] (x étant le numéro de console).

Par défaut, à partir de [Ctrl] + [Alt] + [F7], il ne s'agit plus de consoles textes, mais de sessions graphiques utilisant un serveur graphique comme Xorg. Si vous utilisez Linux en mode graphique, c'est cette session par défaut qui est affichée. Il est aussi possible d'utiliser plusieurs sessions graphiques différentes sur la même machine à l'aide des sessions graphiques suivantes ([F8] - [F12]).

Le shell, ou interface système est une couche logicielle qui fournit l'interface utilisateur d'un système d'exploitation. Il correspond à la couche la plus externe de ce dernier.

Il existe 2 types d'interfaces système/shell :

- en ligne de commandes (ou CLI pour *Command Line Interface/Interpreter*). Le programme fonctionne alors à partir d'instructions saisies de commandes au clavier en mode texte ;
- graphique fournissant une interface graphique pour l'utilisateur (GUI, pour *Graphical User Interface*).

Deux méthodes d'accès au shell CLI sont possibles en fonction des ressources matérielles disponibles, du système d'exploitation ou du paramétrage utilisé :

- le mode console qui affiche un shell CLI unique en plein écran, c'est l'interface homme-machine de base du système d'exploitation, celui que vous retrouverez si votre session graphique ne démarre pas ;
- le mode terminal qui émule une console et qui affiche en général le shell CLI dans une portion de l'écran.

Il existe différents shell CLI pour les systèmes Unix. Sous Linux, le plus utilisé dans les distributions actuelles est sans aucun doute « bash », pour « Bourne Again Shell » publié pour la première fois en 1989 par Brian Fox pour la Free Software Foundation. Dans la suite de ce hors-série, nous utiliserons exclusivement ce shell.

L'interpréteur de commandes/shell est un programme faisant partie des composants de base d'un système d'exploitation. Le shell est un fichier exécutable chargé d'interpréter les commandes, de les transmettre au système et de retourner le résultat. Il existe plusieurs shells, les plus courants étant « sh » (*Bourne shell*), « bash » (*Bourne again shell*), « csh » (*C Shell*), « tcsh » (*Tenex C shell*), « ksh » (*Korn shell*) et zsh (*Zero shell*). Leur nom correspond généralement au nom de l'exécutable.

À retenir

L'interpréteur de commandes est l'interface entre l'utilisateur et le système d'exploitation, d'où son nom anglais « shell » ou « coquille ». Le shell est ainsi chargé de faire l'intermédiaire entre le système d'exploitation et l'utilisateur grâce aux lignes de commandes saisies par ce dernier.

À savoir

Le hors série n° 27 de *Linux Pratique*, en pages 12 et 13 présente une frise chronologique de la création des différents shells. Je vous invite vivement à consulter aussi dans ce même document l'histoire de chacun des shells présentés.

On pourrait se demander pourquoi il existe autant d'interpréteurs de commandes différents. La raison principale est que chaque nouveau shell étend ou modifie les fonctionnalités de base des shell historiques, chaque shell ayant choisi sa spécialisation : par exemple zsh se focalise sur l'interactivité, alors que ksh lui intègre des fonctionnalités de programmation avancée. La plupart du temps, des extensions non standards ne sont pas compatibles entre les différents shells. Des raisons de licences propriétaires ont aussi poussé les développeurs à créer de nouveaux interpréteurs. On trouvera un tableau de comparaison des shells existants à l'adresse [1].

3 PRINCIPE DU SHELL

Une fois votre ordinateur mis sous tension, le BIOS (*Basic Input Output System*) de votre carte mère prend la main et va lire le *Master Boot Record* (MBR) de la première unité de stockage de masse (disque dur, clé USB, etc.). Il est chargé de lancer le système d'exploitation ou un chargeur de système du type de Grub/Lilo. Le noyau est alors démarré et lance « init » le premier processus du système d'exploitation. Celui-ci va utiliser son fichier de configuration `/.etc/initab` qui va lancer les attentes de connexion des terminaux virtuels via le programme « getty » (*get teletype*). Ce programme est chargé de détecter une demande de connexion. Il demande un identifiant qu'il renvoie à l'application « login » afin d'identifier l'utilisateur. Une fois le mot de passe entré et vérifié, l'utilisateur est authentifié et l'interpréteur de commandes est lancé (ce dernier est configuré dans le fichier `/.etc/passwd`).

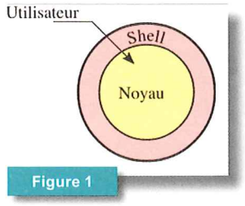


Figure 1

Le shell (ou « coquille » en français) est maintenant l'interface obligée de l'utilisateur pour l'accès au noyau du système d'exploitation (figure 1).

Lorsque l'utilisateur entre une commande qui est exécutée, un processus est créé et ouvre 3 flux de données (figure 2) :

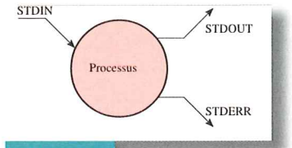


Figure 2

- le premier est l'entrée standard (STDIN = *standard input*), par défaut c'est ce que vous saisissez au clavier ;
- le deuxième est la sortie standard (STDOUT = *standard output*), par défaut, c'est l'écran ou plus précisément le shell ;
- le troisième est la sortie standard des messages d'erreurs consécutifs à une commande (STDERR = *standard error*), qui est généralement par défaut l'écran.

Chacun de ces flux de données est identifié par un numéro descripteur, 0 pour l'entrée standard, 1 pour la sortie standard et 2 pour la sortie standard des messages d'erreur. Ces descripteurs seront utiles pour les redirections (voir page 68).

Voyons comment est organisé le système de fichiers Unix/Linux.

4 LE SYSTÈME DE FICHIERS SOUS UNIX

Les données sont présentées à l'utilisateur et aux programmes selon une organisation structurée, sous la forme de répertoires et de fichiers (figure 3). Pour pouvoir stocker ces données structurées sur un périphérique, il faut utiliser un format qui les représente sous la forme d'une succession de blocs de données : c'est ce qu'on appelle un système de fichiers. Les systèmes de fichiers les plus courants sont FAT16/32 (*File Allocation Table*, disquettes et clefs USB), NTFS (*New Technology File System*) (Windows), Ext2, Ext3 et Ext4 (Linux, *Extended Filesystem*), ISO 9660 (CD) et UDF (DVD).

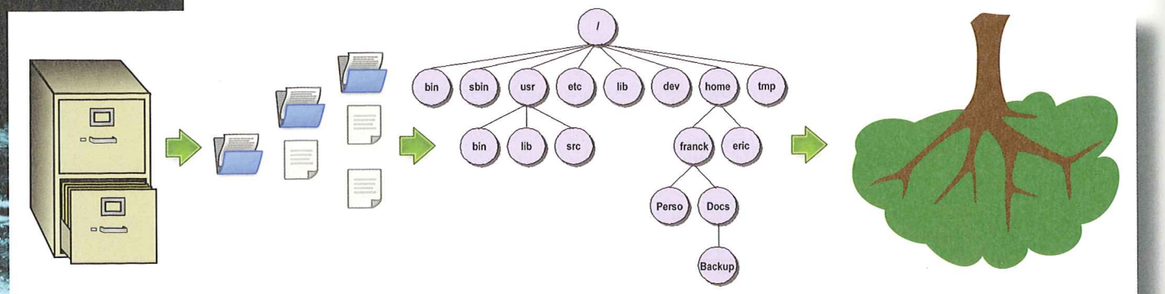


Figure 3

Contrairement au système de fichiers Windows, il n'existe pas de lecteurs `A:`, `C:`, etc. L'entrée du système de fichiers se situe à la racine et est notée « `/` ». La principale particularité du système de fichiers sous Linux est que tout est représenté sous la forme de fichiers de types différents. Les plus communs sont les fichiers texte ou les fichiers exécutables (programmes) nommés fichiers réguliers. Les répertoires (dossiers ou *directory*) sont aussi des fichiers qui contiennent d'autres fichiers et permettent l'organisation sous la forme d'une arborescence.

Il existe des fichiers périphériques qui se trouvent dans le répertoire `/dev` et qui correspondent aux disques durs, clés USB, imprimantes, carte son, mémoire, etc. Les périphériques sont eux aussi reconnus comme des fichiers. On retrouve aussi les fichiers de type liens physiques et symboliques dont nous verrons l'utilité en page 56.

L'organisation des fichiers respecte une architecture arborescente plus ou moins standard. Cette arborescence peut se développer sur une ou plusieurs partitions, sur un ou plusieurs disques.

Toute l'architecture de l'arborescence est contrôlée par l'administrateur « root », mis à part les répertoires contenus dans le répertoire `/home` et qui appartiennent aux différents utilisateurs du système.

L'intégration d'un système de fichiers externe (clé USB, disque externe) ou supplémentaire (ajout de disque dur, par exemple) dans l'arborescence existante se nomme le montage. Cette opération associe un répertoire du système de fichiers existant à la racine du système de fichiers à intégrer.
Un système de fichiers peut être monté dès le démarrage du système, mais aussi, automatiquement ou manuellement, après le démarrage.
L'opération inverse s'appelle le « démontage » et consiste à dissocier le répertoire de la racine du système.

La figure 4 présente le mécanisme de montage.

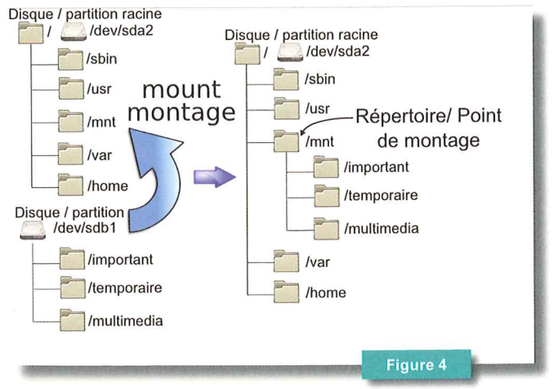


Figure 4

À savoir

Sous un système UNIX, un fichier, quel que soit son type est identifié par un numéro appelé « numéro d'inode », qu'on pourrait traduire en français par « i-noeud ». Ainsi, derrière la façade du shell, un répertoire n'est qu'un fichier, identifié aussi par un inode, contenant une liste d'inodes représentant chacun un fichier.

À retenir

La différence entre un lien physique et symbolique se trouve au niveau de l'inode, un lien symbolique n'a pas d'inode propre, il a l'inode du fichier vers lequel il pointe. Par contre, un lien physique possède son propre inode. Pour connaître le numéro d'inode d'un fichier, vous pouvez taper : `ls -i mon-fichier`.

Dressons maintenant une liste des principaux répertoires ainsi que leur rôle :

`/`	Répertoire « racine », point d'entrée du système de fichiers
`/bin`	Répertoire contenant les exécutables de base, comme par exemple `cp`, `mv`, `ls`, etc.
`/boot`	Répertoire contenant les fichiers nécessaires au démarrage (Grub, Lilo) et le noyau linux
`/dev`	Répertoire contenant des fichiers spéciaux nommés devices qui permettent le lien avec les périphériques de la machine
`/,etc`	Répertoire contenant les fichiers de configuration du système
`/home`	Répertoire contenant les fichiers personnels des utilisateurs (un sous-répertoire par utilisateur)
`/lib`	Répertoire contenant les bibliothèques et les modules du noyau (`/lib/modules`)
`/mnt`	Contient les points de montage temporaires des systèmes de fichiers
`/opt`	Répertoire permettant l'installation d'applications extérieures
`/root`	Répertoire personnel de l'administrateur
`/sbin`	Répertoire contenant les exécutables destinés à l'administration du système
`/tmp`	Répertoire contenant des fichiers temporaires utilisés par certains programmes
`/usr`	Répertoire contenant les exécutables des programmes (`/usr/bin` et `/usr/sbin`), la documentation (`/usr/doc`), et les programmes pour le serveur graphique (`/usr/X11R6`)
`/var`	Répertoire contenant les fichiers qui servent à la maintenance du système (les fichiers de journaux notamment dans `/var/log`)
`/media`	Répertoire contenant les « points de montage » des médias usuels : CD, DVD, disquette, clef USB...
`/dev/null`	On peut envoyer une infinité de données à ce périphérique qui les ignorera...
`/dev/zero`	On peut lire une infinité de zéros depuis ce périphérique
`/dev/random`	On peut lire des nombres aléatoires depuis ce périphérique

Dans son répertoire de connexion (`/home/login`), l'utilisateur peut effectuer les opérations qu'il désire alors qu'il lui est interdit de modifier la plupart des autres objets de l'arborescence du système de fichiers. En effet, la notion de droit est présente dans tout le système d'exploitation. Nous y reviendrons en page 57. ▌

Récapitulatif

- ⇨ La ligne de commandes est l'essence même du système Unix et donc Linux. Il serait dommage de vous en passer !
- ⇨ Son utilisation peut vous permettre de gagner du temps, mais aussi de mieux maîtriser votre système. Elle vous permettra de vous dépanner rapidement en cas de problème.
- ⇨ Il existe des shells différents avec des fonctionnalités spécifiques.
- ⇨ Le système de fichiers Unix est composé exclusivement de fichiers de types différents. Il est organisé sous la forme d'une arborescence qui possède des répertoires avec des rôles bien définis.

RÉFÉRENCES

[1] **http://en.wikipedia.org/wiki/Comparison_of_command_shells**

[2] **http://en.wikipedia.org/wiki/List_of_terminal_emulators**

AVANT DE DÉBUTER...

UTILISER LA LIGNE DE COMMANDES SUR LES SYSTÈMES LINUX, MAC OS X ET WINDOWS

Yann Morère

La ligne de commandes n'est pas exclusivement réservée aux systèmes Unix et Linux. On peut la retrouver sur d'autres systèmes comme Mac OS X et Windows. Voyons comment l'utiliser sur ces plateformes.

INTRODUCTION

Même si vous n'utilisez pas de distribution Linux, vous pouvez tirer avantage de l'utilisation de la ligne de commandes sur d'autres systèmes comme Mac OS X ou Windows tout en conservant les utilitaires et la syntaxe décrite dans les pages de ce mook. Si l'utilisation de la ligne de commandes est immédiate sur Linux et Mac OS X, sous Windows il faudra installer des logiciels supplémentaires.

Voyons maintenant comment accéder à l'interpréteur de commandes sur ces 3 principaux systèmes d'exploitation.

1 UTILISER LA LIGNE DE COMMANDES SOUS LINUX

Sur ce système, vous avez 2 possibilités :

- un accès au terminal exclusivement en mode texte par l'intermédiaire des 6 premiers terminaux textes disponibles par défaut ;
- un accès à un terminal en mode graphique par l'intermédiaire d'un émulateur de terminal.

1.1 Le terminal mode texte

Dans le chapitre précédent, nous avons rapidement mentionné qu'il était possible d'obtenir un shell à partir d'un terminal en mode texte sur son ordinateur en basculant sur les « consoles » texte par le raccourci [CTRL] + [ALT] + [Fx], avec x pouvant prendre les valeurs de 1 à 6. Ensuite, une fois dans un terminal texte, il est possible de basculer entre les 6 terminaux par le raccourci [ALT] + [Fx] avec x compris entre 1 et 6.

Comme pour le mode graphique, il faut s'authentifier afin de pouvoir utiliser l'ordinateur. Attention, la saisie du mot de passe se fait ici en aveugle et aucun retour d'information visuelle des touches que vous tapez n'apparaîtra (figures 1 et 2).

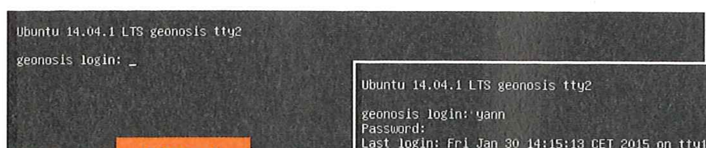


Figure 1

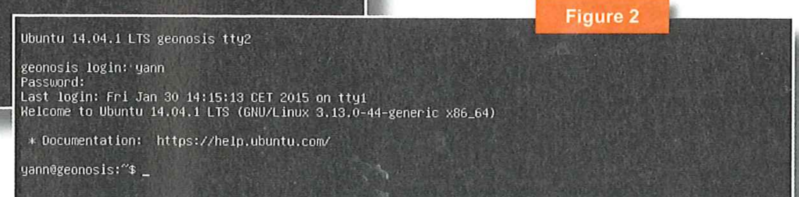


Figure 2

Figure 3

Il est bien sûr possible de se connecter simultanément sur plusieurs terminaux pour réaliser des travaux différents. Ensuite, il suffit d'entrer des commandes Unix valides (figure 3). On se déconnecte à l'aide de la commande **exit** ou encore le raccourci [CRTL] + [D].

Si le « copier/coller » à la souris vous manque dans cet univers dédié au clavier, la communauté a pensé à vous est a créé l'utilitaire **gpm** (*general purpose mouse*). Ce programme fournit le support de la souris pour les applications utilisées en mode texte : VI, Emacs, Mutt, Midnight Commander, etc. On l'installe facilement à l'aide de la commande suivante ou via son gestionnaire de paquets habituel :

Terminal

```
$ sudo apt-get install gpm
```

Normalement, le service est démarré et la souris est maintenant fonctionnelle dans votre terminal. Le copier/coller à l'aide du bouton de la molette est de nouveau actif. Voyons maintenant l'accès à la console en mode graphique.

1.2 Les émulateurs de terminal en mode graphique

Sur une machine utilisant une interface graphique, il est bien plus simple d'utiliser un émulateur de terminal aussi appelé console virtuelle ou terminal virtuel. Il s'agit d'un logiciel qui émule le fonctionnement d'un terminal informatique dans une fenêtre graphique. Ceci permet d'utiliser plusieurs terminaux sur un seul moniteur d'ordinateur.

Il en existe de nombreux, la page [2] en dresse la liste pour les principaux systèmes d'exploitation. Sous Linux, les plus usités sont : xterm, gnome-terminal, aterm, rxvt, konsole et xfce4-terminal.

Dans la suite, nous utiliserons gnome-terminal, l'émulateur développé pour l'environnement de bureau GNOME, disponible par défaut sur la distribution Ubuntu.

Pour lancer rapidement gnome-terminal, il suffit d'utiliser le raccourci [CTRL] + [ALT] + [T] ou encore [ALT] + [F2], qui ouvre la fenêtre ***Lancer une application***, puis on saisit « gnome-terminal ». Vous pouvez aussi passer par le menu ***Applications > Accessoires > Terminal***.

Vous avez maintenant accès à l'interpréteur de commandes. Le curseur qui clignote vous indique que le système attend vos commandes.

Pour avoir un second terminal, il est possible d'en ouvrir un dans une nouvelle fenêtre, mais il est bien plus simple et pratique de le créer dans un nouvel onglet de l'émulateur existant. Cela se fait par l'intermédiaire du raccourci [CTRL] + [SHIFT] + [T] ou par l'intermédiaire du menu ***Fichier > Ouvrir un Onglet > default***. Un clic droit dans le terminal fait apparaître un menu contextuel qui permet de lancer un nouveau terminal ou de créer un nouvel onglet.

Il est possible de renommer les titres de chaque onglet afin d'éviter de confondre les différents terminaux. Pour cela, un simple clic droit sur le titre fait apparaître le menu contextuel qui permet le renommage (figure 4, page suivante). Il est aussi possible de configurer les raccourcis clavier de votre terminal (figure 5, page suivante).

À savoir

Concernant les raccourcis clavier, on notera que les éternels raccourcis de copier/coller sont devenus [MAJ] + [CTRL] + [C] et [MAJ] + [CTRL] + [V], car dans un shell, le [CTRL] + [C] sert à stopper une commande et ne peut donc pas être utilisé pour la copie.

À savoir

Il est possible de réaliser rapidement des copier/coller de texte à l'intérieur du terminal à l'aide de la souris. On réalise la sélection du texte à copier à l'aide du bouton gauche et un glisser, on colle ensuite le texte sélectionné à l'aide du clic de la molette de votre souris. Cette fonctionnalité existe depuis les débuts du serveur X, les souris à l'époque possédaient 3 boutons.

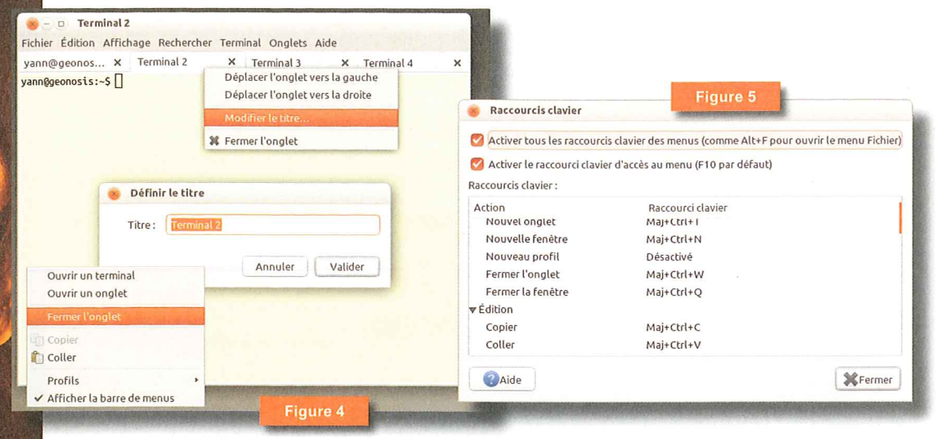


Figure 4

Figure 5

À retenir

On peut rapidement augmenter/diminuer la taille des polices de caractères du terminal, par les raccourcis [CTRL] + [+] et [CRTL] + [-].

Il est bien sûr possible d'adapter suivant vos préférences l'apparence de votre terminal en termes de couleurs, d'image de fond, de transparence, de taille de police de caractères, de forme du curseur, de position de la barre de défilement, etc. Ces configurations spécifiques peuvent être stockées sous la forme d'un profil. On peut simplement modifier le profil par défaut ou en créer de nouveaux. On accède à ces préférences par le menu ***Édition > Préférences du profil***. On crée un nouveau profil à l'aide du menu ***Fichier > Nouveau Profil***. L'utilisation de ces profils permet d'obtenir un affichage adapté à chaque tâche que vous effectuez en ligne de commandes (figures 6 et 7).

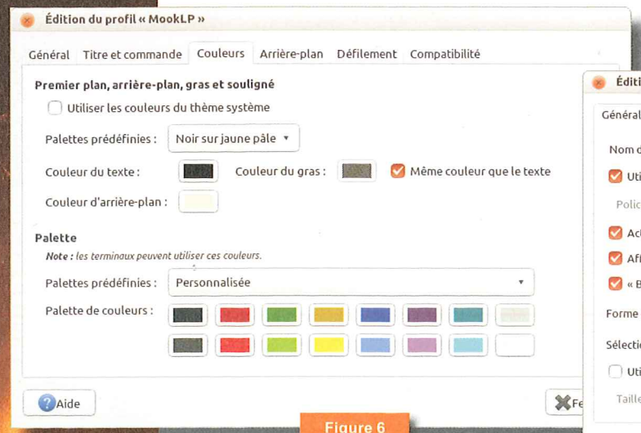


Figure 6

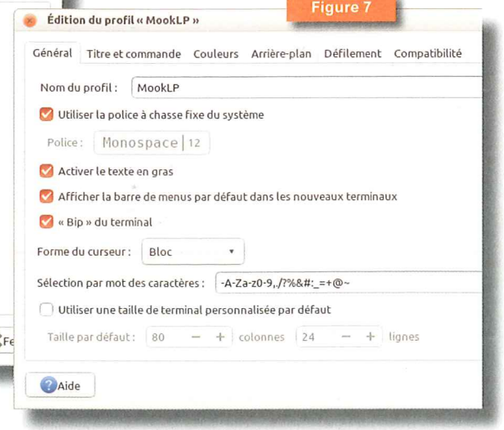


Figure 7

Il existe d'autres types d'émulateurs avec des spécificités graphiques :

- Terminator est un terminal virtuel permettant d'envoyer les lignes de commandes saisies au clavier sur un groupe de terminaux, de diviser horizontalement et verticalement la fenêtre pour disposer de plusieurs terminaux, de lancer des commandes automatiquement au démarrage du terminal ;

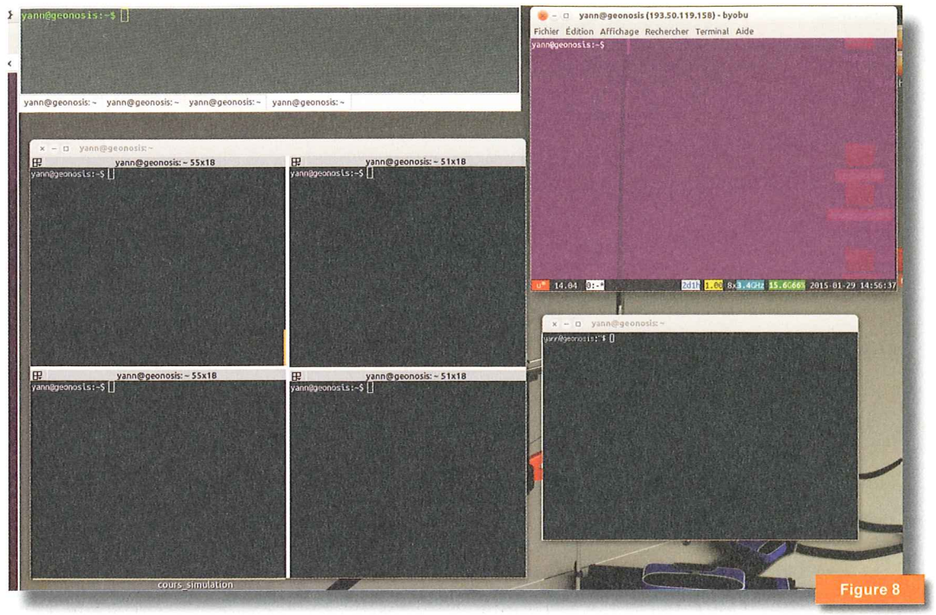


Figure 8

- Tilda est un terminal « drop down » qui s'affiche en se déroulant vers le bas lorsque l'on appuie sur une touche. Il est simple, mais très fonctionnel et gère les onglets. Il permet un accès très rapide à la ligne de commandes depuis votre interface graphique ;
- Byobu est une amélioration de GNU Screen, le multiplexeur de terminaux en mode texte. Il permet l'affichage d'informations système et de notifications sur deux lignes dans le bas d'une session de terminal.

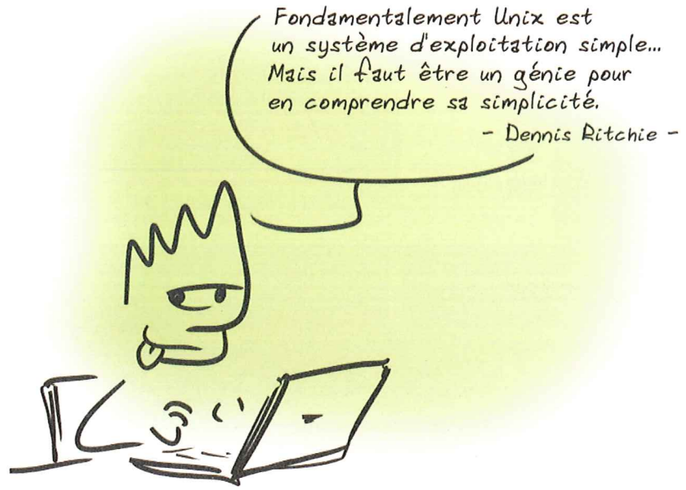


Illustration basée sur le GKND Creator (https://framalab.org/gknd-creator).
Œuvre originale de Simon « Gee » Giraudot.
Licence Creative Commons By-Sa.

2 UTILISER LA LIGNE DE COMMANDES SOUS MAC OS X

Historiquement, le système d'exploitation d'Apple était purement graphique et il n'y avait pas d'accès au moindre mode texte. La ligne de commandes a fait son apparition avec l'arrivée de Mac OS X. Ce dernier est basé sur Darwin [2],un système d'exploitation libre et gratuit construit autour du noyau XNU et développé notamment par Apple. Ce système est dérivé de NeXTSTEP et de FreeBSD. Il est distribué sous licence APSL (*Apple Public Source License*), certifiée par la FSF (*Free Software Foundation*).

Ce qui n'est pas libre dans Mac OS X, c'est l'interface Aqua ainsi que le moteur graphique Quartz.

NOTE

Cette ouverture vers le logiciel libre n'est pas innocente de la part d'Apple qui compte ainsi attirer des développeurs open source utilisant Linux ou BSD.

Bien sûr ici, pas de mode texte exclusif, on utilisera l'émulateur de terminal Terminal.app. Cette application se trouve dans le dossier `/Applications/Utilitaires/`, quand vous la lancez, une fenêtre s'ouvre et affiche un interpréteur de commandes bash. Comme pour les terminaux Linux, il est possible de configurer le rendu de la fenêtre de terminal (figure 9) par l'intermédiaire du menu ***Terminal > Préférences***. Il est aussi possible d'ouvrir plusieurs terminaux dans des onglets (raccourci [⌘] + [T]), mais aussi de scinder le terminal en 2 horizontalement (raccourci [⌘] + [D]).

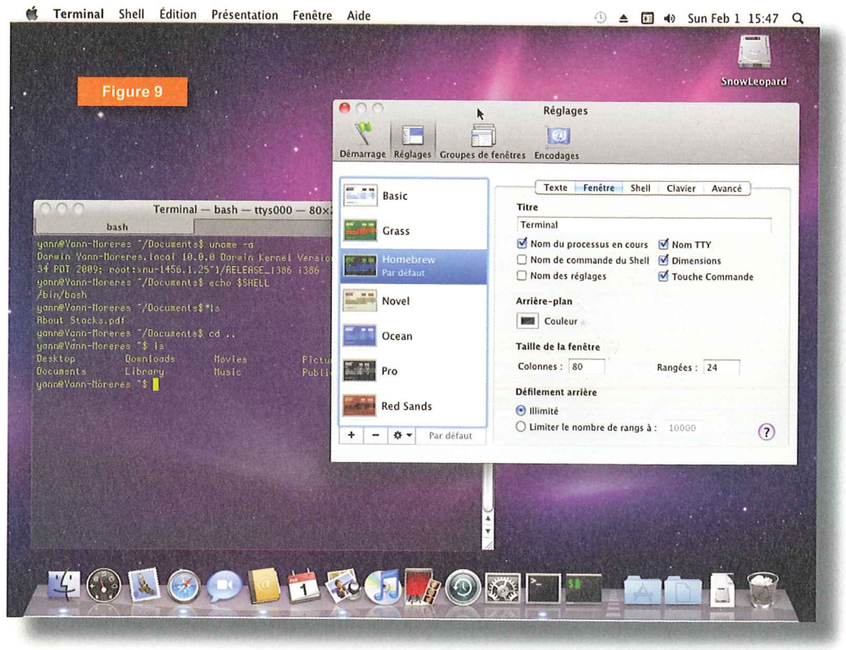


Figure 9

Nous sommes en présence d'un véritable système Unix et de son shell associé, mais attention le système de fichiers et son arborescence sont différents d'un système Linux standard. Ainsi, la plupart des manipulations d'administration système Linux ne peuvent pas être utilisées sur Mac OS X.

Par exemple, le fichier de configuration par défaut du shell bash se trouve dans le répertoire `/private/etc/bashrc` :

Terminal

```
yann@SnowLeopard ~$ more /private/etc/bashrc
# System-wide .bashrc file for interactive bash(1)
shells.
if [ -z "$PS1" ]; then
   return
fi
# PS1='\h:\W \u\$ '
PS1='\u@\h \w\$ '
# Make bash check its window size after a process
completes
shopt -s checkwinsize
```

De même, pour réaliser la recherche d'un fichier dans l'arborescence on peut utiliser la commande `locate` bien connue, mais la base de données des fichiers doit être préalablement créée à l'aide de la commande suivante :

Terminal

```
$ sudo launchctl load -w /System/Library/
LaunchDaemons/com.apple.locate.plist
```

alors que sous Linux, il suffit d'un simple :

Terminal

```
$ sudo updatdb
```

De manière identique, le changement du nom de l'hôte sous Mac OS X se fait à l'aide de la commande :

Terminal

```
sudo scutil --set HostName SnowLeopard
```

alors que sous Linux, il suffit simplement d'éditer le fichier `/etc/hostname`.

Ce système possède donc ses spécificités et ses propres outils de configuration qu'il faudra prendre en main si vous désirez aller plus loin avec la ligne de commandes.

Cependant, les outils de base de gestion du système de fichiers et de gestion des droits sont identiques. Toutes les manipulations présentées dans la suite pourront être réalisées à l'aide des terminaux Mac OS X.

Comme pour Linux, il existe des émulateurs de terminal alternatifs pour Mac OS X. La principale différence avec Linux est que certains sont payants :

- iTerm2 (gratuit, visible en figure 10, page suivante) : un émulateur fenêtré ou « dropdown » qui permet la division horizontale et verticale, une fonction de recherche est intégrée ainsi que l'auto-complétion, un historique de copier/coller, etc. [3] ;
- Terminator [4] : il s'agit du même émulateur que sous Linux. Sur Mac OS X, il faudra l'installer à l'aide de Fink ;

À savoir

Le projet Fink a pour but de rendre disponibles les logiciels open source Unix sur Darwin et Mac OS X. Ils sont modifiés afin d'être recompilés pour le système d'exploitation de destination. Fink utilise des outils de gestion de paquets binaires Debian, tels `dpkg` et `apt-get`. Vous avez le choix entre le téléchargement des paquets binaires précompilés ou la construction des paquets à partir des sources.

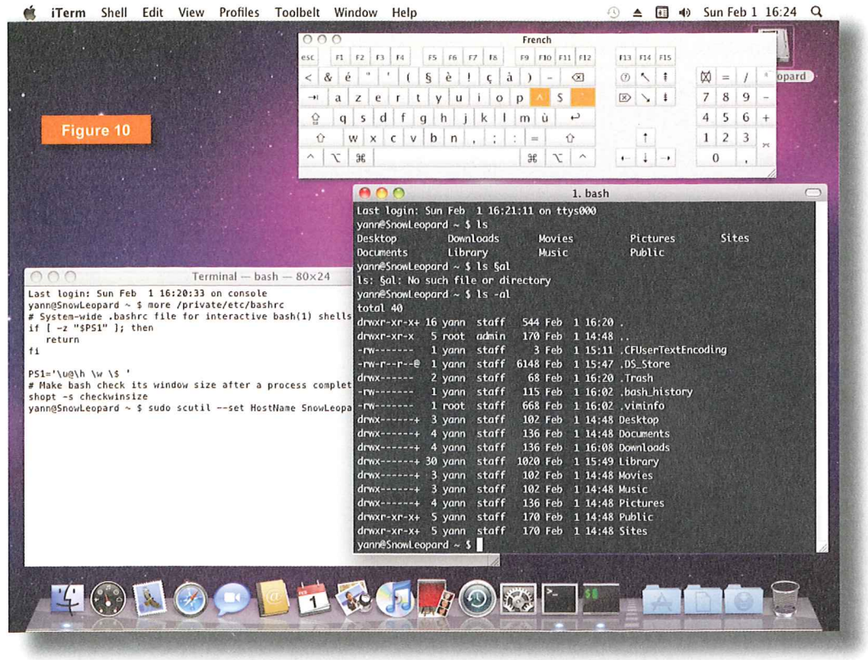


Figure 10

- Cathode [5] : son seul intérêt réside dans l'affichage « vintage » d'une ancienne console avec l'écran cathodique. C'est sûr, on va l'utiliser pour son côté pratique :) ;
- MacTerm [6] : ce dernier possède une chose amusante : la possibilité de faire des macros pour votre ligne de commandes... comme si les scripts n'existaient pas !

Voyons maintenant comment profiter d'un interpréteur de commandes sous Windows.

3 UTILISER LA LIGNE DE COMMANDES SOUS WINDOWS

3.1 Aperçu du terminal DOS

La ligne de commandes est à la base de Windows. En effet, les premières versions de Windows (1.0, 2.0, 3.0 et 3.11 qui embarquaient le support du réseau) étaient la couche graphique au-dessus du système d'exploitation DOS (*Disk Operating System*), à la manière de Xorg pour Linux. Les versions de Windows ont évolué et masqué cet accès au terminal. Cependant, il est toujours présent : l'application se nomme « Invite de commandes », la traduction littérale de « prompt ». On lance cette application à l'aide du menu ***Démarrer > Tous les programmes > Accessoires > Invite de commandes*** ou plus facilement, on entre la commande `cmd` dans la zone de recherche des programmes dans le menu ***Démarrer***. On obtient la fenêtre « Invite de commandes » de la figure 11. Il est alors possible d'interagir avec le système comme on le fait sous Unix.

Ce terminal utilise le shell DOS qui est assez différent des shells Unix même si quelques commandes sont identiques (`cd`, `mkdir`, `echo finger`, etc.). Le nombre de commandes est limité [7] et il est bien moins puissant que les interpréteurs de commande Unix en ce qui concerne la programmation.

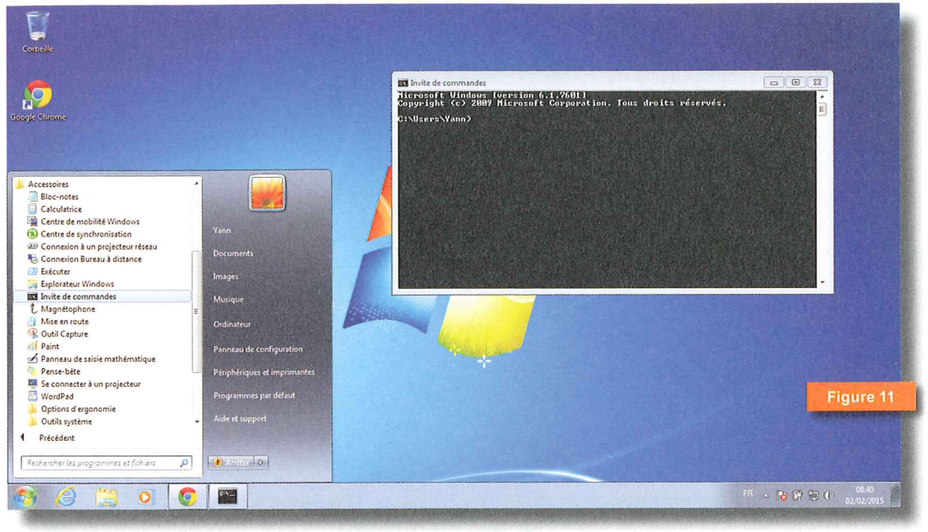


Figure 11

Cependant, c'est cet outil que vous rencontrerez si l'interface graphique de votre Windows ne démarre plus et qu'il lance une « console de restauration ». Je vous conseille donc d'y jeter un œil attentif si vous utilisez ce système d'exploitation.

Le but ici étant de vous former à la ligne de commandes Unix, nous n'utiliserons pas le shell DOS, mais bien un shell Unix. Il nous faut pour cela installer des outils supplémentaires par l'intermédiaire du projet Cygwin.

3.2 Cygwin : retrouvez la philosophie Linux sous Windows

Le projet Cygwin [8] est une large collection de logiciels libres qui permettent de fournir des fonctionnalités similaires à une distribution Linux sous Windows. Il se présente sous la forme d'une DLL (*Dynamic Linked Librairie*, librairie partagée sous Windows : `cygwin1.dll`). Mais attention, cela ne transforme pas votre Windows en une distribution Linux. Vous ne pourrez pas lancer d'application native Linux directement. De même, les spécificités des systèmes d'exploitation Unix comme les signaux, ou le système de fichiers ne pourront être exploités directement sous Windows.

3.3 Installation de Cygwin

Commencez par télécharger le programme d'installation adapté à votre système d'exploitation (32 ou 64 bits) à partir de l'adresse [8], rubrique ***Install Cygwin***. Ensuite, exécutez le programme d'installation (figure 12) et appuyez sur le bouton ***Suivant***.

Figure 12

On choisit le mode d'installation : ***Install from internet*** (figure 13).

On poursuit en choisissant le répertoire d'installation (figure 14). On peut ici laisser les choix par défaut : ***C:\cygwin*** et ***All Users***.

On sélectionne le répertoire où Cygwin (`setup.exe`) va conserver les informations d'installation pour les éventuelles mises à jour et nouvelles installations (figure 15).

L'étape suivante concerne le type de connexion : ***Direct Connection*** par défaut (figure 16, ci-contre).

Enfin, on choisit un serveur dans la liste des miroirs (figure 17).

Vient ensuite la sélection des utilitaires à installer. Les outils de base sont déjà présélectionnés (figure 18).

Nous allons en ajouter quelques-uns qui pourraient s'avérer utiles :

- Editors : `nedit`, `nano` ;
- Net : `nc`, `ncftp`, `curl`, `openssh`, `ping` ;
- X11 : `xinit`, `font-adobe-dpi100`, `font-adobe-dpi75`, `font-beat-stream-dpi100`, `font-beat-stream-dpi75`, `xclock`, `xeyes`, `xlogo`, `xmessage`, `xhost`, `xmore`, `xloadimage` ;
- Shells : `bash-completion`, `xterm`, `rxvt`, `rxvt-unicode`.

Pour sélectionner un paquet à installer, déroulez les bons sous-menus (en cliquant dessus), cherchez le paquet, puis cliquez dans la colonne ***New*** pour (dé)sélectionner le paquet.

Finalement, on clique sur ***Suivant*** pour lancer l'installation.

L'outil intégré de gestion des dépendances (figure 19, page 26) entre les paquets vous indique les programmes supplémentaires qui vont être installés et ensuite le téléchargement commence. Finalement, le programme d'installation vous propose de créer une icône sur le bureau ainsi qu'une entrée dans le menu ***Démarrer*** (figure 20, page suivante).

Figure 13

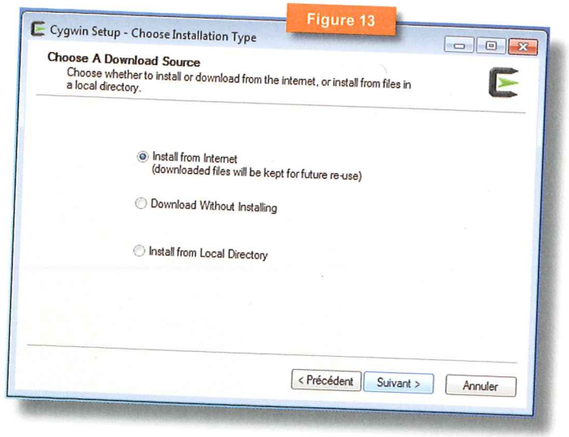


Figure 14

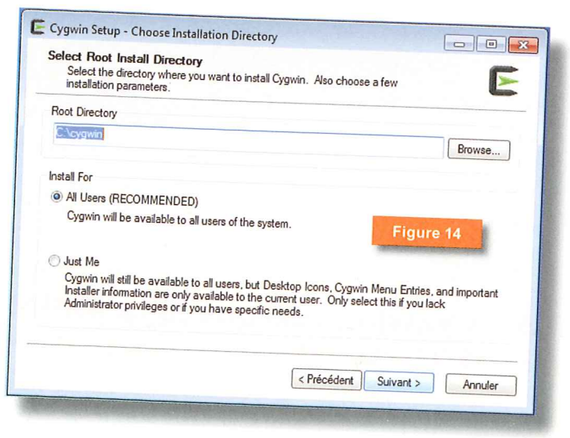


Figure 15

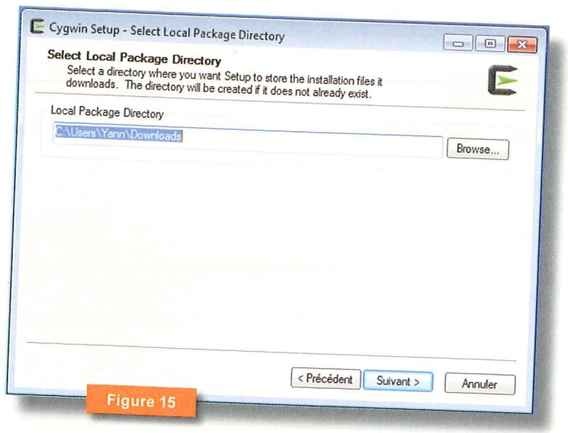


Figure 16

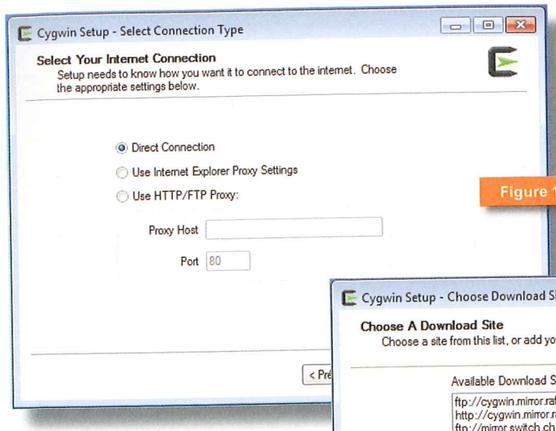


Figure 17

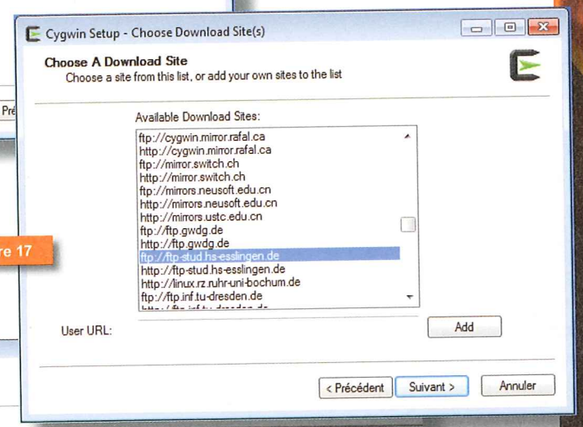
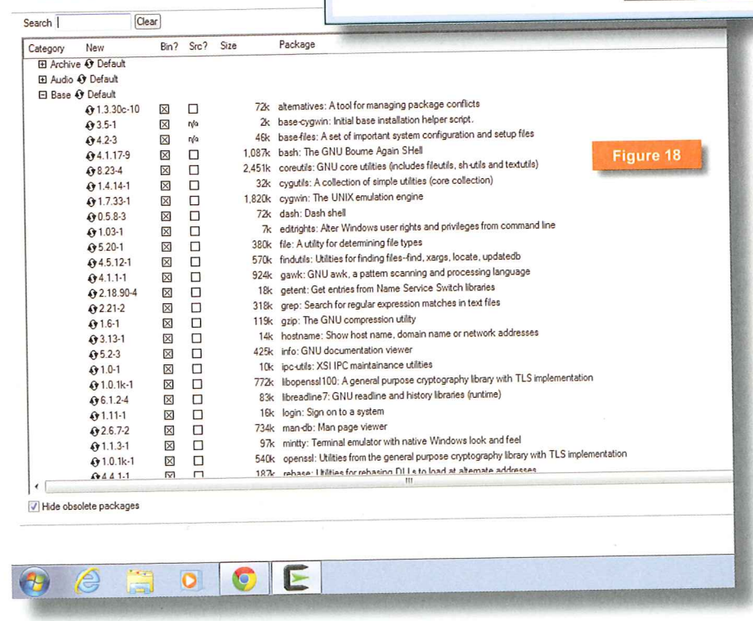


Figure 18



À retenir

Pour faire une mise à jour des outils Cygwin, il suffit de relancer le programme `setup.exe` et faire la/les (dé)sélection(s) souhaitée(s), puis finaliser l'installation.

À savoir

Le programme d'installation a créé dans mon répertoire `c:\user\yann\downloads` (répertoire de lancement du programme `setup.exe`) un répertoire `ftp%.....cygwin`. Ne l'effacez pas, car il conserve les paramètres de votre configuration. Si vous relancez `setup.exe`, vous pouvez à nouveau ajouter ou enlever des paquets. Il met également à jour, si besoin, les paquets installés. Par contre, dans le dossier `ftp%.....cygwin`, on trouve un dossier `release` qui contient toutes les archives des paquets qui ont été installés dans Cygwin : ceux-ci occupent une place inutile, vous pouvez effacer ce dossier en toute confiance.

On choisit le mode d'installation : ***Install from internet*** (figure 13).

On poursuit en choisissant le répertoire d'installation (figure 14). On peut ici laisser les choix par défaut : ***C:\cygwin*** et ***All Users***.

On sélectionne le répertoire où Cygwin (`setup.exe`) va conserver les informations d'installation pour les éventuelles mises à jour et nouvelles installations (figure 15).

L'étape suivante concerne le type de connexion : ***Direct Connection*** par défaut (figure 16, ci-contre).

Enfin, on choisit un serveur dans la liste des miroirs (figure 17).

Vient ensuite la sélection des utilitaires à installer. Les outils de base sont déjà présélectionnés (figure 18).

Nous allons en ajouter quelques-uns qui pourraient s'avérer utiles :

- Editors : `nedit`, `nano` ;
- Net : `nc`, `ncftp`, `curl`, `openssh`, `ping` ;
- X11 : `xinit`, `font-adobe-dpi100`, `font-adobe-dpi75`, `font-beat-stream-dpi100`, `font-beat-stream-dpi75`, `xclock`, `xeyes`, `xlogo`, `xmessage`, `xhost`, `xmore`, `xloadimage` ;
- Shells : `bash-completion`, `xterm`, `rxvt`, `rxvt-unicode`.

Pour sélectionner un paquet à installer, déroulez les bons sous-menus (en cliquant dessus), cherchez le paquet, puis cliquez dans la colonne ***New*** pour (dé)sélectionner le paquet.

Finalement, on clique sur ***Suivant*** pour lancer l'installation.

L'outil intégré de gestion des dépendances (figure 19, page 26) entre les paquets vous indique les programmes supplémentaires qui vont être installés et ensuite le téléchargement commence. Finalement, le programme d'installation vous propose de créer une icône sur le bureau ainsi qu'une entrée dans le menu ***Démarrer*** (figure 20, page suivante).

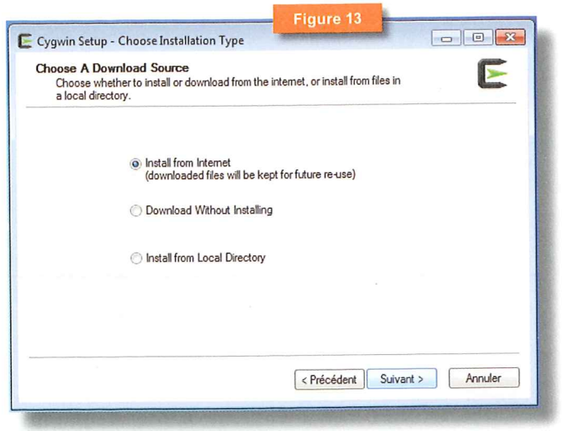


Figure 13

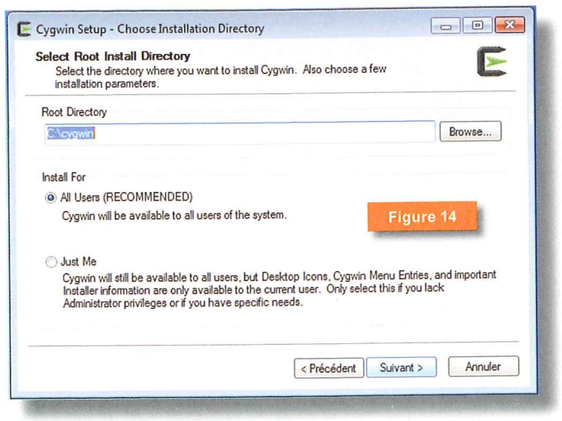


Figure 14

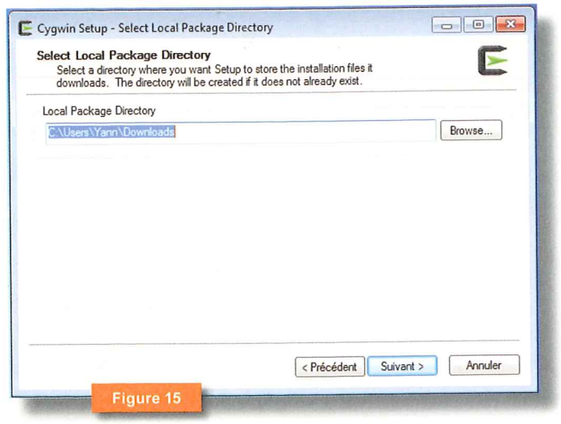


Figure 15

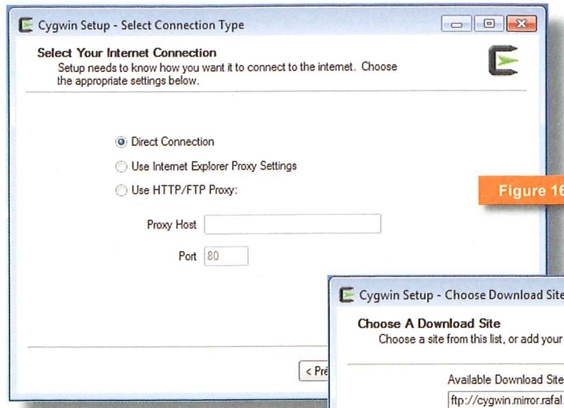


Figure 16

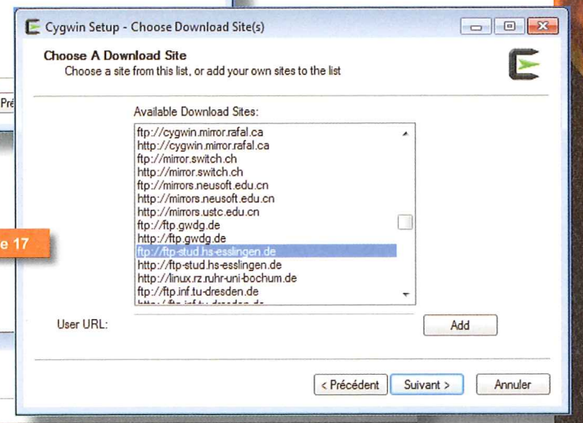


Figure 17

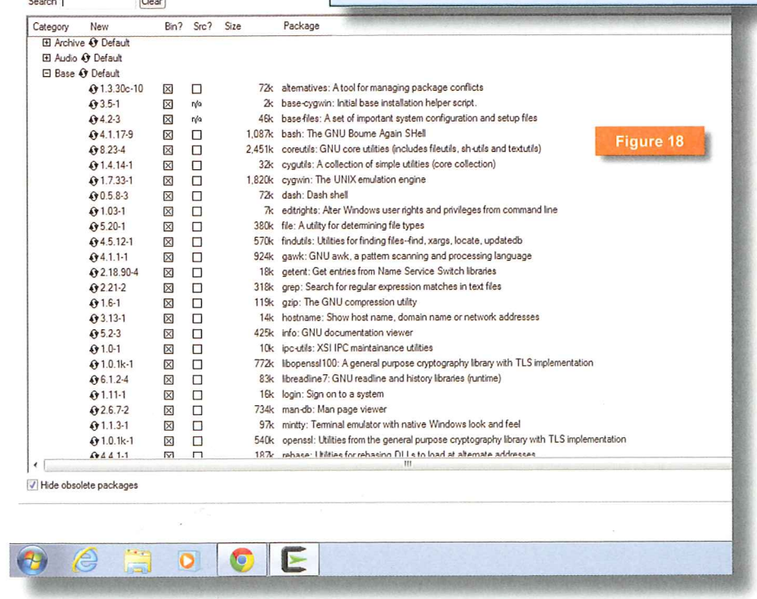


Figure 18

À retenir

Pour faire une mise à jour des outils Cygwin, il suffit de relancer le programme `setup.exe` et faire la/les (dé)sélection(s) souhaitée(s), puis finaliser l'installation.

À savoir

Le programme d'installation a créé dans mon répertoire `c:\user\yann\downloads` (répertoire de lancement du programme `setup.exe`) un répertoire `ftp%.....cygwin`. Ne l'effacez pas, car il conserve les paramètres de votre configuration. Si vous relancez `setup.exe`, vous pouvez à nouveau ajouter ou enlever des paquets. Il met également à jour, si besoin, les paquets installés. Par contre, dans le dossier `ftp%.....cygwin`, on trouve un dossier `release` qui contient toutes les archives des paquets qui ont été installés dans Cygwin : ceux-ci occupent une place inutile, vous pouvez effacer ce dossier en toute confiance.

C'est terminé, nous allons pouvoir passer à l'utilisation.

3.4 Utilisation de Cygwin

Si vous avez demandé l'installation d'une icône sur le bureau, il suffit de double-cliquer sur « Cygwin Terminal » pour lancer un interpréteur de commandes bash (figure 20). Le cas échéant, il faudra passer par le menu ***Démarrer > Tous les programmes > Cygwin > Cygwin Terminal***.

Vous avez maintenant accès à un shell bash en mode texte sur votre système Windows.

Les plus attentifs auront remarqué que je vous ai fait installer les paquets supplémentaires pour, notamment, le support de certains protocoles réseau et de X, le serveur graphique d'Unix.

Pour aller un peu plus loin, voyons comment lancer des applications graphiques Unix dans notre Windows.

Pour cela, il faut commencer par lancer le serveur graphique. Ce dernier se trouve dans le menu ***Démarrer > Tous les programmes > Cygwin-X > XWin Serveur***.

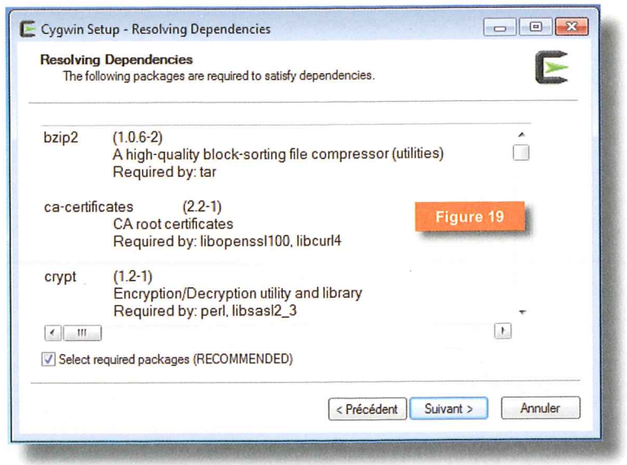


Figure 19

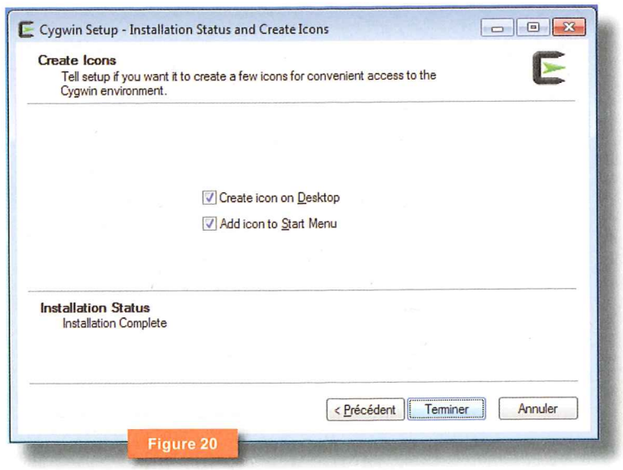


Figure 20

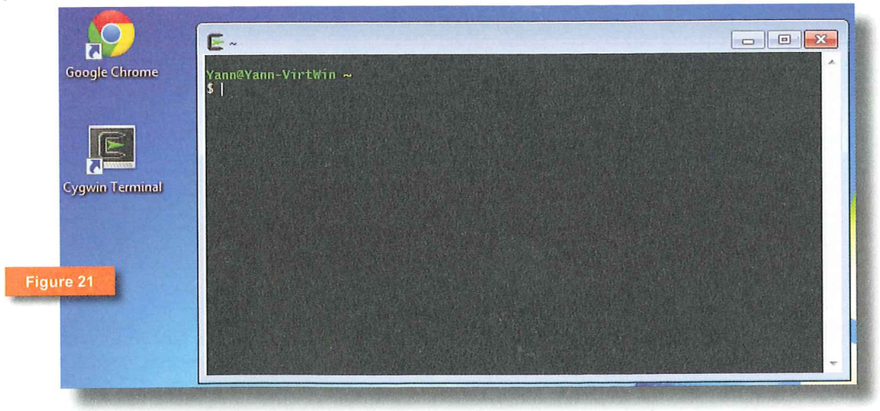


Figure 21

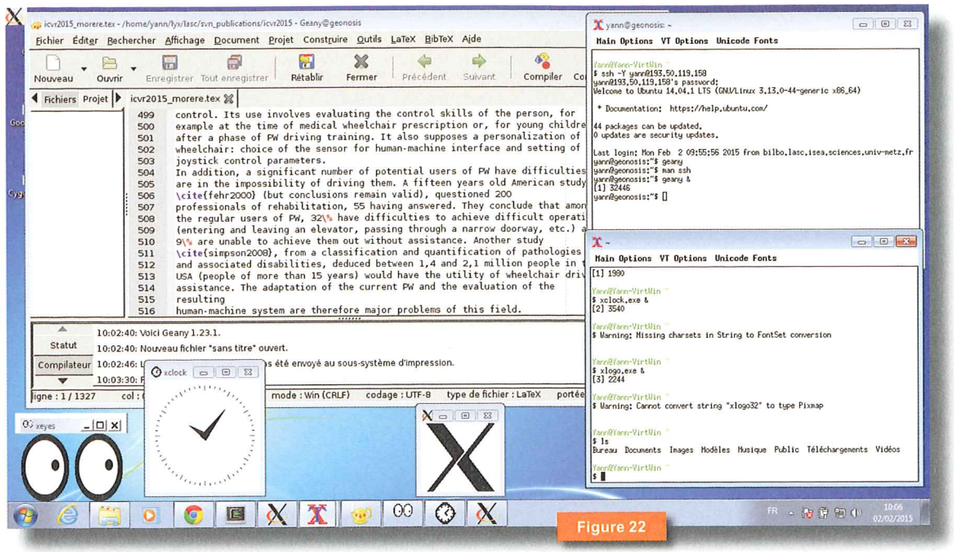


Figure 22

Normalement, une icône X doit apparaître dans votre zone de notification de Windows et un interpréteur de commandes bash dans un terminal texte graphique doit également être lancé. Vous pouvez maintenant lancer des applications depuis le terminal texte graphique (figure 22).

Pour les plus téméraires, il est même possible de déporter dans votre Windows (cf. le terminal graphique en haut à droite ainsi que l'application Geany de la figure 21), l'affichage d'une application se trouvant sur une machine Unix distante, à travers une liaison cryptée SSH. Si cela vous intéresse, je vous conseille de poursuivre la lecture de ce mook.

CONCLUSION

Il vous est maintenant possible d'utiliser la ligne de commandes sur les trois principaux systèmes d'exploitation. Pour Mac OS X et Windows, vous serez vraisemblablement confrontés à quelques limitations par rapport à une vraie distribution Linux dans le cadre de l'utilisation des commandes de ce mook. Vous pouvez alors simplement installer un vrai système d'exploitation Linux dans une machine virtuelle par l'intermédiaire de VirtualBox. ▌

RÉFÉRENCES

[1] **http://en.wikipedia.org/wiki/List_of_terminal_emulators**

[2] **http://fr.wikipedia.org/wiki/Darwin_%28informatique%29**

[3] **http://iterm2.com/features.html**

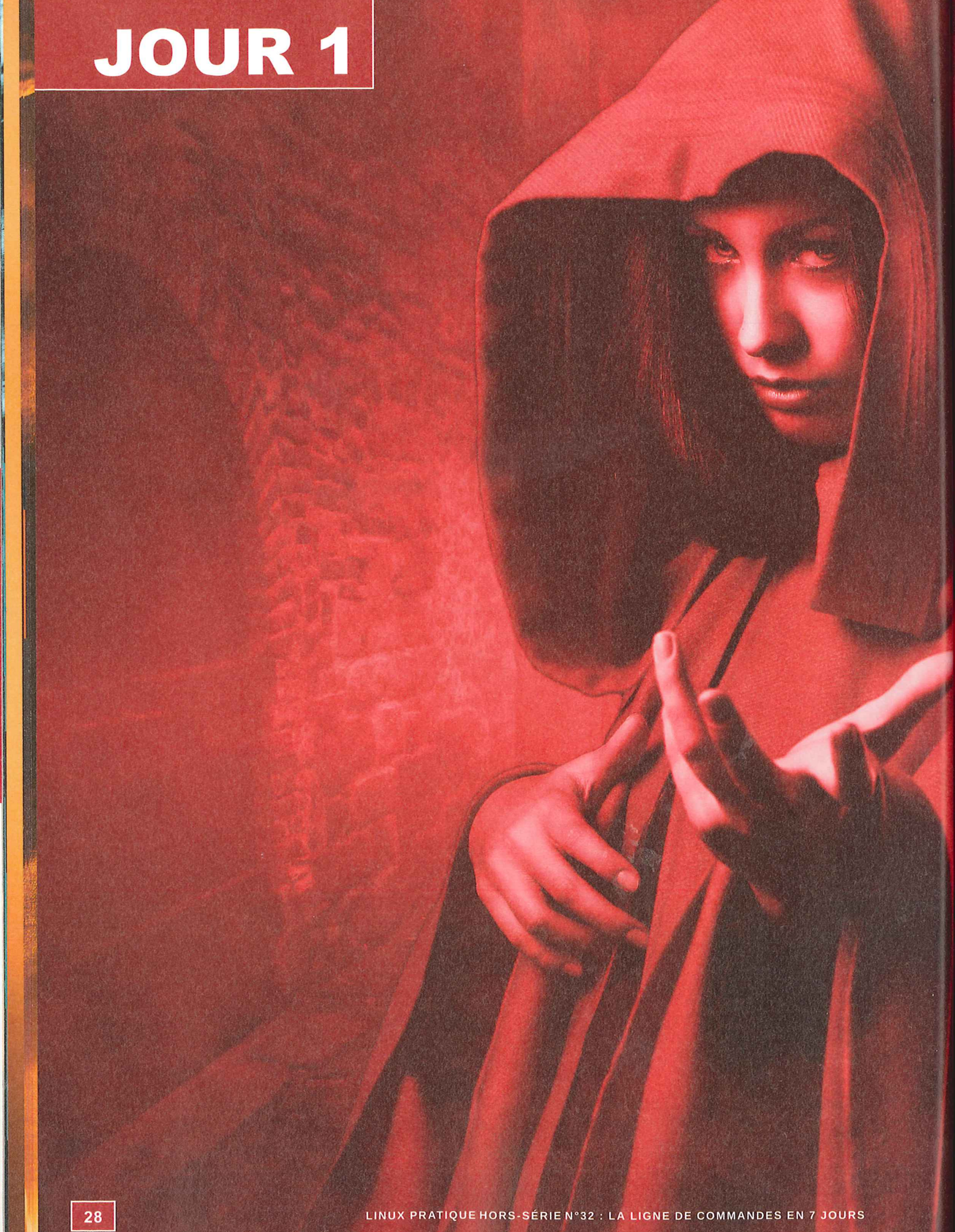
[4] **http://gnometerminator.blogspot.fr/p/introduction.html**

[5] **http://www.secretgeometry.com/apps/cathode**

[6] **http://www.macterm.net/**

[7] **https://technet.microsoft.com/fr-fr/library/cc772390.aspx**

[8] **https://cygwin.com/index.html**



JOUR 1

DÉBUTER AVEC LA LIGNE DE COMMANDES

Yann Morère

On y est. Vous avez installé les outils Cygwin sous Windows ou configuré une machine virtuelle Linux à l'aide de VirtualBox. On peut maintenant entrer dans le vif du sujet : interagir avec le système à l'aide de la ligne de commandes.

1 AU DÉMARRAGE DE VOTRE SHELL

Si votre authentification a été validée par le système, le shell bash est lancé. Au moment de son exécution, celui-ci utilise plusieurs fichiers de configuration en cascade. Il commence par le fichier **/etc/profile** qui va définir certaines variables d'environnement (comme la variable **PS1** responsable de l'affichage de l'invite de commandes/prompt), mais aussi définir des fonctionnalités (l'autocomplétion par exemple) par l'intermédiaire des fichiers contenus dans le répertoire **/etc/profile.d/**. Ensuite, le fichier **.profile** ou **.bash_profile**, s'il existe est exécuté. Ce dernier a pour rôle de définir une variable importante, **PATH**, qui contient la liste des répertoires des exécutables accessibles directement par l'utilisateur. Ce fichier lance aussi le fichier **.bashrc** qui contient la configuration personnalisée de l'interpréteur de commandes de l'utilisateur courant.

Le répertoire **/etc/skel** contient les versions « standards » de ces fichiers. Ils sont copiés dans le répertoire de l'utilisateur lors de la création du compte.

Lorsque vous vous connectez sur un terminal ou lancez un émulateur de terminal, un texte précède le curseur clignotant, il s'agit de l'invite de commande nommée aussi « prompt ». La plupart du temps, il ressemble à ceci :

Terminal
```
yann@geonosis:~$
```

Ce qui correspond à :

Terminal
```
nom@machine~$
```

nom représente l'identifiant de l'utilisateur connecté, **machine** représente le nom de la machine, **~** est un raccourci représentant le répertoire personnel de l'utilisateur **/home/nom**. Finalement, **$** signifie que vous êtes connecté en tant qu'utilisateur.

NOTE

Sur les systèmes GNU/Linux, une convention veut que l'on affiche le caractère # pour l'administrateur et $ pour les utilisateurs standards. Comme vous serez amené à être administrateur de temps en temps, cela vous permettra de différencier les modes et ainsi éviter de faire des erreurs.

On interagit avec le shell à l'aide de commandes entrées au clavier. Il s'agit d'une chaîne de caractères constituée d'une commande interne au shell ou d'un nom de programme exécutable (commande externe au shell), d'options qui vont modifier l'exécution de la commande et de paramètres passés au programme (des fichiers la plupart du temps).

À savoir

Les plus attentifs auront peut être noté la présence des fichiers `.bash_logout` et `.bash_history` qui contiennent respectivement, les commandes à exécuter lors de la déconnexion de l'utilisateur et la liste des dernières commandes bash exécutées.

À retenir

Les fichiers commençant pas un « . » sont par défaut cachés. Il faut utiliser des options spécifiques de listage pour les visualiser. Nous verrons cela un peu plus loin.

Voici un exemple de commande shell complète et complexe :

Terminal
```
$ mencoder loong_fichier.avi -oac copy -ovc copy -ss
00:00:00 -endpos 00:30:00 -o Fichier_30minutes.avi
```

Cette commande permet de récupérer dans un fichier vidéo les 30 premières minutes du fichier global.

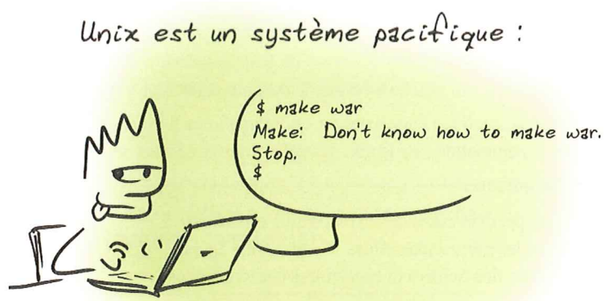


Illustration basée sur le GKND Creator (https://framalab.org/gknd-creator). Œuvre originale de Simon « Gee » Giraudot. Licence Creative Commons By-Sa.

2 SYNTAXE DES COMMANDES SHELL

La syntaxe générale d'une commande Unix est la suivante :

Terminal
```
nom [-options] [argument1 argument2 argument3 etc.]
```

où **nom** est le nom de la commande, **options** représente une ou plusieurs options, **argument1** est le premier argument, **argument2** le second, etc. Dans une commande, chaque mot est séparé des autres par un espace ou une tabulation. Les arguments entre crochets sont facultatifs, certaines commandes s'utilisant seules.

Les options sont le plus souvent composées d'un seul caractère précédé d'un tiret (parfois un +). Il est possible d'accoler plusieurs options (donc, plusieurs caractères). Par exemple :

Terminal
```
ls -a -l -s -i
```

peut être remplacé par

Terminal
```
ls -alsi
```

Mais il est possible d'utiliser des versions plus compréhensibles nommées « forme longue ». Ces options sont précédées de 2 tirets et se composent d'un ou plusieurs mots-clés :

À savoir

Les commandes internes sont des commandes intégrées dans le shell. Ce dernier exécute l'action, mais aucun processus supplémentaire n'est créé. On peut citer les commandes `cd`, `pwd`, `type`, etc.). Les commandes externes sont des fichiers exécutables stockés dans le système de fichiers (ex. : `ls`, `cp`, `mv`, etc.). Ainsi, lors de leur exécution, le shell demande au noyau de charger le fichier et un nouveau processus est créé. Afin de déterminer si une commande est externe ou interne, on utilise la commande `type`.

À retenir

Comme l'espace est le séparateur dans une ligne de commandes shell, il sera judicieux de ne pas les utiliser dans les noms de fichiers que l'on va créer. On remplacera avantageusement l'espace par un « underscore » pour éviter tout problème.

```
Terminal
cp --interactive --remove-destination --archive source destination
```

Si l'option demande un paramètre, il est séparé par un espace. Par exemple :

```
Terminal
gcc -Wall hello.c -o hello
```

permet de compiler le fichier **hello.c** et de nommer l'exécutable produit **hello**.

Certaines options sont assez génériques et sont disponibles sur de nombreuses commandes :

- **-v**, **--verbose** active le mode « verbeux », qui affiche à l'écran le déroulement des opérations ;
- **-h**, **--help** affiche l'aide rapide de la commande ;
- **--version** affiche la version du programme.

Unix est un système d'exploitation qui respecte la casse des caractères, c'est-à-dire qu'il fait la différence entre les majuscules et les minuscules. Ainsi, la commande **Ls** ne fonctionnera pas, alors que **ls** renverra la liste des fichiers et répertoires contenus dans le répertoire courant.

```
Terminal
yann@geonosis:~$ Ls
bash: Ls : commande introuvable
yann@geonosis:~$ ls
Bureau     Filer   Modèles  Public        Téléchargements
Documents  Images  Musique  public_html   Vidéos
morere5@scifa-212-41-1:~$
```

3 UTILISER BASH

3.1 Navigation au clavier

L'utilisation du shell Bash peut être simplifiée par l'utilisation de raccourcis clavier bien utiles :

- [Ctrl] + [P] (ou touche [↑]) : pour rappeler la commande précédente ;
- [Ctrl] + [N] (ou touche [↓]) : pour rappeler la commande suivante ;
- [Ctrl] + [A] (ou touche [Home]) : pour aller en début de ligne ;
- [Ctrl] + [E] (ou touche [Fin]) : pour aller en fin de ligne ;
- [Ctrl] + [D] (ou touche [Suppr]) : pour supprimer un caractère ;
- [Ctrl] + [K] : pour supprimer tout ce qui se trouve à droite du curseur ;
- [Ctrl] + [U] : pour supprimer tout ce qui se trouve à gauche du curseur ;
- [Ctrl] + [L] : pour effacer l'écran ;
- [Ctrl] + [R] : pour lancer une recherche dans l'historique ; on saisit un motif et la plus récente commande qui le contient apparaît (une succession de [Ctrl]+[R] permet de remonter dans l'historique).

La fermeture de l'interpréteur de commandes s'effectue à l'aide de la commande **exit** ou par le raccourci [Ctrl]+[D].

3.2 Complétion automatique

Les interpréteurs de commandes proposent généralement la complétion automatique des commandes, une fonctionnalité vraiment pratique qui permet de diminuer les saisies clavier et d'éviter les fautes de frappe.

Par exemple, sous Bash, il vous suffit de commencer à saisir une commande ou un nom de fichier, puis d'appuyer sur la touche [Tab] pour que le shell complète automatiquement votre saisie, ou affiche la liste des possibilités s'il y en a plusieurs (avec 2 tabulations). Si vous obtenez un « bip » pour tout résultat, c'est qu'aucun fichier ni aucune commande commençant par les caractères saisis n'existent.

La fonctionnalité d'autocomplétion de votre shell est possible grâce à cette instruction contenue dans le fichier **.bashrc** :

```
Fichier
if [ -f /etc/bash_completion ]; then
. /etc/bash_completion
fi
```

Le fichier **/etc/bash_completion**, fourni par le paquet **bash-completion**, comporte l'ensemble des règles d'autocomplétion et c'est grâce à lui que vous pouvez compléter un nom de commande, de fichier, etc.

Le répertoire **/etc/bash_completion.d/** comporte lui aussi des règles d'autocomplétion, plus complexes, propres à certaines commandes (**subversion**, **inkscape**, **bzr**, **dkms**, etc.). Pour obtenir la liste des commandes supportant la complétion automatique, vous pouvez saisir la commande **complete**.

Il est possible de créer ses propres règles d'autocomplétion et de les ajouter au fichier **/etc/bash_completion** ou dans le répertoire **/etc/bash_completion.d/**.

3.3 Historique des commandes

Bash conserve un historique de toutes les commandes saisies. Cet historique est stocké dans le fichier **.bash_history** se trouvant dans votre répertoire personnel.

Pour rappeler la commande précédemment saisie, un simple appui sur la touche [↑] suffit (ou un double point d'exclamation **!!**). La navigation dans l'historique s'effectue avec les touches [↑] et [↑].

La commande **history** permet de lister tout l'historique et les commandes sont identifiées par un numéro :

```
Terminal
yann@geonosis:~$ history
[...]
132968  vim /etc/apache2/sites-enabled/000-default.conf
132969  service apache2 restart
[...]
```

À savoir

Le bip peut également signifier que vous tentez d'exécuter une commande qui nécessite les droits d'administrateur du système, alors que vous êtes connecté en tant qu'utilisateur normal.

JOUR 1

Terminal

```
cp --interactive --remove-destination --archive source destination
```

Si l'option demande un paramètre, il est séparé par un espace. Par exemple :

Terminal

```
gcc -Wall hello.c -o hello
```

permet de compiler le fichier **hello.c** et de nommer l'exécutable produit **hello**.

Certaines options sont assez génériques et sont disponibles sur de nombreuses commandes :

- **-v**, **--verbose** active le mode « verbeux », qui affiche à l'écran le déroulement des opérations ;
- **-h**, **--help** affiche l'aide rapide de la commande ;
- **--version** affiche la version du programme.

Unix est un système d'exploitation qui respecte la casse des caractères, c'est-à-dire qu'il fait la différence entre les majuscules et les minuscules. Ainsi, la commande **Ls** ne fonctionnera pas, alors que **ls** renverra la liste des fichiers et répertoires contenus dans le répertoire courant.

Terminal

```
yann@geonosis:~$ Ls
bash: Ls : commande introuvable
yann@geonosis:~$ ls
Bureau     Filer   Modèles  Public        Téléchargements
Documents  Images  Musique  public_html   Vidéos
morere5@scifa-212-41-1:~$
```

3 UTILISER BASH

3.1 Navigation au clavier

L'utilisation du shell Bash peut être simplifiée par l'utilisation de raccourcis clavier bien utiles :

- [Ctrl] + [P] (ou touche [↑]) : pour rappeler la commande précédente ;
- [Ctrl] + [N] (ou touche [↓]) : pour rappeler la commande suivante ;
- [Ctrl] + [A] (ou touche [Home]) : pour aller en début de ligne ;
- [Ctrl] + [E] (ou touche [Fin]) : pour aller en fin de ligne ;
- [Ctrl] + [D] (ou touche [Suppr]) : pour supprimer un caractère ;
- [Ctrl] + [K] : pour supprimer tout ce qui se trouve à droite du curseur ;
- [Ctrl] + [U] : pour supprimer tout ce qui se trouve à gauche du curseur ;
- [Ctrl] + [L] : pour effacer l'écran ;
- [Ctrl] + [R] : pour lancer une recherche dans l'historique ; on saisit un motif et la plus récente commande qui le contient apparaît (une succession de [Ctrl]+[R] permet de remonter dans l'historique).

La fermeture de l'interpréteur de commandes s'effectue à l'aide de la commande **exit** ou par le raccourci [Ctrl]+[D].

3.2 Complétion automatique

Les interpréteurs de commandes proposent généralement la complétion automatique des commandes, une fonctionnalité vraiment pratique qui permet de diminuer les saisies clavier et d'éviter les fautes de frappe.

Par exemple, sous Bash, il vous suffit de commencer à saisir une commande ou un nom de fichier, puis d'appuyer sur la touche [Tab] pour que le shell complète automatiquement votre saisie, ou affiche la liste des possibilités s'il y en a plusieurs (avec 2 tabulations). Si vous obtenez un « bip » pour tout résultat, c'est qu'aucun fichier ni aucune commande commençant par les caractères saisis n'existent.

La fonctionnalité d'autocomplétion de votre shell est possible grâce à cette instruction contenue dans le fichier **.bashrc** :

Fichier

```
if [ -f /etc/bash_completion ]; then
. /etc/bash_completion
fi
```

Le fichier **/etc/bash_completion**, fourni par le paquet **bash-completion**, comporte l'ensemble des règles d'autocomplétion et c'est grâce à lui que vous pouvez compléter un nom de commande, de fichier, etc.

Le répertoire **/etc/bash_completion.d/** comporte lui aussi des règles d'autocomplétion, plus complexes, propres à certaines commandes (**subversion**, **inkscape**, **bzr**, **dkms**, etc.). Pour obtenir la liste des commandes supportant la complétion automatique, vous pouvez saisir la commande **complete**.

Il est possible de créer ses propres règles d'autocomplétion et de les ajouter au fichier **/etc/bash_completion** ou dans le répertoire **/etc/bash_completion.d/**.

3.3 Historique des commandes

Bash conserve un historique de toutes les commandes saisies. Cet historique est stocké dans le fichier **.bash_history** se trouvant dans votre répertoire personnel.

Pour rappeler la commande précédemment saisie, un simple appui sur la touche [↑] suffit (ou un double point d'exclamation **!!**). La navigation dans l'historique s'effectue avec les touches [↑] et [↑].

La commande **history** permet de lister tout l'historique et les commandes sont identifiées par un numéro :

Terminal

```
yann@geonosis:~$ history
[...]
132968  vim /etc/apache2/sites-enabled/000-default.conf
132969  service apache2 restart
[...]
```

À savoir

Le bip peut également signifier que vous tentez d'exécuter une commande qui nécessite les droits d'administrateur du système, alors que vous êtes connecté en tant qu'utilisateur normal.

Pour rappeler une commande rapidement, il suffit alors de saisir :

Terminal
```
nom@machine :~$ !n
```

où **n** est le numéro de la commande en question. Pour rappeler la dernière commande commençant par vi, on saisira :

Terminal
```
yann@geonosis:~$ !vi
```

Par défaut, 500 entrées sont conservées dans l'historique, mais il est possible de modifier cette valeur en éditant le fichier **~/.bashrc**, dans lequel vous trouverez d'ailleurs plusieurs autres variables associées à l'historique :

Fichier
```
export HISTCONTROL=ignoredups
```

signifie qu'une commande identique à la dernière commande ne sera pas conservée dans l'historique.

Fichier
```
export HISTCONTROL=ignorespace
```

signifie que les commandes qui commencent par le caractère espace ne seront pas enregistrées.

Fichier
```
export HISTCONTROL=ignoreboth
```

est un raccourci pour les options **ignoredups** et **ignorespace**.

Fichier
```
export HISTTIMEFORMAT="%D %H:%M "
```

ajoute la date et l'heure devant chaque commande de l'historique.

Fichier
```
export HISTSIZE=1000
```

définit le nombre de commandes à retenir pour la commande **history**.

Fichier
```
export HISTFILESIZE=2000
```

permet de définir le nombre maximum d'entrées dans l'historique (500 par défaut).

Fichier
```
export HISTIGNORE="ls:cd:ll:la:exit"
```

permet de définir une liste de commandes qui seront automatiquement exclues de l'historique. On notera également dans le fichier **.bashrc** la présence de l'option :

Fichier
```
shopt -s histappend
```

qui permet de grouper les historiques au travers des sessions (les nouvelles lignes sont ajoutées au fichier, le contenu n'est donc pas écrasé à chaque session).

4 LA COMMANDE MAN

Le nombre de commandes et leurs options sont tellement grands, qu'il n'est pas possible de les retenir dans leur ensemble. Il devient alors indispensable de pouvoir retrouver rapidement la syntaxe d'une commande ainsi que les options correctes qui vont vous permettre de réaliser l'opération souhaitée. Cela passe bien sûr par le RTFM (*Read The F*****g Manual*). En ligne de commandes, le manuel est accessible par la commande **man**. C'est LA commande à savoir utiliser convenablement pour progresser dans l'utilisation de l'interpréteur de commandes. Il s'agit du programme de visualisation des pages de manuel.

La syntaxe d'utilisation de la commande man, dans sa version simplifiée, est la suivante :

Terminal
```
man [-s<section>] <nom_de_commande>
```

La plupart du temps, il est possible d'omettre l'option **-s**.

L'argument obligatoire **<nom_de_commande>** est le nom d'un programme, d'un utilitaire ou d'une fonction. Un numéro de section peut être précisé (option **-s**). **man** limite alors la recherche à cette section. Les sections sont au nombre de 9 et possèdent chacune leurs spécificités :

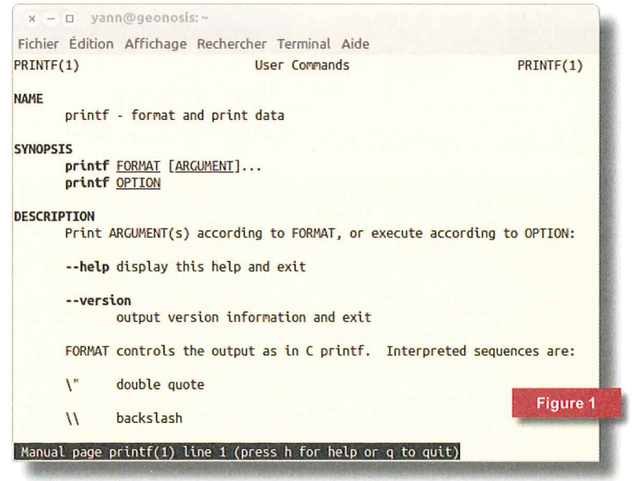


Figure 1

- 1 : Programmes exécutables ou commandes de l'interpréteur de commandes (shell) ;
- 2 : Appels système (fonctions fournies par le noyau) ;
- 3 : Appels de bibliothèque (fonctions fournies par les bibliothèques des programmes) ;
- 4 : Fichiers spéciaux (situés généralement dans **/dev**) ;
- 5 : Formats des fichiers et conventions. Par exemple **/etc/passwd** ;
- 6 : Jeux ;
- 7 : Divers ;
- 8 : Commandes de gestion du système (généralement réservées au super-utilisateur/administrateur/root) ;
- 9 : Sous-programmes du noyau.

La plupart du temps, nous utiliserons la section 1 pour les commandes shell. Mais la commande **man** fait bien mieux. Il est possible d'accéder directement à la syntaxe de fonctions en programmation C. Par exemple, dans la figure 2 (page suivante), en changeant de section, il est possible de consulter la page de manuel de la fonction shell **printf**, mais aussi la description de cette même fonction en langage C.

Pour se déplacer dans une page du manuel, on utilise les touches suivantes :

- [Enter] ou [Return] pour avancer d'une ligne ;
- [Space] ou [PageDown] pour avancer d'une page ;
- [b] ou [PageUp] pour reculer d'une page ;
- / suivi d'un mot pour rechercher ce mot ; pour rechercher l'occurrence suivante du même mot, tapez [n] pour « next » ;
- [q] pour quitter man, revenir au prompt et continuer de travailler.

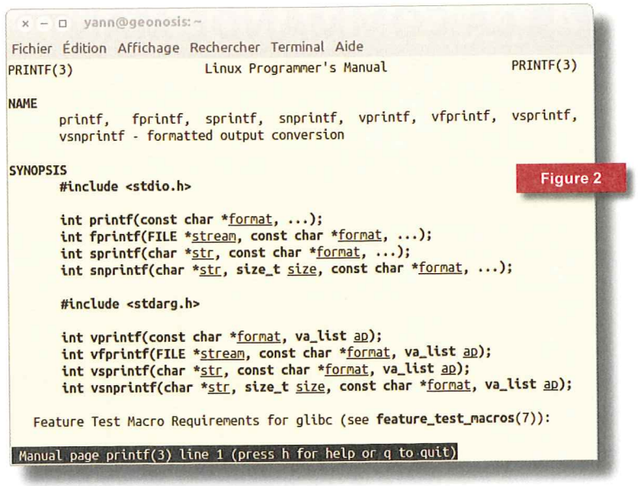


Figure 2

La structure d'une page de man est la suivante :

Terminal

```
COMMAND(1)        Manuel de l'utilisateur Linux        COMMAND(1)
NAME
     commande - résumé de l'action de la commande
SYNOPSYS
    <syntaxe complète de la commande>
DESCRIPTION
    Des explications concernant l'exécution de la commande
OPTIONS
    Liste des options disponibles et ce qu'elles font
FILES
    Les fichiers utilisés par la commande
SEE ALSO
    commande_connexe(1), commande_proche(5), commande_liée(3) etc.
BUGS
     les bugs existants dans la commande
AUTHOR
    Le nom de l'auteur
```

L'item « SEE ALSO » peut parfois être intéressant pour trouver des commandes connexes. Par exemple, en tapant :

Terminal

```
$ man man
```

On apprend qu'il existe les commandes **whatis** et **apropos** :

- **whatis** affiche une ligne de description des pages de manuel de la commande passée en paramètre ;

Terminal

```
$ whatis gcc
gcc (1)              - GNU project C and C++ compiler
```

- **apropos** cherche le nom et la description des pages de manuel du mot-clé passé en paramètre.

Terminal

```
$ apropos gcc
c89-gcc (1)          - ANSI (1989) C compiler
c99-gcc (1)          - ANSI (1999) C compiler
gcc (1)              - GNU project C and C++ compiler
gcc-4.8 (1)          - GNU project C and C++ compiler
x86_64-linux-gnu-gcc-ar-4.8 (1) - a wrapper around ar adding the -
plugin option
gcc-ar-4.8 (1)       - a wrapper around ar adding the - plugin option
x86_64-linux-gnu-gcc-nm-4.8 (1) - a wrapper around nm adding the -
plugin option
gcc-nm-4.8 (1)       - a wrapper around nm adding the - plugin option
x86_64-linux-gnu-gcc-ranlib-4.8 (1) - a wrapper around ranlib adding the
- plugin option
gcc-ranlib-4.8 (1)   - a wrapper around ranlib adding the - plugin option
x86_64-linux-gnu-gcc (1) - GNU project C and C++ compiler
x86_64-linux-gnu-gcc-4.8 (1) - GNU project C and C++ compiler
```

Certaines options de la commande **man** sont très utiles. Par exemple, l'option **-f** permet d'obtenir la description succincte d'une commande, c'est l'équivalent de la commande **whatis** :

Terminal

```
yann@geonosis:~$ man -k whatis
whatis (1)           - Afficher une ligne de description des
pages de manuel
yann@geonosis:~$ whatis whatis
whatis (1)           - Afficher une ligne de description des
pages de manuel
```

De même, l'option **-k** permet d'obtenir la liste des commandes contenant le mot-clé passé en paramètre. C'est l'équivalent de la commande **apropos** :

Terminal

```
yann@geonosis:~$ man -k apropos
apropos (1)          - Chercher le nom et la description des
pages de manuel
yann@geonosis:~$ apropos apropos
apropos (1)          - Chercher le nom et la description des
pages de manuel
```

NOTE

Certains utilitaires provenant du projet GNU possèdent une documentation encore plus fournie appelée les pages info. Malheureusement, l'organisation en arbre de cette documentation la rend plus difficile à utiliser.

À savoir

Au sujet de la commande apropos : le mot clé est considéré en tant qu'expression rationnelle ; dès qu'une commande possède la chaîne de caractères « gcc », la commande apropos en affiche les informations.

5 LES VARIABLES D'ENVIRONNEMENT

Une fois connecté, l'interpréteur de commandes, par l'intermédiaire de ses fichiers de configuration, crée et initialise des variables d'environnement qui permettent de définir le contexte d'exécution des commandes shell que vous utiliserez.

La commande **set** vous permet de lister et manipuler l'ensemble des variables et fonctions définies dans votre shell :

Terminal

```
yann@geonosis:~$ set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:
force_fignore:histappend:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
```

Cet environnement d'exécution par défaut peut être modifié, notamment par la commande **env**, qui va permettre de lister, définir ou supprimer des variables. Son utilisation dans sa forme la plus simple renvoie la liste des variables. Il est aussi possible d'utiliser la commande **printenv** :

Terminal

```
yann@geonosis:~$ env
TERM=xterm
SHELL=/bin/bash
SSH_CLIENT=172.24.129.175 52042 22
SSH_TTY=/dev/pts/23
USER=yann
PWD=/home/yann
LANG=fr_FR.UTF-8
HOME=/home/yann
LOGNAME=yann
SSH_CONNECTION=172.24.129.175 52042 193.50.119.158 22
PROMPT_COMMAND=history -a; history -r
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=localhost:12.0
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env
```

Par défaut, les variables définies au niveau du shell ne sont pas transmises aux commandes lancées à partir de celui-ci : le shell doit d'abord les exporter. Toutefois, on remarque que les variables listées par la commande **env** ou **printenv** sont déjà exportées.

La commande **locale** permet quant à elle de visualiser les variables concernant la langue et l'encodage utilisés pendant la session :

Terminal

```
yann@geonosis:~$ locale
LANG=fr_FR.UTF-8
LANGUAGE=
LC_CTYPE="fr_FR.UTF-8"
LC_NUMERIC="fr_FR.UTF-8"
LC_TIME="fr_FR.UTF-8"
[...]
```

Certaines de ces variables sont importantes à connaître :

- **HOME**, qui désigne le chemin absolu vers le répertoire personnel de l'utilisateur ;
- **LANG**, qui est utilisée par les différents programmes pour déterminer la langue des messages à afficher dans l'application. Les différentes traductions se trouvent dans les fichiers **.mo** des sous-répertoires de **/usr/share/locale** ;
- **LOGNAME**, qui désigne le nom de l'utilisateur ;

- **OLDPWD** qui désigne le chemin vers le répertoire courant précédent (on y accède via la commande **cd**) ;
- **PATH** est une variable très importante, car elle contient tous les chemins d'accès aux commandes du système ; ainsi, les commandes dont le chemin d'accès est spécifié dans la variable **PATH** peuvent être lancées depuis n'importe quel endroit de l'arborescence de fichiers, sans mentionner leur chemin absolu (la notion de chemin relatif et chemin absolu sera étudiée en jour 2).
- **PS1** désigne le prompt principal (par défaut **nom@machine:~$**) ;
- **PS2** désigne le prompt secondaire, qui apparaît au début de la ligne lorsqu'une commande n'est pas terminée et que l'utilisateur doit continuer sur la ligne suivante (par défaut **>**) ;
- **PWD** désigne le chemin absolu vers le répertoire courant ;
- **TERM** désigne le type de terminal utilisé ; elle doit se trouver dans l'environnement, car elle est utilisée par de nombreuses commandes faisant appel aux caractéristiques d'affichage (**vi**, **more**, **clear**, etc.).

Pour afficher le contenu d'une variable (sa valeur), on utilise le caractère **$**. Par exemple, pour visualiser le contenu de la variable **PATH**, on peut utiliser la commande **echo** qui sert à afficher du texte :

Terminal

```
yann@geonosis:~$ echo $PATH
/home/yann/bin:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/
local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/home/yann/
android-sdk-linux/tools:/usr/local/cuda/bin:/home/yann/.local/bin
```

Si la commande ne retourne rien, c'est que la variable n'est pas définie. La valeur d'une variable peut être initialisée à tout moment par l'utilisateur : le caractère **=** permet d'affecter une valeur à une variable (et de créer celle-ci par la même occasion si elle n'existe pas). Par défaut, les variables créées ne sont pas exportées dans l'environnement d'exécution. Il faut les exporter explicitement à l'aide de la commande **export** :

Terminal

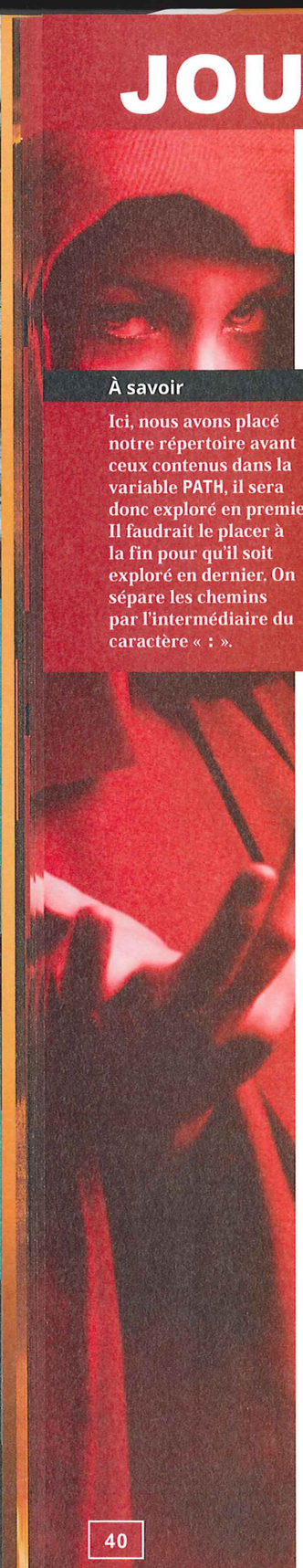
```
yann@geonosis:~$ mavariable=savaleur
yann@geonosis:~$ env | grep mavariable
yann@geonosis:~$ export mavariable
yann@geonosis:~$ env | grep mavariable
mavariable=savaleur
yann@geonosis:~$ echo $mavariable
savaleur
```

Ici, la commande **env | grep ...** permet de rechercher dans la sortie de la commande **env**, la présence de la chaîne de caractères « mavariable » permettant ainsi de vérifier la présence ou non de la variable dans l'environnement d'exécution. Nous verrons très bientôt cette notion de chaînage de commande.

Maintenant, nous allons modifier la variable **PATH** explicitée plus haut. Prenons un exemple concret : vous avez installé sur votre PC les outils de compilation pour le Raspberry Pi dans un répertoire spécifique de votre répertoire personnel. Cependant, par défaut, le compilateur utilisé sera celui prévu pour l'architecture de votre système d'exploitation (x86, 32 ou 64bits).

Terminal

```
yann@geonosis:~$ which gcc
/usr/bin/gcc
```

Pour pouvoir utiliser par défaut les outils de compilation de votre répertoire, il faut modifier la variable **PATH** afin que votre répertoire apparaisse avant le répertoire **/usr/bin** qui contient le compilateur par défaut :

Terminal

```
[yann@archery bin]$ export PATH=$HOME/raspberry/tools/arm-
bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin:$PATH
[yann@archery bin]$ which gcc
/home/yann/raspberry/tools/arm-bcm2708/gcc-linaro-arm-linux-
gnueabihf-raspbian/bin/gcc
[yann@archery bin]$
```

À savoir

Ici, nous avons placé notre répertoire avant ceux contenus dans la variable PATH, il sera donc exploré en premier. Il faudrait le placer à la fin pour qu'il soit exploré en dernier. On sépare les chemins par l'intermédiaire du caractère « : ».

Si vous procédez ainsi, cette modification ne sera effective que pour la session courante de votre shell et la valeur de **PATH** sera réinitialisée dès la prochaine session shell. Pour modifier la variable **PATH** de façon permanente, on ajoute la ligne dans le fichier **/home/user/.bashrc**.

Il existe aussi la variable **CDPATH** qui permet, à la manière de **PATH**, de définir une liste de répertoires, où la commande **cd** doit rechercher le répertoire dont vous saisissez le nom. En l'adaptant à vos besoins, vous pourrez gagner du temps en évitant de saisir le chemin complet vers les répertoires que vous utilisez couramment.

6 PERSONNALISER VOTRE SHELL

6.1 Le prompt

Par défaut, sur de nombreuses distributions, le prompt est constitué de votre nom d'utilisateur, suivi du nom de votre ordinateur. Suit le caractère tilde (**~**) qui indique que vous vous trouvez dans votre répertoire personnel, puis le caractère **$** qui indique que vous agissez en tant qu'utilisateur « normal » ; en effet, si vous étiez root (l'administrateur du système), ce caractère serait remplacé par un dièse (**#**).

La valeur du prompt est stockée dans une variable d'environnement **PS1**. Dans ce cas par défaut, le contenu de **PS1** est le suivant :

Fichier

```
\u@\h:\w\$
```

ce qui se traduit par :

Terminal

```
yann@geonosis:~$
```

Chaque élément variable et configurable du prompt est introduit par le caractère antislash (\). Les autres caractères, ainsi que les espaces, sont insérés en tant que tels. Voici leurs significations :

- **\e** désigne un caractère espace (équivalent à \033) ;
- **\u** désigne le nom de l'utilisateur courant ;
- **\h** désigne le nom d'hôte de la machine ;
- **\w** désigne le répertoire courant (par son chemin absolu) ;

- **\a** désigne un caractère d'appel (bip sonore) ;
- **\[** permet d'entamer une séquence de caractères non imprimables ;
- **\]** termine une séquence de caractères non-imprimables ;
- **\d** désigne la date ;
- **\n** désigne le saut de ligne ;
- **\t** pour l'heure au format HH:MM:SS (sur 24H) ;
- **\W** pour le répertoire courant.

Il est également possible d'afficher en couleur chaque élément composant le prompt. On fait précéder l'élément d'un code couleur parmi les suivants :

- **\033[01;30m\]** : Noir ;
- **\033[01;31m\]** : Rouge ;
- **\033[01;32m\]** : Vert ;
- **\033[01;33m\]** : Jaune ;
- **\033[01;34m\]** : Bleu ;
- **\033[01;35m\]** : Violet ;
- **\033[01;36m\]** : Cyan ;
- **\033[01;37m\]** : Gris.

Pour obtenir des nuances plus claires, on pourra remplacer les occurrences de **01** par **00**. Puis, pour rétablir les couleurs par défaut, on utilisera le code **\033[00m\]**. Pour modifier l'apparence du prompt, on modifie la valeur de la variable **PS1**. Par exemple :

Terminal

```
yann@geonosis:~$ export PS1='\[\033[01;34m\]\u\[\033[01;31m\]@\
[\033[01;32m\]\h\[\033[00m\]:\[\033[01;33m\]\w\[\033[00m\]\$'
```

nous donne le résultat de la figure 3 :

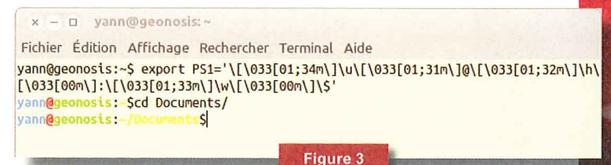


Figure 3

Cette modification n'est active que pendant la session de l'interpréteur de commandes. Si le terminal est fermé puis ouvert à nouveau, le prompt revêt son aspect initial. Pour que la modification persiste, il faut éditer le fichier **~/.bashrc** et modifier la ligne définissant la variable **PS1**.

Il est possible d'obtenir un prompt en couleur plus facilement. Il suffit de dé-commenter la ligne suivante dans le fichier **~/.bashrc**.

Fichier

```
# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
force_color_prompt=yes
```

Il suffit alors de modifier la partie de code suivante dans le même fichier pour adapter à ses besoins les couleurs du prompt :

Fichier

```
if [ "$color_prompt" = yes ]; then
    PS1='$\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='\u@\h:\w\$ '
fi
```

Voici un prompt trouvé à l'adresse [1]. Il est un peu chargé, mais complet. Il affiche des informations dans la console et dans le titre de la fenêtre (figure 4) :

Fichier

```
export PROMPT_COMMAND='export H1="'history 1|sed -e
"s/^[\ 0-9]*//; s/[\d0\d31\d34\d39\d96\d127]*//g; s/\
(.\{1,50\}\).*$/\1/g"'";history -a;echo -e "sgr0\ncnorm\
nrmso"|tput -S'
export PS1='\n\e[1;30m[\j:\!\e[1;30m]\e[0;36m \T \d
\e[1;30m[\e[1;34m\u@\H\e[1;30m:\e[0;30m'tty 2>/dev/null'
\e[0;32m+${SHLVL}\e[1;30m] \e[1;30m\w\e[0;30m\[\033]0;
[ ${H1}... ] \w - \u@\H +$SHLVL @'tty 2>/dev/null' -
[ 'uptime' ]\007\]\n\[\]\$ '
```

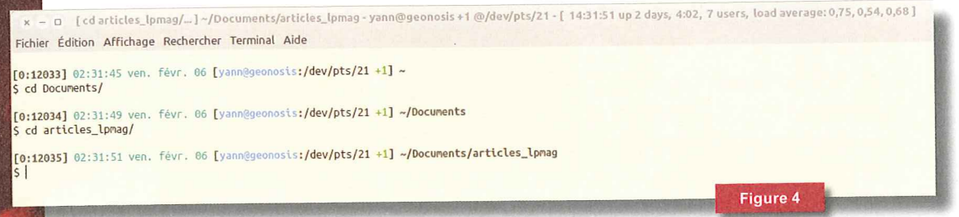


Figure 4

6.2 Les alias

Les alias permettent de créer des versions raccourcies des commandes que vous utilisez fréquemment. Un certain nombre d'alias existent par défaut sur votre système ; ils sont définis dans votre fichier **~/.bashrc** :

Fichier

```
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b
~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] &&
echo terminal || echo error)" "$(history|tail -n1|sed -e
'\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
```

Vous avez également la possibilité de créer un fichier **~/.bash_aliases** qui contiendra tous vos alias supplémentaires. Après toute modification dans l'un ou l'autre de ces fichiers, il est nécessaire de relancer votre émulateur de terminal pour que le nouvel **alias** soit pris en compte.

On peut aussi définir un alias directement avec la ligne de commandes. La syntaxe est la suivante :

À savoir

Le programme `dircolors` (contenu dans le paquet `coreutils`) permet de bénéficier de la coloration syntaxique pour les commandes usuelles `ls`, `dir`, `vdir`, `grep`, `fgrep` et `egrep` que nous détaillerons dans la suite.

Terminal

```
$ alias <alias>='<commande>'
```

Par exemple les alias suivants permettent d'afficher les chemins de la variable **PATH**, l'heure courante ainsi que la date courante :

Terminal

```
yann@geonosis:~$ alias path='echo -e ${PATH//:/\\n}'
yann@geonosis:~$ alias nowtime='date +"%T"'
yann@geonosis:~$ alias nowdate='date +"%d-%m-%Y"'
yann@geonosis:~$ path
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games
yann@geonosis:~$ nowtime
15:23:32
yann@geonosis:~$ nowdate
06-02-2015
```

La commande **unalias** permet de désactiver momentanément un alias :

Terminal

```
yann@geonosis:~$ unalias nowtime
```

Récapitulatif

- Les fichiers commençant pas un « . » sont par défaut cachés. Il faut utiliser des options spécifiques de listage pour les visualiser.
- Sur les systèmes GNU/Linux, une convention veut que l'on affiche le caractère **#** pour l'administrateur et **$** pour les utilisateurs standards.
- Les commandes internes sont des commandes intégrées dans le shell. Les commandes externes sont des fichiers exécutables stockés dans le système de fichiers.
- Unix est un système d'exploitation qui respecte la casse des caractères, c'est-à-dire qu'il fait la différence entre les majuscules et les minuscules.
- En ligne de commandes, le manuel est accessible par la commande **man**.
- Les variables d'environnement permettent de définir le contexte d'exécution des commandes shell que vous utiliserez.
- La ligne de commandes est assistée des outils d'historique, d'auto-complétion et de coloration syntaxique.
- Les alias permettent de créer des versions raccourcies des commandes que vous utilisez fréquemment.

RÉFÉRENCE

[1] **http://www.askapache.com/linux/bash-power-prompt.html**

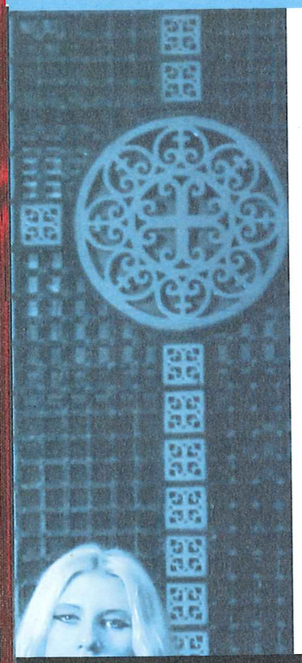
JOUR 2

CONNAÎTRE SON SYSTÈME DE FICHIERS ET GÉRER SES FICHIERS

Yann Morère

Lorsque l'on utilise son ordinateur, la majorité du temps on crée ou modifie de l'information que l'on stocke dans des fichiers dans l'arborescence du système de fichiers. Voyons comment gérer efficacement ce dernier et comment effectuer des opérations simples sur ses fichiers à l'aide de la ligne de commandes.

JOUR 2



1 CONNAÎTRE SON ENVIRONNEMENT DE TRAVAIL

Avant de jouer avec les fichiers et l'arborescence de fichiers, une fois connecté sur notre terminal, il convient de prendre quelques informations sur notre environnement de travail : shell utilisé, distribution utilisée, version du noyau, personnes connectées, etc.

Commençons par afficher les informations du système d'exploitation installé. Pour cela, on utilise le programme `lsb_release` :

Terminal

```
yann@yann-biboo:~$ lsb_release  -idcr
Distributor ID:    Ubuntu
Description: Ubuntu 14.04.1 LTS
Release:     14.04
Codename:    trusty
```

Les options utilisées sont les suivantes :

- `-i`, `--id` : affiche l'identifiant du distributeur, ici Ubuntu ;
- `-d`, `--description` : affiche la description de la distribution ;
- `-r`, `--release` : affiche le numéro de version de la distribution ;
- `-c`, `--codename` : affiche le nom de code de la distribution.

Voyons à présent comment retrouver la version du noyau et l'architecture utilisée. Pour cela, on utilise la commande `uname` qui affiche les informations système :

Terminal

```
yann@yann-biboo:~$ uname
Linux
yann@yann-biboo:~$ uname -a
Linux yann-biboo 3.13.0-45-generic #74-Ubuntu SMP Tue Jan
13 19:36:28 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
yann@SnowLeopard ~$ uname
Darwin
yann@SnowLeopard ~$ uname -a
Darwin SnowLeopard 10.0.0 Darwin Kernel Version 10.0.0:
Fri Jul 31 22:47:34 PDT 2009; root:xnu-1456.1.25~1/
RELEASE_I386 i386
```

La commande utilisée sans argument nous informe du type de noyau. Utilisée avec l'option `-a`, la commande est bien plus verbeuse : nom du noyau, version du noyau, date de compilation et type d'architecture.

Si vous n'êtes pas l'installateur ou l'administrateur de la machine, il est possible que le shell par défaut ne soit pas bash. Pou reconnaître votre interpréteur de commandes, il faut afficher le contenu de la variable `SHELL` à l'aide de la commande `echo` :

Terminal

```
yann@yann-biboo:~$ echo $SHELL
/bin/bash
```

À savoir

Pour mieux connaître son système, on peut aussi afficher le contenu de fichiers spécifiques à chaque distribution : `cat /etc/lsb-release` sur Ubuntu ; `cat /etc/redhat-release` sur Redhat/Fedora/Centos, `cat /etc/SuSE-release` sur openSUSE, `cat /etc/issue` sur Debian, `cat /etc/slackware-version` sur Slackware. Nous verrons bientôt plus en détail la commande `cat`.

Si d'autres shells sont installés sur votre machine, il possible de changer d'interpréteur de commandes en lançant l'exécutable du shell éponyme :

Terminal

```
yann@bilbo:~$ /bin/tcsh
bilbo:~> /bin/csh
bilbo:~% exit
exit
bilbo:~> exit
exit
yann@bilbo:~$
```

On quitte le shell lancé par la commande `exit`. Vous noterez que chaque shell possède une configuration différente du prompt. Cela permet de les reconnaître.

On peut ensuite afficher l'identifiant du terminal sur lequel vous travaillez. La commande `tty` s'utilise sans argument :

Terminal

```
yann@yann-biboo:~$ tty
/dev/pts/8
```

dans le cas d'un émulateur graphique de terminal (pseudo terminal),

Terminal

```
yann@yann-biboo:~$ tty
/dev/tty1
```

dans le cas du premier terminal console.

La commande `uptime` nous permet de savoir depuis combien de temps le système est démarré. L'option `-p` affiche les informations de durée de manière plus conviviale :

Terminal

```
yann@bilbo:~$ uptime
 15:52:07 up 65 days,  3:43,  4 users,  load average: 0,00, 0,03, 0,05
yann@bilbo:~$ uptime -p
up 9 weeks, 2 days, 3 hours, 43 minutes
```

Voyons maintenant qui est connecté. Pour cela on utilise la commande `w`. Elle affiche les utilisateurs connectés, la date de connexion, ce qu'ils sont en train d'exécuter ainsi que le terminal utilisé :

Terminal

```
yann@yann-biboo:~$ w
 16:31:59 up 1 day,  8:47,  6 users,  load average: 0,86, 1,06, 1,06
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
yann     tty1                      15:44   47:51   0.10s  0.09s -bash
yann     :0       :0               sam.08  ?xdm?  11:34m  0.20s mate-session
yann     pts/0    :0.0             sam.08   7.00s  9:51   0.00s w
yann     pts/1    :0.0              15:51   34:55   0.05s  0.01s ssh yann@193.50
yann     pts/8    :0.0             sam.17   40:31   0.20s  0.20s bash
yann     pts/9    :0.0             sam.18   14:07   0.30s  0.30s bash
```

Les options de cette commande permettent de filtrer et réduire l'affichage. Voici à quoi correspondent les différentes colonnes affichées :

- USER : nom d'utilisateur ;
- TTY : type et numéro de terminal (console/tty ou émulateur pts) ;
- FROM : machine de connexion ou adresse IP ;
- LOGIN@ : instant de connexion ;
- IDLE : durée d'inactivité ;
- JCPU : le temps JCPU est le temps utilisé par tous les processus rattachés à ce terminal ;
- PCPU : le temps JCPU est le temps utilisé par le processus courant affiché dans le champ « WHAT » ;
- WHAT : la ligne de commandes du processus courant de `USER`.

La commande `who am i` permet de récupérer les informations du terminal sur lequel vous être en train de taper la commande :

Terminal
```
yann@yann-biboo:~$ who am i
yann     pts/0        2015-02-07 08:03 (:0.0)
```

Cela vous permet de retrouver un terminal précis, dans le cas où vous en avez ouvert un grand nombre.

La commande `users` affiche la liste des utilisateurs connectés sur la machine. Si cela n'a aucun intérêt sur votre machine de bureau où seul votre compte existe, cela peut être utile dans un vrai environnement multi-utilisateurs.

Terminal
```
yann@yann-biboo:~$ users
yann yann yann yann yann yann
```

On voit ici que je suis connecté 6 fois !

2 CHEMIN ABSOLU ET CHEMIN RELATIF

Pour utiliser un fichier dans une commande shell, la plupart du temps son simple nom ne suffit pas (sauf s'il se trouve dans le répertoire courant de travail). Sous Unix, il est nécessaire de faire référence à un fichier par l'intermédiaire d'un « chemin », c'est-à-dire l'enchaînement des répertoires de l'arborescence qui vont conduire à celui dans lequel il est stocké. Le chemin indique précisément dans quel répertoire se trouve le fichier. La syntaxe se présente sous la forme suivante :

Terminal
```
yann@yann-biboo:~$ cat /chemin/vers/le/repertoire/de/mon/fichier.txt
```

Sous Linux, les noms des répertoires et des fichiers au sein d'un « chemin » sont séparés par un slash « / ». Sous Windows, le séparateur est le caractère anti-slash « \ », ce qui pose de nombreux soucis interopérabilité. Le « . » représente le répertoire courant. Les « .. » représentent le répertoire précédent dans l'arborescence. On le nommera aussi le répertoire « parent ». Les deux points permettent en effet de remonter d'un niveau dans l'arborescence du système de fichiers.

On distingue deux types de chemins :

- le chemin **absolu** prend pour référence la racine de l'arborescence, il commence donc toujours par le symbole « / » ;

- un chemin **relatif** dépend du répertoire courant où se trouve l'utilisateur au moment où il fait référence au fichier.

D'après le schéma de l'arborescence de la figure 1, si je (utilisateur yann) souhaite faire référence de manière absolue au fichier `cd1.ogg`, j'écrirai :

Terminal
```
/home/yann/Musique/cd1.ogg
```

Si je me situe (répertoire courant) dans le répertoire `Videos`, le chemin relatif vers ce même fichier s'écrira :

Terminal
```
../Musique/cd1.ogg
```

en utilisant les « `..` » désignant le répertoire précédent dans l'arborescence.

Si je me situe dans le répertoire `Images` et que je saisis la commande suivante :

Terminal
```
yann@yann-biboo: ~/Images$ play cd1.ogg
play FAIL formats: can't open input file 'cd1.ogg': No such file
or directory
```

le système me répond qu'il ne trouve pas le fichier `cd1.ogg`, puisque ce dernier se trouve dans le répertoire `Musique`. En revanche, si j'exécute cette commande depuis le répertoire `Musique`, elle sera tout à fait fonctionnelle.

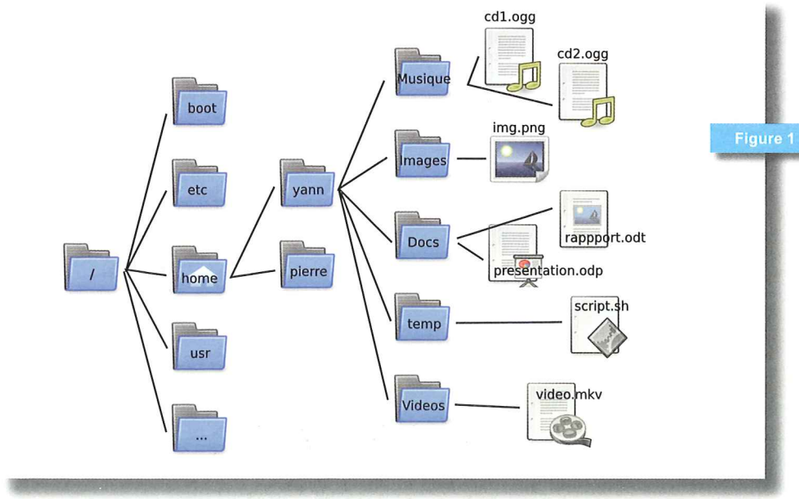
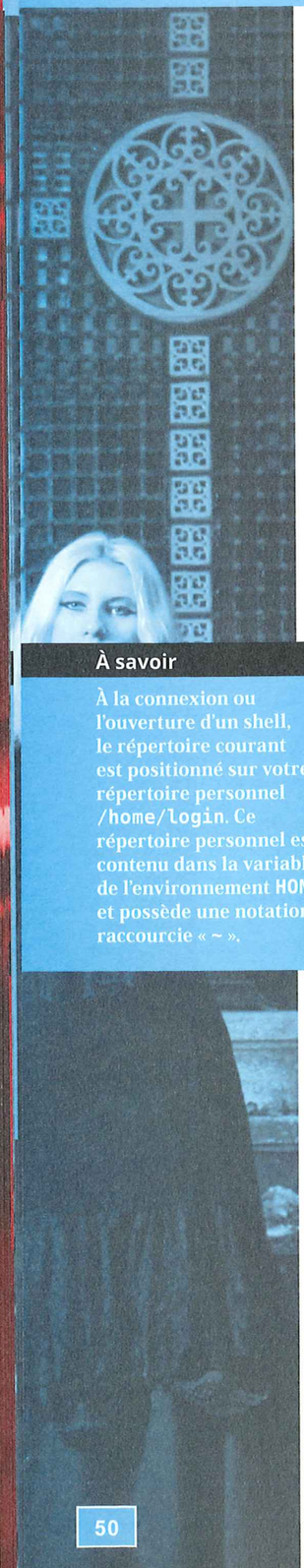


Figure 1

3 NAVIGUER DANS L'ARBORESCENCE

Avec l'exemple précédent, on se rend compte qu'il est indispensable de pouvoir se déplacer dans l'arborescence du système de fichier afin d'éviter de saisir continuellement des chemins absolus ou relatifs vers les fichiers que l'on désire utiliser.

JOUR 2



Avant de se déplacer, il faut savoir ou l'on est, vous savez le petit point rouge sur les cartes vous indiquant « Vous êtes ici », et bien sous Unix ce rôle est dévolu à la commande `pwd`, pour *Print Working Directory*. Elle affiche le nom complet du répertoire ainsi que son chemin absolu :

Terminal
```
[yann@yann-vaio article_mook_shell]$ pwd
/home/yann/Documents/articles_lpmag/article_mook_shell
```

Pour se déplacer d'un répertoire à un autre, on utilise la commande `cd` (pour « *change directory* ») suivie du nom (ou du chemin) du répertoire de destination :

Terminal
```
[yann@yann-vaio Musique]$ cd ../Vidéos/
[yann@yann-vaio Vidéos]$ pwd
/home/yann/Vidéos
```

Les commandes suivantes permettent de revenir directement au répertoire personnel depuis n'importe quel lieu de l'arborescence.

Terminal
```
[yann@yann-vaio Vidéos]$ pwd
/home/yann/Vidéos
[yann@yann-vaio Vidéos]$ cd $HOME
[yann@yann-vaio ~]$ pwd
/home/yann
[yann@yann-vaio ~]$ cd Vidéos/
[yann@yann-vaio Vidéos]$ cd ~
[yann@yann-vaio ~]$ pwd
/home/yann
[yann@yann-vaio ~]$ cd Vidéos/
[yann@yann-vaio Vidéos]$ cd
[yann@yann-vaio ~]$ pwd
/home/yann
```

Pour revenir au répertoire courant précédent, on utilise le tiret (moins). Cette action ne remonte pas forcement d'un cran dans l'arborescence, mais annule la dernière commande de déplacement. Voyons cela :

Terminal
```
[yann@yann-vaio Mixxx]$ pwd
/home/yann/Musique/Mixxx
[yann@yann-vaio Mixxx]$ cd -
/home/yann/Musique
[yann@yann-vaio Musique]$
```

Afin de revenir dans le répertoire précédent dans l'arborescence, il faut utiliser les « `..` » :

Terminal
```
[yann@yann-vaio brasil]$ pwd
/home/yann/Musique/brasil
[yann@yann-vaio brasil]$ cd ../../
[yann@yann-vaio ~]$ pwd
/home/yann
```

Dans l'exemple précédent, on remonte 2 fois dans le système de fichiers.

À savoir

À la connexion ou l'ouverture d'un shell, le répertoire courant est positionné sur votre répertoire personnel `/home/login`. Ce répertoire personnel est contenu dans la variable de l'environnement `HOME` et possède une notation raccourcie « ~ ».

4 CRÉER ET SUPPRIMER DES RÉPERTOIRES

Maintenant que nous savons nous déplacer dans les répertoires, voyons comment les créer et les supprimer. Cela vous permettre de concevoir votre propre arborescence de fichiers et d'y ranger vos documents.

La commande de création d'un répertoire est `mkdir` pour « *make directory* ». Ainsi, pour créer le répertoire test dans le répertoire courant, on saisit :

Terminal
```
yann@yann-vaio ~]$ mkdir test/
```

L'option `-p` permet de créer un répertoire et ses sous-répertoires simultanément :

Terminal
```
[yann@yann-vaio ~]$ mkdir -p test/soustest/sousoustest
```

La suppression d'un répertoire s'effectue avec la commande `rmdir` pour « *remove directory* ». Attention, si le répertoire passé en paramètre n'est pas vide, le système refusera de réaliser la suppression.

Terminal
```
[yann@yann-vaio ~]$ rmdir test/
rmdir: impossible de de supprimer " test/ ": Le dossier
n'est pas vide
```

L'option `-p` permet de supprimer le répertoire ciblé, ainsi que son (ou ses) répertoire(s) parent(s). Ainsi, la commande :

Terminal
```
[yann@yann-vaio ~]$ rmdir -p test/soustest/
```

va effacer les répertoires `soustest` et `test`.

5 LES CARACTÈRES ET EXPRESSIONS GÉNÉRIQUES

Les caractères et expressions génériques sont issus d'un mécanisme plus général appelé expressions rationnelles (« *regular expressions* » en anglais) ou expressions régulières que l'on verra par la suite. Ces caractères et expressions sont utilisés pour spécifier un modèle de noms d'entrées. Ce modèle est ensuite interprété par le shell pour créer une liste triée de noms d'entrées. Par défaut, le shell traite uniquement les entrées non cachées du répertoire courant.

À retenir

Pour les nouveaux utilisateurs, il peut être déroutant de ne pas avoir de retour d'information lors de l'exécution d'une commande. Ceci est le fonctionnement normal, si la commande n'affiche rien, c'est qu'il n'y avait rien à afficher et que tout s'est bien passé.

À savoir

Les répertoires étant un type de fichier particulier, il est possible d'effacer un répertoire à l'aide de la commande d'effacement de ficher « `rm` » que nous verrons un peu plus loin. Cette dernière avec l'option `-f` (`force`) permet de forcer l'effacement d'un répertoire non vide. Combinée avec l'option `-r` (`recursive`) il est possible d'effacer récursivement une arborescence complète. Dans les mains de l'administrateur, c'est une commande redoutable.

À savoir

Du fait de la conception du système de fichiers, l'effacement d'un objet en mode console est définitif. Il n'est pas possible de le récupérer par l'intermédiaire d'une corbeille comme en mode graphique. Soyez donc très attentif à vos commandes d'effacement.

À savoir

Pour que bash interprète les caractères génériques, il est nécessaire que l'option `noglob` de la commande interne `set` soit à l'état `off`. C'est le cas sur la plupart des distributions Linux. Pour connaître l'état des différentes options de la commande interne `set`, on utilise la commande `set -o`.

Considérons l'arborescence suivante :

Terminal
```
.
ÐÐÐ fic1.txt
ÐÐÐ fic2.txt
ÐÐÐ fic3.txt
ÐÐÐ rep1
Ð   ÐÐÐ sousrep1
ÐÐÐ rep2
    ÐÐÐ sousrep2
```

Les caractères génériques du shell sont les suivants : `*`, `?`, `[]`.

Le caractère générique `*` désigne n'importe quelle chaîne de caractères, même la chaîne vide. La commande suivante liste toutes les entrées du répertoire ayant l'extension `.txt` :

Terminal
```
yann@geonosis:~$ echo *.txt
fic1.txt  fic2.txt  fic3.txt  fic_copie.txt
```

La commande suivante liste toutes les entrées du répertoire ayant dans le nom la chaîne de caractères « `cop` » :

Terminal
```
yann@geonosis:~$ echo *cop*
fic_copie.txt
```

Le caractère générique `?` désigne un et un seul caractère. La commande suivante liste toutes les entrées du répertoire dont le nom commence par `fic` contient 4 caractères et possédant l'extension `.txt` :

Terminal
```
yann@geonosis:~$ echo fic?.txt
fic1.txt fic2.txt fic3.txt
```

Plusieurs caractères génériques différents peuvent être présents dans un même modèle.

Les caractères génériques `[]` désignent un seul caractère parmi un groupe de caractères. Ce groupe est précisé entre les caractères `[]`. Il peut prendre deux formes :

- Forme ensemble : tous les caractères souhaités sont explicitement mentionnés entre `[]`. Ici, le modèle `[12]` désigne tous les noms d'entrées contenant à cette position le caractère 1 ou 2 :

Terminal
```
yann@geonosis:~$ echo fic[12].txt
fic1.txt fic2.txt
```

- Forme intervalle : seules les bornes de l'intervalle sont précisées et séparées par un `-`. Ici, le modèle `[1-3]` désigne tous les noms d'entrées contenant à cette position le caractère 1,2 ou 3 :

Terminal
```
yann@geonosis:~$ echo fic[1-3].txt
fic1.txt fic2.txt fic3.txt
```

À savoir

À noter, l'interprétation du modèle `[a-z]` est conditionnée par la valeur des paramètres régionaux. Cette syntaxe est donc non portable et il est déconseillé de l'employer, car elle peut avoir des effets néfastes.

6 LISTER, CRÉER, COPIER, RENOMMER ET SUPPRIMER VOS FICHIERS

Nous allons maintenant travailler avec les fichiers standards. Dans un premier temps, nous allons voir comment afficher la liste de fichiers et répertoires du répertoire courant. Pour cela, on utilise la commande `ls` :

Terminal
```
yann@geonosis:~/Documents/articles_lpmag/article_mook_shell$ ls
debutersurLMW.odt   introduction.odt    jour2.odt          preface.odt
formation_unix.pdf  introductionv2.odt  lphs20.pdf         prefacev2.odt
images              jour1.odt           lphs27-small.pdf
```

La commande `ls` possède de nombreuses options, les plus courantes sont les suivantes :

- `-a` permet d'afficher les fichiers/répertoires cachés (dont le nom commence par un `.`) ;
- `-A` permet d'afficher les fichiers/répertoires cachés (dont le nom commence par un `.`) en omettant le répertoire « `.` » (répertoire courant) et « `..` » (répertoire parent) ;
- `-l` affiche les informations détaillées sur les fichiers/répertoires : droits, propriétaire du fichier, groupe auquel il appartient, taille en octets, heure de la dernière modification ;
- `-t` trie les fichiers par date de modification, en affichant la plus récente modification en premier ;
- `-rt` trie les fichiers par date de modification, en affichant la plus ancienne modification en premier ;
- `-C` liste les fichiers sous forme de colonnes ;
- `-F` ajoute un caractère spécifique pour distinguer les différents types de fichiers : caractère slash (`/`) après les répertoires, caractère astérisque (`*`) après les fichiers exécutables, signe égal (`=`) si c'est un socket, caractère pipe (`|`) si c'est un fichier de type FIFO, et caractère arobase (`@`) si c'est un lien symbolique ;
- `-h` (`- -human-readable`) affiche la taille des fichiers dans un format facilement compréhensible ;
- `-R` affiche récursivement le contenu des répertoires et sous-répertoires.
- `-S` affiche les fichiers du plus petit au plus gros ;
- `-G` combinée `-l` affiche les informations du contenu en omettant le groupe auquel appartient le fichier ;
- `-m` affiche sur une seule ligne.

Voici un exemple :

Terminal
```
yann@geonosis:~/Documents/articles_lpmag/article_mook_shell$ ls -lhG
total 39M
-rw-rw-r-- 1 yann 3,8M févr.  5 15:56 debutersurLMW.odt
-rw-r--r-- 1 yann 2,8M févr.  8 14:45 formation_unix.pdf
drwxrwxr-x 5 yann 4,0K févr.  9 17:50 images
-rw-rw-r-- 1 yann 1,7M janv. 29 16:02 introduction.odt
-rw-rw-r-- 1 yann 1,4M févr.  5 17:29 introductionv2.odt
```

NOTE

Les explications des différents champs de l'affichage long de la commande `ls` seront abordés un peu plus loin.

À savoir

Lors du listage de contenu, le . désigne le répertoire courant et .. le répertoire parent.

La commande `cat` va nous permettre de créer un fichier texte avec du contenu. Cette commande sert normalement à concaténer des fichiers entre eux et les afficher sur la sortie standard. Nous allons ici utiliser la redirection de la sortie pour créer notre fichier texte.

NOTE

Les explications concernant les redirections seront abordées plus loin. Il s'agit ici de créer rapidement un fichier texte en utilisant une « moulinette » que l'on comprendra par la suite ;)

Terminal

```
yann@geonosis:~$ cat > fichier.txt
Mon premier fichier texte
et voila
je tape CTRL-D pour sortie de la commande
et créer le fichier
yann@geonosis:~$ ls -al fichier.txt
-rw-rw-r-- 1 yann yann 99 févr. 13 10:55 fichier.txt
yann@geonosis:~$
```

On peut aussi créer un fichier avec un éditeur de texte du type vim (VI), emacs ou encore nano dont l'utilisation est plus proche ce que l'on utilise en mode graphique. Je vous conseille d'ailleurs d'utiliser ce dernier si vous ne connaissez pas du tout les deux autres.

Voyons maintenant comment copier des fichiers. On utilise pour cela la commande `cp`. La syntaxe est la suivante :

Terminal

```
$ cp <fichier(s)/répertoires_à_copier> <répertoire_de_
destination>
```

s'il s'agit de de copier des fichiers dans un répertoire, et

Terminal

```
$ cp <fichier/repertoire_source_à_copier> <fichier_
destination_de_copie>
```

s'il s'agit de copier un fichier dans le répertoire courant. Dans ce cas, nom source et nom destination doivent être différents.

La commande suivante permet de copier les fichiers texte dans le répertoire `sousrep2` :

Terminal

```
yann@geonosis:~$ cp fic1.txt fic2.txt fic3.txt rep2/
sousrep2/
```

Comme nous l'avons déjà dit, si vous n'obtenez aucun message d'erreur, c'est que tout s'est bien passé.

La commande `cp` possède de nombreuses options. L'une d'elles est particulièrement intéressante, car elle permet de copier une arborescence complète en copiant les répertoires récursivement : c'est l'option `-r`.

À savoir

Vi et Emacs sont les éditeurs de texte historiques des systèmes Unix. Chacun d'entre eux possède ses adeptes et ses détracteurs. La page [2] vous donnera plus d'informations concernant cette « guerre » entre les éditeurs de texte.

Terminal

```
$ cp -r <répertoire(s)_à_copier> <répertoire_de_destination>
```

En voici un exemple :

Terminal

```
yann@geonosis:~$ cp -r rep2 rep1/sousrep1/
```

D'autres options sont, elles aussi, très intéressantes (mode interactif, copie forcée, conservation des attributs, etc.). Je vous laisse les découvrir à l'aide de la commande `man`.

Le déplacement de fichiers/répertoires s'effectue à l'aide de la commande `mv`. La syntaxe est la suivante

Terminal

```
$ mv <fichier/répertoire_à_déplacer> <répertoire_de_destination>
```

s'il s'agit de déplacer un ou plusieurs fichiers dans un autre répertoire. Dans le cas où la source et la destination se trouvent dans le même répertoire, on effectue un renommage et non un réel déplacement.

Terminal

```
yann@geonosis:~$ mv rep1/sousrep1/rep2/ rep1/
```

Finalement, voyons comment effacer des fichiers/répertoires. On utilise la commande `rm` (*remove*). La syntaxe est la suivante :

Terminal

```
$ rm <fichier_à_supprimer>
```

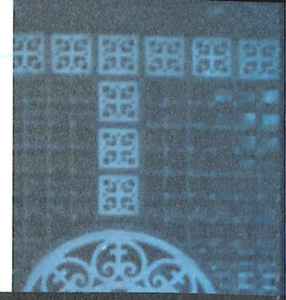
On peut aussi utiliser les caractères génériques ici :

Terminal

```
yann@geonosis:~$ rm *.txt
yann@geonosis:~$ rm -rf rep1/
```

Comme on l'a vu plus avant, l'option `-f` permet de forcer la suppression des fichiers sans confirmation de cette action. Cette option doit être utilisée avec prudence.

Illustration basée sur le GKND Creator (https://framalab.org/gknd-cre). Œuvre originale de Simon « Gee » Giraudot. Licence Creative Commons By-Sa.

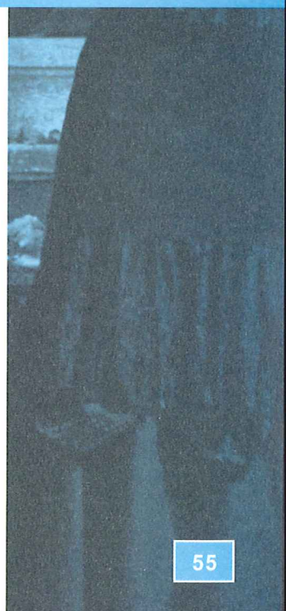


À savoir

La commande `tree` issue du paquet éponyme permet d'afficher de manière visuelle une arborescence de fichiers. Attention, ce paquet n'est pas installé par défaut.

À savoir

Attention, la commande `rename` existe, mais ne peut pas être utilisée pour le renommage d'un fichier/répertoire. Celle-ci est écrite pour effectuer le renommage de plusieurs fichiers en même temps en utilisant les expressions rationnelles.



JOUR 2

7 LIENS SYMBOLIQUES ET LIENS PHYSIQUES

Les systèmes de fichiers Unix fournissent la possibilité de créer des liens sur des fichiers ou des répertoires.

Un lien est une référence à un fichier ou un répertoire existant. On peut le manipuler comme s'il s'agissait du fichier cible. Il existe deux sortes de liens :

- les liens physiques ou liens durs : ils permettent de donner plusieurs noms à un même fichier en pointant sur un numéro d'inode. Un fichier peut donc avoir plusieurs noms, et existera tant qu'il possède au moins un nom. Les liens physiques ne peuvent pas référencer de répertoires, ni des objets d'un autre système de fichiers que celui dans lequel ils sont créés.
- les liens symboliques : ils permettent de référencer des fichiers ou des répertoires se trouvant dans d'autres systèmes de fichiers que le leur. On peut les comparer aux raccourcis que vous trouvez sous Windows par exemple. Ils sont très couramment utilisés, car il est possible de les utiliser sur les répertoires. Cependant, ils sont extrêmement dépendants de leur cible. Si cette dernière est supprimée ou déplacée, tous les liens symboliques qui s'y réfèrent deviennent invalides.

La commande pour créer un lien est `ln`. Cette commande utilise la syntaxe suivante :

Terminal
```
ln [-s] source destination
```

où `source` est le nom du fichier ou du répertoire source auquel le lien doit se référer, et `destination` est le nom du lien. L'option `-s` permet de créer un lien symbolique. Utilisée seule, la commande crée des liens physiques.

Terminal
```
yann@geonosis:~/temp$ ln fic.txt  fic_lien.txt
yann@geonosis:~/temp$ ls -i fic.txt fic_lien.txt
8297660 fic_lien.txt  8297660 fic.txt
```

La commande `ls -i` permet de vérifier que les 2 noms pointent bien sur le même inode du système de fichiers. Les données stockées sur le disque dur sont référencées par deux noms de fichiers différents.

On crée un lien symbolique par la commande suivante :

Terminal
```
yann@geonosis:~/temp$ ln -s fic.txt fic_liensymbolique.txt
yann@geonosis:~/temp$ ls -al fic*
lrwxrwxrwx 1 yann yann  7 févr. 22 17:07 fic_liensymbolique.
txt -> fic.txt
-rw-rw-r-- 2 yann yann 57 févr. 22 16:52 fic_lien.txt
-rw-rw-r-- 2 yann yann 57 févr. 22 16:52 fic.txt
```

La commande `ls -al` nous permet de vérifier que le lien symbolique est simplement un pointeur vers le fichier source. On remarque aussi que sa taille n'est pas la même que celle du fichier lié.

La suppression des liens se fait de manière similaire à celle d'un fichier en utilisant la commande `rm`. La destination n'est pas affectée en général, sauf si le lien est un lien physique et constitue la dernière référence au fichier pointé par le lien.

À savoir

Si vous supprimez la source d'un lien symbolique, ce dernier pointe alors sur un objet inexistant et l'affichage de son contenu est alors inaccessible :

```
yann@geonosis:
~/temp$ rm fic.txt
yann@geonosis:
~/temp$ ls -l fic*
lrwxrwxrwx 1 yann
yann  7 févr.
22 17:07 fic_
liensymbolique.txt
-> fic.txt
-rw-rw-r-- 1 yann
yann 57 févr. 22
16:52 fic_lien.txt
yann@geonosis:
~/temp$ cat fic_
liensymbolique.txt
cat: fic_
liensymbolique.txt:
Aucun fichier ou
dossier de ce type
```


Les liens symboliques n'ont pas de droits d'accès ni de propriétaires, les informations de sécurité de la source sont utilisées lorsqu'on accède au lien.

8 GESTION DES DROITS

Linux étant un système multi-utilisateurs, il faut que les données de chaque utilisateur soient protégées des actions que peuvent effectuer les autres utilisateurs. Cette sécurité se base sur l'identité de l'utilisateur qui réalise les commandes et sur les droits dont cet utilisateur dispose sur les objets que cette commande doit manipuler. Par exemple, vous ne pourrez effectuer la suppression d'un fichier seulement si vous possédez les droits de suppression sur ce fichier.

Le système définit, pour chaque fichier/répertoire, les droits auxquels un utilisateur aura accès. Dans la pratique, chaque utilisateur va définir les droits qu'il veut se donner sur ses propres fichiers, ainsi que les accès qu'il veut laisser aux autres utilisateurs.

De plus, il est possible de créer des groupes d'utilisateurs afin de donner simplement des droits sur les fichiers à un ensemble d'utilisateurs. Il est donc possible de définir les droits sur un fichier à trois niveaux différents :

- au niveau du propriétaire du fichier (par défaut, l'utilisateur qui l'a créé) ;
- au niveau du groupe propriétaire du fichier (par défaut, les fichiers créés par les utilisateurs qui appartiennent au groupe) ;
- au niveau de tous les autres utilisateurs (c'est-à-dire les utilisateurs qui ne sont ni les propriétaires du fichier, ni les utilisateurs du groupe du fichier).

Les droits des fichiers sont représentés par trois groupes de trois lettres indiquant les droits de chacun des niveaux précédents. Expliquons cela sur le retour de la commande `ls -al` suivante :

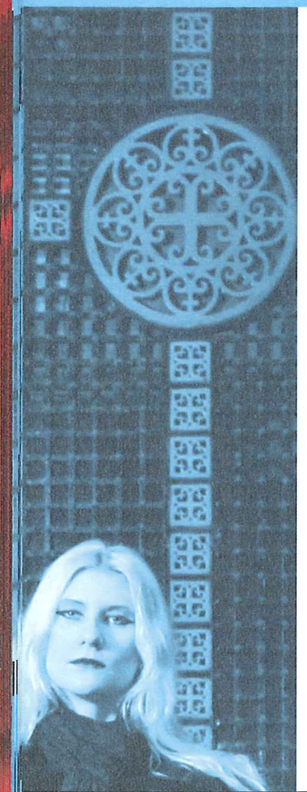
Terminal
```
yann@geonosis:~/temp$ ls -al fic_lien.txt
-rw-rw-r-- 1 yann yann 57 févr. 22 16:52 fic_lien.txt
```

Vous obtenez ainsi diverses informations dans l'ordre suivant :

- le type de fichier et les permissions qui sont octroyées aux différents utilisateurs du système ;
- le nombre de liens pointant vers le fichier ou répertoire ;
- le propriétaire du fichier ou répertoire ;
- le groupe auquel il appartient ;
- la taille du fichier (en octets) ;
- la date et l'heure de dernière modification ;
- le nom du fichier ou du répertoire.

Le premier caractère indique le type de fichier :

- `-` dans le cas d'un fichier ;
- `d` pour un répertoire ;
- `l` pour un lien symbolique ;
- `c` pour un périphérique en mode caractère ;
- `b` pour un périphérique en mode bloc.



Ensuite on retrouve trois triplets de caractères parmi `r`, `w`, `x` ou `-`. Le premier groupe correspond aux droits du propriétaire du fichier/répertoire, le second aux droits du groupe d'utilisateurs et le dernier représente les droits des autres utilisateurs du système.

Le droit de lecture est représenté par la lettre `r` (pour *Read only*), le droit d'écriture par la lettre `w` (pour *Writeable*), et le droit d'exécution par la lettre `x` (pour `eXecutable`). Le droit de lecture correspond à la possibilité d'ouvrir et de consulter un fichier, ou de lister le contenu d'un répertoire. Le droit d'écriture correspond à la possibilité de modifier un fichier, ou de créer ou supprimer un fichier d'un répertoire. Enfin, le droit d'exécution correspond à la possibilité d'exécuter un fichier contenant un programme, ou d'entrer dans un répertoire.

Si ces lettres sont présentes, cela signifie qu'elles accordent la permission qu'elles désignent. Au contraire, si vous trouvez un tiret `-` à la place d'une lettre, cela signifie que la permission n'est pas accordée.

Ainsi dans notre exemple précédent, le propriétaire et les membres du groupe ont le droit de lire et de modifier le fichier, alors que les autres utilisateurs ne peuvent que lire.

Les permissions peuvent aussi être exprimées en mode numérique :

- 1 pour le droit d'exécution (`x`) ;
- 2 pour le droit d'écriture (`w`) ;
- 4 pour le droit de lecture (`r`).

On attribue un chiffre à chaque « groupe » (propriétaire, groupe et autres) en additionnant simplement leurs droits. Par exemple, les permissions `644` sur un fichier correspondent à `-rw-r--r--`, ce qui signifie que le propriétaire du fichier a le droit de le lire et d'y écrire, tandis que le groupe et les autres utilisateurs n'ont que le droit de lecture.

À retenir

Pour utiliser `chmod`, il faut être propriétaire du fichier dont vous voulez modifier les droits d'accès.

Maintenant que nous avons détaillé cette notion de droit d'accès, voyons comment les modifier. Cette opération est réalisée à l'aide de la commande `chmod`.

On peut utiliser la commande avec les permissions numériques vues précédemment ou encore en indiquant les droits avec les lettres. Cette dernière méthode est plus longue à écrire, mais permet de paramétrer plus finement les droits. Mais de manière générale, on préférera utiliser les permissions numériques.

Par exemple, la commande suivante permet d'enlever tous les droits au groupe et aux autres utilisateurs, et de fixer les droits en lecture et écriture pour le propriétaire sur le fichier passé en paramètre :

Terminal

```
yann@geonosis:~/temp$ chmod 600 fic_lien.txt
yann@geonosis:~/temp$ ls -al fic_lien.txt
-rw------- 1 yann yann 57 févr. 22 16:52 fic_lien.txt
```

Avec l'autre méthode, nous allons utiliser les options suivantes :

- `u` = user (propriétaire) ;
- `g` = group (groupe) ;

- `o` = other (autres) ;
- `a` = all (tous).

En association avec les opérateurs suivants :

- `+` signifie : ajouter le droit ;
- `-` signifie : supprimer le droit ;
- `=` signifie : affecter le droit.

Ainsi, on pourra ajouter le droit d'écriture au groupe, ajouter le droit en lecture et enlever l'écriture aux autres, sans toucher aux droits du propriétaire à l'aide de la commande :

Terminal

```
yann@geonosis:~/temp$ chmod g+w,o+r-w fic_lien.txt
yann@geonosis:~/temp$ ls -al fic_lien.txt
-rw--w-r-- 1 yann yann 57 févr. 22 16:52 fic_lien.txt
```

Une option importante de la commande `chmod` est `-R`. Elle permet d'affecter les nouveaux droits récursivement dans les sous-répertoires du dossier passé en paramètre.

Terminal

```
yann@geonosis:~/temp$ chmod -R 755 sockets
```

À savoir

La commande `umask` (*user file creation mode mask*, masque de création de fichier par l'utilisateur) définit les permissions par défaut d'un répertoire ou d'un fichier créé.

La commande `chown` est la commande nécessitant les droits d'administrateur (root) qui permet de changer le propriétaire d'un fichier ou dossier (*change owner*).

Récapitulatif

- À l'ouverture d'un shell, le répertoire courant est positionné sur votre répertoire personnel `/home/login`. Ce répertoire personnel est contenu dans le variable de l'environnement HOME et possède une notation raccourcie « `~` ».
- La commande `uname -a` affiche les informations système et permet de retrouver la version du noyau ainsi que l'architecture utilisée.
- La commande `uptime` nous permet de savoir depuis combien de temps le système est démarré.
- Le chemin absolu prend pour référence la racine de l'arborescence, il commence donc toujours par le symbole « `/` ».
- Un chemin relatif dépend du répertoire courant où se trouve l'utilisateur au moment où il fait référence au fichier.
- Lors du listage de contenu, le `.` désigne le répertoire courant et `..` le répertoire parent.
- Du fait de la conception du système de fichiers, l'effacement d'un objet en mode console est définitif. Il n'est pas possible de le récupérer par l'intermédiaire d'une corbeille comme en mode graphique. Soyez donc très attentif à vos commandes d'effacement.
- La commande `ln` permet de créer des liens physiques et symboliques.

RÉFÉRENCES

[1] **http://www.askapache.com/linux/bash-power-prompt.html**

[2] **http://fr.wikipedia.org/wiki/Guerre_d'%C3%A9diteurs**

JOUR 3

AFFICHER, RECHERCHER ET FILTRER DES INFORMATIONS DANS VOTRE SYSTÈME

Yann Morère

Afin d'aller un peu plus loin dans la compréhension et la maîtrise de son système, il est nécessaire de pouvoir consulter et rechercher des informations dans les fichiers et des journaux (logs). La bonne exploitation des données qu'ils contiennent est la clé d'une bonne capacité à résoudre un problème éventuel. Généralement, vous aurez besoin des droits d'administration pour pouvoir accéder à certains fichiers protégés. Cette journée sera consacrée à ces activités.

JOUR 3

1 DEVENIR ADMINISTRATEUR/ROOT

À savoir

Il est important de ne jamais utiliser le compte administrateur autrement que pour des actions ponctuelles d'administration. On ne travaille pas avec le compte administrateur, c'est dangereux. Une mauvaise action peut entraîner la panne du système. Pourtant, c'est ce que vous faites constamment sous Windows...

Une grande partie des commandes que nous allons utiliser dans la suite ne sont vraiment performantes que si on les utilise en tant qu'administrateur afin d'avoir les droits d'accès aux fichiers protégés. Cet utilisateur privilégié (sûrement vous-même d'ailleurs) a réalisé la configuration initiale du système, à l'aide des assistants d'installation. Mais il va aussi s'occuper de la maintenance : création de comptes d'utilisateurs, installation/désinstallation de programmes, paramétrage du matériel, etc. L'administrateur est une personne ayant un compte d'utilisateur avec des privilèges supplémentaires. En effet, la plupart des fichiers de configuration, journaux du système sont protégés des utilisateurs standards afin d'éviter tout problème. Seul l'administrateur (root) sera autorisé à les modifier (parfois même consulter). Toutes ces tâches font partie de ce que l'on appelle l'administration du système.

Sur une distribution comme Ubuntu ou dérivés, le compte créé à l'installation du système possède les droits d'administration. Cependant, ces droits ne sont pas actifs par défaut. Il faut utiliser la commande **sudo** (« *substitute user do* ») pour exécuter une commande avec les droits administrateur. Cette commande permet de donner de façon permanente des droits à un ou plusieurs utilisateurs sur un ou plusieurs programmes (sauvegarde, arrêt machine, etc.).

sudo sert à exécuter une commande en tant qu'administrateur ou un autre utilisateur. Pour cela, elle se base sur une politique de sécurité. Par défaut, celle-ci utilise le plugin **sudoers** dont la configuration se trouve dans le fichier **/etc/sudoers**. Ce fichier est composé de règles qui définissent les comptes qui peuvent accéder aux droits d'administration. Il est possible de configurer très finement les accès en fonction des utilisateurs et des programmes. Voici le fichier de configuration par défaut sur Ubuntu :

Fichier

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

À savoir

Par défaut, sous Ubuntu, l'accès direct au compte système (root) est désactivé. La logique du système est d'utiliser sudo pour effectuer toutes les tâches administratives. Il est fortement déconseillé d'activer l'accès et d'utiliser directement le compte root sous Ubuntu. Ceci n'est pas le cas sous Debian par exemple.

Ainsi, comme mon compte fait partie du groupe « **adm** » (cela a été configuré automatiquement à l'installation du système), je peux utiliser la commande **sudo** :

Terminal

```
yann@geonosis:~$ cat /etc/sudoers
cat: /etc/sudoers: Permission non accordée
yann@geonosis:~$ sudo cat /etc/sudoers
[sudo] password for yann:
#
# This file MUST be edited with the 'visudo' command
as root.
```


On remarque que la première commande ne fonctionne pas, car je n'ai pas les droits d'accès sur ce fichier. Par contre, en utilisant la commande **sudo**, je peux visualiser le contenu du fichier. Bien entendu, il faut saisir le mot de passe de connexion afin de s'authentifier auprès du système.

Dans le cas où vous devez utiliser les droits d'administration pendant un certain temps, l'utilisation de la commande **sudo** peut devenir fastidieuse. On pourra alors lancer un shell administrateur à l'aide de la commande suivante.

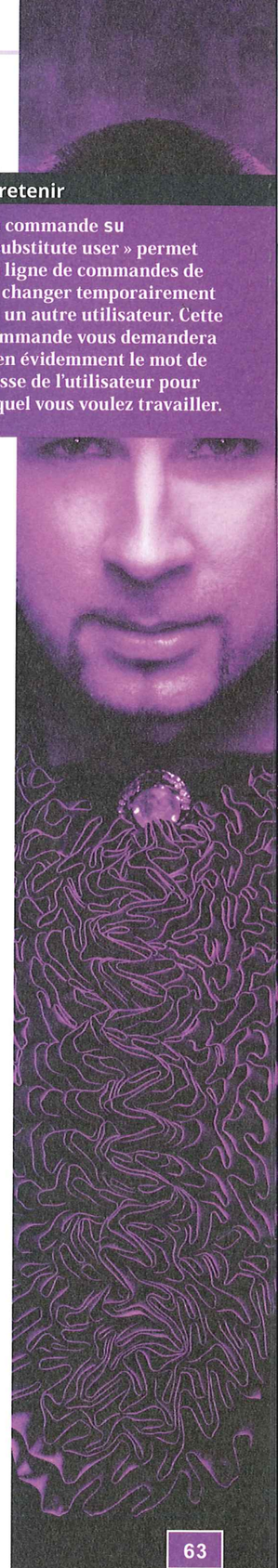
Terminal

```
yann@geonosis:~$ sudo -s
[sudo] password for yann:
root@geonosis:~# exit
exit
yann@geonosis:~$
```

On remarque le **#** final qui nous indique que l'on travaille avec le compte administrateur. On sort de ce shell à l'aide de la commande **exit**.

À retenir

La commande su « substitute user » permet en ligne de commandes de se changer temporairement en un autre utilisateur. Cette commande vous demandera bien évidemment le mot de passe de l'utilisateur pour lequel vous voulez travailler.



2 AFFICHER LE CONTENU D'UN FICHIER

Afin de pouvoir résoudre un problème, il est souvent nécessaire de consulter le contenu des fichiers journaux. Pour cela, nous allons utiliser des commandes d'affichage. Les plus usuelles sont **cat**, **more** et **less**. Elles s'utilisent en passant le fichier en paramètre.

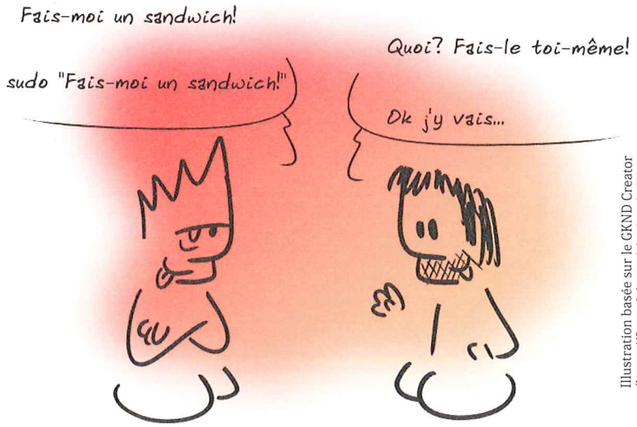
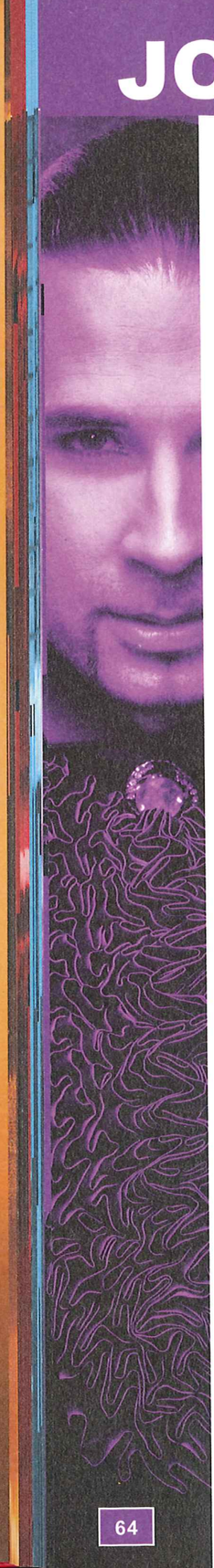


Illustration basée sur le GKND Creator (https://framalab.org/gknd-cre). Œuvre originale de Simon « Gee » Giraudot. Licence Creative Commons By-Sa.

Nous avons déjà vu **cat** au jour 2. Elle permet d'afficher le contenu d'un fichier. Cependant ce n'est pas sa fonction première. Normalement, elle sert à concaténer des fichiers et envoyer le résultat sur la sortie standard. On en détourne ici son utilisation pour l'affichage de tout le contenu du fichier en une seule fois dans le



terminal. Ensuite, elle rend la main immédiatement. Cette commande est à éviter dans le cas de fichiers très volumineux, car vous subirez un défilement rapide du contenu dans le terminal sans pouvoir le lire ! On notera que le paramètre **-n** permet d'afficher les numéros de ligne.

La commande **more** permet de visualiser le contenu du fichier page par page. On passe d'une page à l'autre à l'aide de la touche [Espace]. Il est aussi possible d'avancer ligne par ligne avec la touche [Return]. On quitte l'affichage à l'aide de la touche [q]. On lance une recherche par le caractère [/] suivi du texte désiré. Le caractère [n] permet de rechercher les occurrences suivantes du texte et [b] ou [N] les occurrences précédentes. Bien que très pratique, on lui préférera **less** bien plus complète.

La commande **less**, à la différence de **more**, ne charge pas le fichier dans son intégralité avant de commencer à l'afficher. C'est pourquoi il est préférable d'utiliser **less** plutôt que **more** dans le cas de gros fichiers. La commande **man**, non sans humour nous indique que « less - opposite of more ». Il s'agit d'un programme similaire à **more**, mais avec plus de fonctionnalités. Voici les principaux raccourcis :

- [Espace] : affiche la suite du fichier. La touche [Espace] fait défiler le fichier vers le bas d'un « écran » de console ;
- [Entrée] : affiche la ligne suivante. Cela permet donc de faire défiler le fichier vers le bas ligne par ligne. Il est aussi possible d'utiliser les touches [Flèches] et [PageUp] [PageDown] pour se déplacer dans le fichier ;
- [d] : affiche les onze lignes suivantes (soit une moitié d'écran). C'est un peu l'intermédiaire entre [Espace] (tout un écran) et Entrée (une seule ligne) ;
- [b] : retourne en arrière d'un écran ;
- [y] : retourne d'une ligne en arrière ;
- [u] : retourne en arrière d'une moitié d'écran (onze lignes) ;
- [q] : arrête la lecture du fichier et quitte **less** ;
- [=] : indique les numéros des lignes affichées à l'écran et pourcentage ;
- [/] : suivi du texte désiré permet de lancer une recherche. Il est aussi possible d'utiliser les expressions régulières que nous verrons plus loin ;
- [n] : permet de rechercher les occurrences suivantes du texte ;
- [N] : permet de rechercher les occurrences précédentes du texte.

Les commandes **head** et **tail** permettent respectivement d'afficher le début et la fin d'un fichier.

Terminal

```
yann@geonosis:~$ head /var/log/syslog
```

La commande précédente affiche les 10 premières lignes du fichier **/var/log/syslog**. De manière similaire, la commande suivante permet d'afficher les 10 dernières lignes de ce fichier :

Terminal

```
yann@geonosis:~$ tail /var/log/syslog
```

Il est possible d'utiliser l'option **-n** avec ces 2 commandes pour fixer le nombre de lignes à afficher :

Terminal

```
yann@geonosis:~$ tail -n 1 /var/log/syslog
Feb 24 12:09:01 geonosis CRON[26007]: (root) CMD (   [ -x /usr/
lib/php5/maxlifetime ] && [ -x /usr/lib/php5/sessionclean ] &&
[ -d /var/lib/php5 ] && /usr/lib/php5/sessionclean /var/lib/php5
$(/usr/lib/php5/maxlifetime))
yann@geonosis:~$ head -n 1 /var/log/syslog
Feb 24 08:00:44 geonosis anacron[26484]: Job 'cron.daily'
terminated
yann@geonosis:~$
```

Une autre option très intéressante pour la commande **tail** est **-f** (**--follow**). Ce paramètre indique de suivre l'évolution de la fin du fichier. Dès qu'une nouvelle donnée y est inscrite, elle est affichée immédiatement à l'écran. Le raccourci [Ctrl]+[C] permet de quitter la commande **tail**. C'est extrêmement utile pour suivre un fichier journal afin d'y détecter les erreurs éventuelles :

Terminal

```
yann@geonosis:~$ tail -f -n 3 /var/log/syslog
```

La commande précédente affiche les 3 dernières lignes du fichier **syslog**.

La commande **touch** permet de modifier la date d'accès et la date de modification de chaque fichier passé en argument. Par défaut, ces horodatages sont remplacés par la date et l'heure courantes, mais l'option **-t** suivie d'une date au format **MMJJhhmm** (mois, jour, heures, minutes) permet d'utiliser cette dernière à la place de la date courante :

Terminal

```
yann@geonosis:~/temp$ ls -al ancien-fichier
-rw-rw-r-- 1 yann yann 46 févr. 24 14:12 ancien-fichier
yann@geonosis:~/temp$ touch -t 02221400 ancien-fichier
yann@geonosis:~/temp$ ls -al ancien-fichier
-rw-rw-r-- 1 yann yann 46 févr. 22 14:00 ancien-fichier
```

La commande **file** permet de déterminer le type d'un fichier. Elle ne se base pas sur l'extension de ce dernier, mais sur son contenu.

Terminal

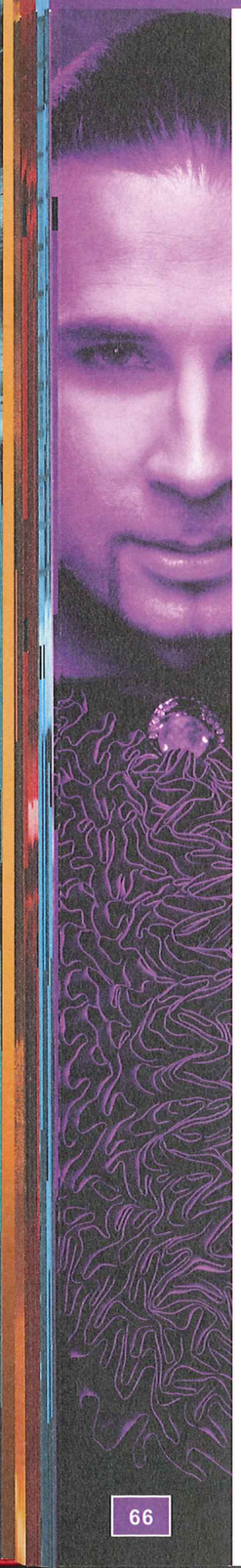
```
yann@geonosis:~$ file style.css
style.css: ASCII text, with very long lines
yann@geonosis:~$ file /opt/vagrant/bin/vagrant
/opt/vagrant/bin/vagrant: a /usr/bin/env bash script, ASCII text
executable
yann@geonosis:~$ file alias_sudo.png
alias_sudo.png: PNG image data, 800 x 600, 8-bit/color RGBA,
non-interlaced
```

La commande **wc** (*word count*) permet de compter le nombre de caractères, de mots et de lignes du fichier passé en paramètre :

Terminal

```
yann@geonosis:~/temp$ wc ancien-fichier
 7  7 46 ancien-fichier
```

JOUR 3



La commande **sort** permet de trier le contenu d'un fichier passé en paramètre :

Terminal

```
yann@geonosis:~/temp$ sort ancien-fichier
ceci
exemple
grand
intérêt
n'est
qu'un
sans
yann@geonosis:~/temp$ sort -r ancien-fichier
sans
qu'un
n'est
intérêt
grand
exemple
ceci
```

La commande **cut** permet de « couper » une partie d'un fichier et d'afficher le résultat dans la console. Par exemple, la commande suivante permet de récupérer la première colonne du fichier **/etc/passwd** qui contient tous les « login » :

Terminal

```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd
root
daemon
bin
sys
[...]
yann
```

L'option **-d** permet de définir le caractère délimiteur de champs, et l'option **-f** permet de spécifier le numéro de champ à conserver.

Parfois, il est nécessaire de comparer des fichiers pour savoir s'ils sont identiques ou non, ou bien de mémoriser uniquement les modifications apportées à un fichier pour pouvoir ensuite les réappliquer à volonté. Pour cela, nous avons à notre disposition des outils comme **cmp**, **diff** et **patch**.

La commande **cmp** permet de tester rapidement si deux fichiers sont différents. Utilisée sans option, la commande s'arrête en affichant la position de la première différence trouvée :

Terminal

```
yann@geonosis:~/temp$ cmp retrogrameB+.c retrogrameB+.c
yann@geonosis:~/temp$ cmp retrogrameB+.c retrogrameP2.c
retrogrameB+.c retrogrameP2.c sont différents: octet 3821, ligne 77
```

Avec ce programme, la comparaison est basique et s'opère au niveau des octets.

Il est possible de comparer des fichiers en prenant en compte le contexte. Pour cela, on utilisera les commandes **diff**, **sdiff** (pour la comparaison côte à côte des fichiers) ou encore **diff3** dans le cas de 3 fichiers à comparer.

Dans l'exemple suivant, la commande **diff** analyse le contenu des fichiers ou répertoires et affiche les différences :

Terminal

```
yann@geonosis:~/Documents/articles_lpmag/article_mook_shell/
images/GG$ diff . utilisees/
Seulement dans .: 2be_or_not_to_be.png
Seulement dans .: alias_sudo.png
Seulement dans .: available_update2.png
et dresse la liste de toutes les différences relevées.
```

La commande permet surtout d'afficher les différences entre deux fichiers semblables (généralement entre deux versions d'un même fichier).

Terminal

```
yann@geonosis:~/temp$ diff ancien-fichier nouveau-fichier
2,3c2,3
< n'est
< qu'un
---
> est
> un
5c5,6
< sans
---
> ayant
> un
```

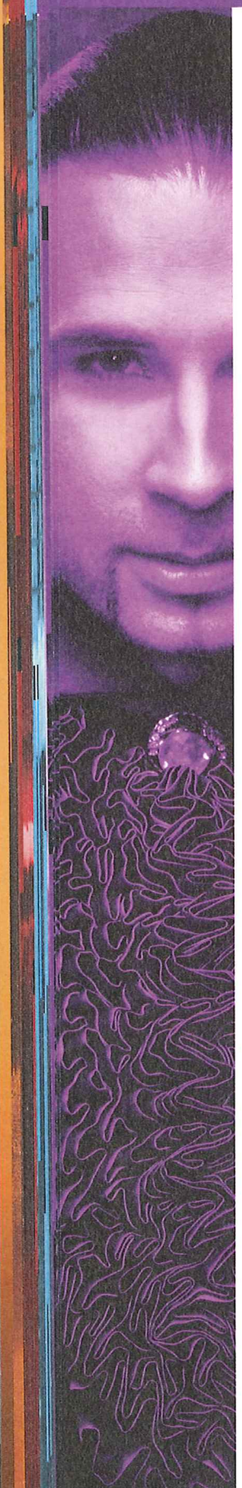
Sur la sortie, les lignes précédées de « **<** » se rapportent au premier fichier, et les lignes précédées de « **>** » se rapportent au second. « 2,3c2,3 » indique que les lignes 2 et 3 du premier fichier sont changées par rapport aux lignes 2 et 3 du second.

Cependant on utilise généralement la version unifiée de **diff** à l'aide de l'option **-u** :

Terminal

```
yann@geonosis:~/temp$ diff -u ancien-fichier nouveau-fichier
--- ancien-fichier  2015-02-24 14:12:31.309733203 +0100
+++ nouveau-fichier 2015-02-24 14:12:51.937733667 +0100
@@ -1,7 +1,8 @@
 ceci
-n'est
-qu'un
+est
+un
 exemple
-sans
+ayant
+un
 grand
 intérêt
```

Ici les signes **-** marquent les lignes ayant été supprimées lors du passage de **ancien-fichier** à **nouveau-fichier**, et les signes **+** les lignes ayant été rajoutées. Les lignes commençant par un espace n'ont tout simplement pas été modifiées et sont incluses pour fournir un minimum de contexte pour notre compréhension, mais aussi pour la commande **patch** chargée d'appliquer automatiquement ces modifications à un troisième fichier.



Les lignes commençant par **@@** indiquent les numéros de lignes correspondant au bloc de modifications qui se trouve à la suite (utile pour la commande **patch**). Les deux lignes commençant par **---** et **+++** indiquent le nom et l'heure de modification des deux fichiers comparés.

Afin d'utiliser ultérieurement la commande **patch**, il faut stocker les différences dans un fichier. Pour cela, on utilise une redirection que l'on expliquera dans la partie suivante :

Terminal
```
yann@geonosis:~/temp$ diff -u ancien-fichier nouveau-fichier > fichier.diff
```

ensuite on applique le « patch » sur une copie de l'ancien fichier :

Terminal
```
yann@geonosis:~/temp$ cp ancien-fichier troisieme-fichier
yann@geonosis:~/temp$ patch -i fichier.diff troisieme-fichier
patching file troisieme-fichier
yann@geonosis:~/temp$ diff troisieme-fichier nouveau-fichier
```

On vient d'appliquer les différences entre **ancien-fichier** et **nouveau-fichier** au **troisieme-fichier**, la dernière commande **diff** ne renvoie donc rien.

3 LES REDIRECTIONS

Nous avons vu en introduction les trois différents flux de données ouverts lors de l'exécution d'une commande : entrée, sortie et erreur. Par défaut, l'entrée standard est le clavier, la sortie et l'erreur sont l'écran. Il est possible de modifier ce comportement par l'intermédiaire des opérateurs de redirection. On utilise les redirections lorsque l'on ne souhaite pas que le retour d'une commande aille directement vers la sortie standard. On peut envoyer le résultat d'une commande dans un fichier tiers, vers une imprimante ou tout autre périphérique ou encore utiliser ce résultat en tant que paramètre d'une nouvelle commande.

Voici l'ensemble des opérateurs de redirections disponibles :

- **>** : redirige dans un fichier et l'écrase s'il existe déjà ;
- **>>** : redirige à la fin d'un fichier et le crée s'il n'existe pas.
- **<** : envoie le contenu d'un fichier à une commande ;
- **<<** : passe la console en mode saisie au clavier, ligne par ligne. Toutes ces lignes seront envoyées à la commande lorsque le mot-clé de fin aura été écrit.
- **2>** : redirige les erreurs dans un fichier (s'il existe déjà, il sera écrasé) ;
- **2>>** : redirige les erreurs à la fin d'un fichier (s'il n'existe pas, il sera créé) ;
- **2>&1** : redirige les erreurs au même endroit et de la même façon que la sortie standard.
- **|** : branche la sortie standard de la commande de gauche sur l'entrée standard de la commande de droite.

- **||** : exécute la commande suivante si la première a échoué.
- **&&** : n'exécute la commande suivante que si la première a réussi.

Voyons maintenant quelques exemples d'utilisation.

Commençons par récupérer la liste des logins de notre système dans un fichier :

Terminal
```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd > logins.txt
```

La redirection précédente s'opère avec écrasement de la cible. Si nous désirons ajouter au contenu existant, on utilise la commande :

Terminal
```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd >> logins.txt
```

Maintenant, si le fichier passé en paramètre n'existe pas, nous redirigeons la sortie d'erreur dans un fichier :

Terminal
```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd2 > logins.txt 2>
erreur.log
yann@geonosis:~/temp$ cat erreur.log
cut: /etc/passwd2: Aucun fichier ou dossier de ce type
```

On fait de même avec l'ajout en fin de fichier d'erreur :

Terminal
```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd2 > logins.txt
2>> erreur.log
yann@geonosis:~/temp$ cat erreur.log
cut: /etc/passwd2: Aucun fichier ou dossier de ce type
cut: /etc/passwd2: Aucun fichier ou dossier de ce type
```

Si l'on ne désire pas séparer les informations dans deux fichiers différents, il est possible de fusionner les sorties dans un seul et même fichier. La redirection **2>&1** redirige toute la sortie d'erreurs dans la sortie standard qui est elle-même redirigée dans un fichier :

Terminal
```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd2 > logins.txt
2>&1
yann@geonosis:~/temp$ cat logins.txt
cut: /etc/passwd2: Aucun fichier ou dossier de ce type
```

Lisons maintenant depuis un fichier à la place du clavier.

Terminal
```
yann@geonosis:~/temp$ cat < logins.txt
root
daemon
bin
[...]
yann
```

Voyons maintenant comment lire des données saisies ligne par ligne au clavier jusqu'à la saisie d'un mot-clé. Ici, on utilisera pour mot-clé « **EOF** » pour « *End Of File* ».

JOUR 3

À savoir

Le mot-clé `EOF` est choisi arbitrairement. On aurait pu choisir `FIN` par exemple ou toute autre chaîne de caractères.

Terminal

```
yann@geonosis:~/temp$ sort << EOF
> zz
> ff
> gg
> hh
> kk
> aa
> yy
> EOF
aa
ff
gg
hh
kk
yy
zz
```

Cette commande permet de trier toutes les lignes entrées au clavier. On termine la saisie en entrant la chaîne « EOF ».

Utilisons maintenant le chaînage des commandes pour trier la liste des logins :

Terminal

```
yann@geonosis:~/temp$ cut -d : -f 1 /etc/passwd | sort
> logins_tries.txt
yann@geonosis:~/temp$ cat logins_tries.txt
avahi
avahi-autoipd
backup
[...]
yann
```

La sortie de la commande **`cut`** est envoyée à l'entrée de la commande **`sort`** qui place le résultat dans un fichier texte par redirection. Vous commencez maintenant réellement à voir la puissance de la ligne de commandes ainsi que son intérêt.

La commande suivante permet de trouver en quelques secondes les 10 répertoires les plus volumineux du répertoire courant :

Terminal

```
yann@geonosis:~/Documents/articles_lpmag$ du | sort -nr |
head
4824928     .
939684 ./article_openshot_imagemovie
909044 ./article_openshot_imagemovie/work
789928 ./article_openshot_imagemovie/work/voyage_bresil
524684 ./article_raspicade
450192 ./article_astro
415112 ./article_impress.js
390152 ./article_impress.js/samples
383812 ./article_impress.js/samples/autonomic2012
383684 ./article_impress.js/samples/autonomic2012/images
```


La commande **`du`** (*disk use*) renvoie la place utilisée par les fichiers et répertoires. On passe cette sortie à la commande **`sort`** qui va les trier numériquement en ordre inverse. Cette nouvelle sortie est ensuite envoyée à la commande **`head`** pour n'en afficher que les 10 premiers.

Il est aussi possible d'enchaîner plusieurs commandes, en les séparant par un point-virgule : **`commande1; commande2; commande3;`** etc. Les commandes sont alors exécutées les unes à la suite des autres.

Terminal

```
yann@geonosis:~/temp$ whoami;date;time
yann
mardi 24 février 2015, 16:29:09 (UTC+0100)

real    0m0.000s
user    0m0.000s
sys     0m0.000s
```

Le symbole **`&`**, contrairement au point-virgule, permet d'exécuter les commandes simultanément : **`commande1 & commande2 & commande3 &`** etc.

La commande suivante permet de lancer simultanément les trois applications et rend la main à l'utilisateur dans la console :

Terminal

```
yann@geonosis:~$ geany & inkscape & gimp-2.8 &
[3] 4327
[4] 4328
[5] 4329
yann@geonosis:~$
```

On a parfois besoin de savoir si une commande a réussi ou non avant d'en lancer une autre. Les opérateurs **`&&`** et **`||`** permettent, respectivement, de lancer une commande si (et seulement si) la précédente a réussi, respectivement échoué.

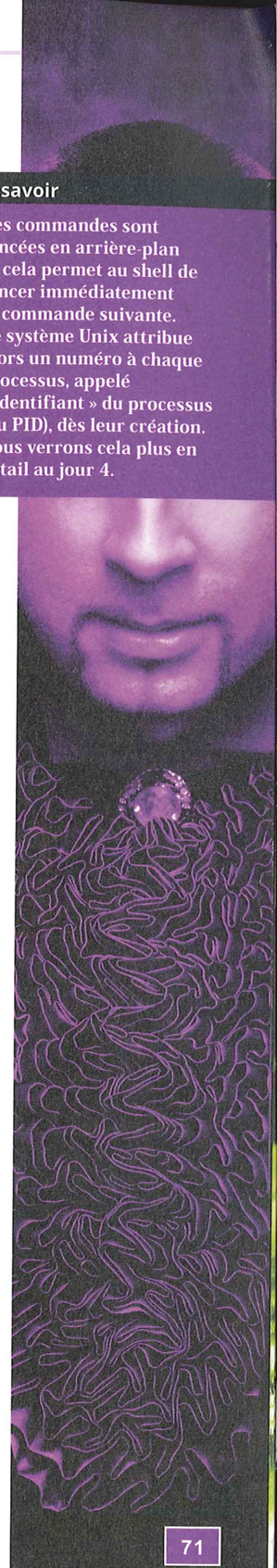
Terminal

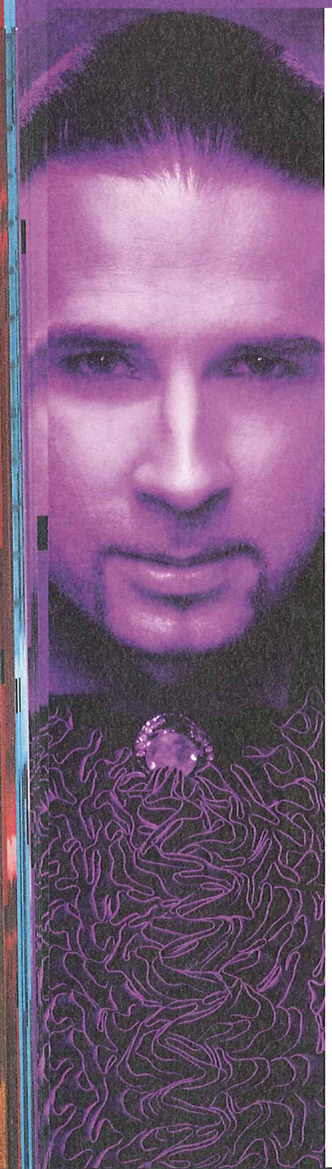
```
yann@geonosis:~/temp$ ls logins.txt 2> /dev/null &&
tail -n 3 logins.txt
logins.txt
dnsmasq
nvidia-persistenced
ntp
yann@geonosis:~/temp$ ls logins2.txt 2> /dev/null &&
tail -n 3 logins.txt
yann@geonosis:~/temp$
```

La première commande affiche les trois dernières lignes du fichier, car il existe, elle n'affiche rien le cas échéant.

À savoir

Ces commandes sont lancées en arrière-plan et cela permet au shell de lancer immédiatement la commande suivante. Le système Unix attribue alors un numéro à chaque processus, appelé « identifiant » du processus (ou PID), dès leur création. Nous verrons cela plus en détail au jour 4.





Terminal

```
yann@geonosis:~/temp$ ls logins2.txt 2> /dev/null &&
tail -n 3 logins.txt || echo "Pas de fichier"
Pas de fichier
yann@geonosis:~/temp$ ls logins.txt 2> /dev/null &&
tail -n 3 logins.txt || echo "Pas de fichier"
logins.txt
dnsmasq
nvidia-persistenced
ntp
yann@geonosis:~/temp$
```

Cette commande permet d'afficher un message si le fichier n'existe pas, et exécute la commande **tail** le cas échéant.

4 RECHERCHER DES FICHIERS

Même si votre arborescence est bien organisée, vu le nombre de fichiers que l'on est amené à stocker, il arrive un moment où l'on a besoin de rechercher un fichier particulier. Là encore, des commandes existent pour vous faciliter la tâche.

La commande **which** permet de localiser une commande dans l'arborescence du système :

Terminal

```
yann@geonosis:~$ which inkscape
/usr/local/bin/inkscape
```

La commande **whereis** permet de localiser les fichiers exécutables, les fichiers sources et les pages de manuel d'une commande dans le système de fichiers :

Terminal

```
yann@geonosis:~$ whereis inkscape
inkscape: /usr/bin/inkscape /usr/bin/X11/inkscape /usr/
local/bin/inkscape /usr/share/inkscape /usr/share/man/
man1/inkscape.1.gz
```

On peut rechercher des fichiers à l'aide de la commande **locate**. Cette commande fait partie du paquet **mlocate** qu'il faudra installer. Elle permet de rechercher des fichiers par leur nom en utilisant une base de données qu'il faudra préalablement remplir à l'aide de la commande **updatedb** :

Terminal

```
yann@geonosis:~/temp$ sudo updatedb
[sudo] password for yann:
yann@geonosis:~/temp$ locate tp_rpi_full.pdf
/data/data2/yann/Bureau/tp1_rpi/tp_raspberrypi/tp_rpi_
full.pdf
```

À savoir

On utilise le terme motif, car il est possible d'utiliser les expressions régulières avec `locate`. Elle recherche le motif passé en paramètre dans la base de données et renvoie tous les occurrences trouvées.

locate écrira sur le terminal l'ensemble des chemins de fichiers dont le nom contient le motif passé en paramètre.

L'autre commande la plus utilisée pour trouver des fichiers est **find**. Elle est plus complète, car elle ne se base pas exclusivement sur le nom, et permet des recherches suivant les options suivantes :

- **-name** : recherche par nom de fichier ;
- **-type** : recherche par type de fichier ;
- **-user** : recherche par propriétaire ;
- **-group** : recherche par appartenance à un groupe ;
- **-size** : recherche par taille de fichier ;
- **-atime** : recherche par date de dernier accès ;
- **-mtime** : recherche par date de dernière modification ;
- **-ctime** : recherche par date de création ;
- **-perm** : recherche par autorisations d'accès ;
- **-links** : recherche par nombre de liens au fichier.

Pour les options **-size**, **-atime**, **-mtime**, **-ctime** et **-links**, il faut spécifier une valeur, précédée par le signe **+** pour « supérieur à », **-** pour « inférieur à », ou rien pour « égal à ».

Terminal

```
yann@geonosis:~/temp$ find . -size +50M -ctime -5
-type f
./inkscape/src/2geom/lib2geom.a
./inkscape/src/inkscape
./inkscape/src/inkview
./Adafruit-Pi-Kernel-o-Matic/custom_
kernel_1.20150223-1.tar.gz
./pibootstrap/libchromiumcontent.so
```

La commande précédente affiche les fichiers dont les dernières modifications remontent à moins de 5 jours, pesant plus de 50Mo et qui sont des fichiers.

Il est possible d'utiliser la sortie de la recherche de la commande **find** pour effectuer des actions à l'aide de l'option **-exec**. La commande suivante trouve dans le répertoire courant les fichiers de plus de 500Mo et en affiche les détails :

Terminal

```
yann@geonosis:~/temp$ find . -size +500M -type f -exec
ls -l {} \;
```

Cette très pratique par exemple pour trouver les fichiers volumineux et les effacer en une seule ligne de commandes :

Terminal

```
yann@geonosis:~/temp$ find . -size +500M -exec
rm {} \;
```

À savoir

Les {} remplacent les fichiers listés par la commande `find`. Ils arrivent en paramètre de la commande `rm` et sont donc effacés.

JOUR 3

5 RECHERCHER DU CONTENU DANS DES FICHIERS

À savoir

On notera que les commandes `egrep` et `fgrep` sont obsolètes, mais ont été conservées par souci de compatibilité.

Le rôle de la commande **`grep`** est de rechercher un mot dans un fichier et d'afficher les lignes dans lesquelles ce mot a été trouvé. Cette commande peut être utilisée de manière très simple ou en association avec les expressions régulières. Elle possède des variantes, **`fgrep`** **`egrep`** et **`rgrep`** qui ne sont en fait que des alias vers la commande **`grep`** associée respectivement aux options **`-F`**, **`-E`** et **`-r`**.

La syntaxe de la commande **`grep`** est la suivante :

Fichier

```
grep motif fichier
```

Par exemple, la commande :

Terminal

```
yann@geonosis:~/temp$ grep fopen retrogrameP2.c
        if((fp = fopen("/proc/cmdline", "r"))) {
        if((fp = fopen("/proc/cpuinfo", "r"))) {
```

permet de recherche la chaîne de caractères « fopen » dans le fichier **`retrogrameP2.c`**.

Si le motif contient des espaces, il faudra le protéger par des guillemets doubles « " » :

Terminal

```
yann@geonosis:~/temp$ grep "O_WRONLY | O_NONBLOCK"
retrogrameP2.c
        if((fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK))
< 0)
        if((fd = open("/dev/input/event0", O_WRONLY | O_
NONBLOCK)) < 0)
```

Si vous désirez faire abstraction de la casse des caractères, il faudra utiliser l'option **`-i`** (**`--ignore-case`**) et **`-n`** (**`--line-number`**) pour afficher les numéros de lignes :

Terminal

```
yann@geonosis:~/temp$ grep -i -n retrogame  retrogrameP2.c
2:ADAFRUIT RETROGAME UTILITY: remaps buttons on Raspberry Pi
GPIO header
4:Retrogame is interrupt-driven and efficient (usually under
0.3% CPU use)
12:Must be run as root, i.e. 'sudo ./retrogame &' or configure
init scripts
```

L'option **`-v`** (**`--invert-match`**) permet d'ignorer un mot. Ainsi, vous récupérez toutes les lignes ne contenant pas ce mot.

Terminal

```
yann@geonosis:~/temp$ grep -i -n -v ";"  retrogrameP2.c | head -n 2
1:/*
2:ADAFRUIT RETROGAME UTILITY: remaps buttons on Raspberry Pi GPIO header
```

Ici, on affiche les 2 premières lignes du fichier ne contenant pas de point-virgule.

La dernière option que nous allons voir est très utile dans le cas ou vous ne savez pas dans quel fichier se trouve la chaîne recherchée. L'option **`-r`** (*recursive*) permet de rechercher dans tous les fichiers et sous-dossiers. Il faudra indiquer en dernier paramètre le nom du répertoire dans lequel la recherche doit être faite (et non pas le nom d'un fichier). Cette commande est équivalente à la commande **`rgrep`** :

Terminal

```
yann@geonosis:~/temp/inkscape$ grep -r TODO src/ | tail -n 5
yann@geonosis:~/temp/inkscape$ rgrep TODO src/ | tail -n 5
src/selection-chemistry.cpp:// TODO determine if this is intentionally
different from SPObject::getPrev()
src/selection-chemistry.cpp:            if (!object) { // TODO check if
this happens in practice
src/selection-chemistry.cpp:    // TODO: avoid roundtrip via file
src/selection-chemistry.cpp:        // TODO: avoid unnecessary roundtrip
between data URI and decoded pixbuf
src/knot.h:    //TODO: all the members above should eventualle become
private, accessible via setters/getters
```

Récapitulatif

- La commande **`sudo`** permet de donner de façon permanente des droits à un ou plusieurs utilisateurs sur un ou plusieurs programmes.
- Les commandes **`cat`**, **`more`** et **`less`** permettent l'affichage du contenu d'un fichier.
- Les commandes **`head`** et **`tail`** permettent respectivement d'afficher le début et la fin d'un fichier.
- La commande **`file`** permet de déterminer le type d'un fichier.
- La commande **`diff`** permet de comparer le contenu de fichiers et de répertoires.
- La commande **`patch`** permet de réappliquer rapidement des changements entre 2 fichiers.
- Les redirections permettent d'envoyer le résultat d'une commande dans un fichier tiers, vers une imprimante ou tout autre périphérique.
- La commande **`find`** ne se base pas exclusivement sur un nom pour effectuer une recherche et permet des recherches suivant de nombreuses options.
- La commande **`grep`** permet de rechercher un mot dans un fichier et d'afficher les lignes dans lesquelles ce mot a été trouvé.

JOUR 4

GÉRER ET SURVEILLER VOTRE SYSTÈME

Yann Morère

Allons maintenant un peu plus loin dans l'administration de notre système. Il s'agira d'apprendre à mettre en œuvre les expressions rationnelles pour la création de filtres puissants, de savoir installer/supprimer de nouvelles applications et de gérer les services de votre système. Ensuite, nous verrons comment surveiller les programmes fonctionnant sur notre système et les terminer si nécessaire.

JOUR 4

1 SHELL ET CARACTÈRES SPÉCIAUX

Dans le jour 1, nous avons déjà vu 2 caractères spéciaux : * et ? utilisés comme caractères génériques. Sous Unix, il y en a de nombreux autres :

- ; : permet de séparer plusieurs commandes écrites sur une même ligne ;
- ( ) : permet de regrouper des commandes ;
- & : permet le lancement d'un processus en arrière-plan. Cela permet d'exécuter d'autres commandes pendant qu'un processus est en marche ;
- | : permet la communication par tube entre deux commandes ;
- # : introduit un commentaire. Tout ce qui suit ce caractère dans une ligne est ignoré par le shell ;
- $: entre « $(» et «) », la commande à exécuter est remplacée par son résultat. $ sert aussi à afficher le contenu d'une variable ;
- ~ : le caractère tilde (« ~ ») permet de désigner le répertoire courant de l'utilisateur ;
- \ : déspécialise le caractère qui suit, si le caractère qui suit celui-ci est un caractère spécial alors le shell l'ignorera ;
- '...' : définit une chaîne de caractères qui ne sera pas évaluée par le shell ;
- «...» : définit une chaîne de caractères dont les variables seront évaluées par le shell ;
- '...' : définit une chaîne de caractères qui sera interprétée comme une commande et remplacée par la chaîne qui serait renvoyée sur la sortie standard à l'exécution de ladite commande.

Voyons rapidement quelques exemples d'utilisation des 4 derniers items de la liste précédente. Les autres ayant déjà été abordés.

On commence par la déspécialisation à l'aide du caractère \. Pour cela, nous allons utiliser la commande echo qui permet d'afficher une ligne de texte.

Terminal

```
yann@geonosis:~$ echo B\?n\*jour\; \\ \\\\ \(\"Hello\" in \'english\'\).
B?n*jour; \ \\ ("Hello" in 'english').
```

Dans la commande précédente, on remarque que \\ a pour effet d'afficher \.

On peut aussi afficher les caractères spéciaux par l'intermédiaire des '...'. Ainsi, il ne seront pas interprétés par le shell :

Terminal

```
yann@geonosis:~$ echo '?\;()"*'
?\;()"*
```

Si on veut mélanger des caractères spéciaux, des variables, des commandes, il faut utiliser les guillemets doubles (« « »). Seuls sont interprétés les méta-caractères « $ » (commandes et variables), « \ » (déspécialisation) et « ' » (commandes).

Terminal

```
yann@geonosis:~$ echo "je suis dans le répertoire $(pwd) \$ \\ 'pwd'."
je suis dans le répertoire /home/yann $ \ /home/yann.
```

On remarque ici que la syntaxe $(...) réalise la même chose que '...'.

2 LES EXPRESSIONS RÉGULIÈRES

Une expression rationnelle est une suite de caractères typographiques, le motif (*pattern* en anglais) décrivant une chaîne de caractères dans le but de la trouver dans un bloc de texte pour lui appliquer un traitement automatisé, comme un ajout, son remplacement ou sa suppression. Ces expressions seront utilisées pour réaliser des filtres très précis pour des commandes comme grep, ed, sed et l'éditeur VI par exemple.

Pour leur fonctionnement, ces expressions utilisent des métacaractères (*wild cards* ou *jokers* en anglais). Voici les fonctions de ces opérateurs :

- . : désigne n'importe quel caractère ;
- [...] : désigne tout caractère parmi ceux énumérés ;
- [^...] : désigne tout caractère excepté ceux énumérés ;
  - l'expression q[^u] recherche les mots dont q n'est pas suivi de u ;
- \< : désigne la position en début de mot ;
  - l'expression grep '\<.r' fichier recherche dans fichier tous les mots dont la seconde lettre est un r ;
- \> : désigne la position en fin de mot ;
- | (pipe) : reconnaît l'un ou l'autre des termes qu'il sépare ;
  - l'expression peu|prou|nombres? trouve « peu », « prou », « nombre » ou « nombres » ;
- + : correspond à ce qui le précède, répété au moins une ou plusieurs fois ;
  - Par exemple, l'expression ba+c trouve « bac », ou « baac », « baaac » ;
- ^ : identifie un début de ligne. Par exemple, l'expression régulière ^a va identifier les lignes commençant par le caractère a ;
  - S'il apparaît au début d'une classe de caractères, il indique que l'expression rationnelle correspond à tout caractère sauf ceux qui sont compris dans cette liste ;
- $: identifie une fin de ligne. Par exemple, l'expression régulière a$ va identifier les lignes se terminant par le caractère a ;
  - L'expression ^chaine$ identifie les lignes qui contiennent strictement la chaîne chaine ;
  - L'expression régulière ^$ identifie une ligne vide ;
- * : désigne le caractère de répétition ;
  - L'expression a* correspond aux lignes comportant 0 ou plusieurs caractères a ;
  - L'expression aa* correspond aux lignes comportant 1 ou plusieurs caractères a ;
  - L'expression régulière .* correspond à n'importe quelle chaîne de caractères ;
  - L'expression régulière [a-z][a-z]* va chercher les chaînes de caractères contenant 1 ou plusieurs lettres minuscules (de a à z) ;
  - L'expression régulière [^][^]* est équivalente à tout sauf un blanc ;
- ( ) : désigne les parenthèses de groupement (utilisées sans recourir à des séquences d'échappement), qui permettent de délimiter un ensemble d'alternatives, ou toute sous-expression rationnelle ;
  - l'expression ch(at|ien) correspond à chat ou chien ;

À savoir

La commande ed, est un éditeur de texte ligne par ligne. sed (pour *Stream EDitor*, « éditeur de flux ») est une commande permettant d'appliquer différentes transformations prédéfinies à un flux séquentiel de données textuelles.

- `{N}` : ce qui précède apparaît exactement N fois ;
- `{N,}` : ce qui précède apparaît N fois ou plus ;
- `{N,M}` : ce qui précède apparaît au moins N fois et au plus M fois ;
- `\b` : désigne le caractère de contrôle retour arrière (correction) ;
- `\t` : désigne le caractère de contrôle de tabulation horizontale ;
- `\n` : désigne le caractère de contrôle de saut de ligne ;
- `\v` : désigne le caractère de contrôle de tabulation verticale ;
- `\f` : désigne le caractère de contrôle de saut de page ;
- `\r` : désigne le caractère de contrôle de retour chariot.

Voici quelques exemples d'utilisation avec la commande `grep`. On utilise l'expression régulière à l'aide de l'option `-e` :

Terminal

```
yann@geonosis:~$ grep -e "^yann\|^www-data" /,etc/passwd
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
yann:x:1000:1000:yann,,,:/home/yann:/bin/bash
```

Cette commande permet de rechercher les lignes qui commencent par `yann` ou `www-data`.

Terminal

```
yann@geonosis:~$ grep -e "nologin$" /,etc/passwd
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

Cette commande permet de rechercher les lignes qui se terminent par `nologin`.

NOTE

On remarquera qu'il est nécessaire de déspécialiser les métacaractères dans la chaîne de caractères de l'expression régulière, afin qu'elle soit bien interprétée par la commande `grep`.

Illustration basée sur le GKND Creator (https://framalab.org/gknd-creator). Œuvre originale de Simon « Cee » Giraudot. sLicence Creative Commons By-Sa.

3 INSTALLER ET SUPPRIMER DES APPLICATIONS

L'installation d'applications est très simple sous Linux, car elles sont toutes stockées dans des dépôts sous la forme de paquets. Ainsi, à l'aide d'une simple connexion internet, il est possible d'installer des milliers de logiciels libres. En mode console, vous avez accès à pas moins de 3 programmes pour la gestion de ces paquets. Un programme de haut niveau `aptitude` qui utilise une interface graphique en mode texte, un programme de niveau intermédiaire `apt`, et un bas niveau `dpkg`. Il faut cependant avoir à l'esprit qu'`aptitude` repose sur `apt` qui lui-même utilise `dpkg` pour la gestion des paquets sur le système.

Dpkg est le programme principal de gestion des paquets. Il possède de nombreuses options et est assez complexe à maîtriser. La plupart du temps, on évitera d'y avoir recours, mis à part en cas de problème majeur avec le système de gestion des paquets.

APT (pour *Advanced Package Tool*) est le programme de gestion de paquets développé par Debian. Il fournit le programme `apt-get` qui fournit un moyen simple de retrouver et d'installer des paquets depuis plusieurs dépôts.

Les options les plus courantes de `apt-get` sont les suivantes :

- `apt-get update` : pour mettre à jour la liste des paquets connus par votre système. Il est conseillé d'exécuter cette commande régulièrement pour mettre à jour vos listes de paquets ;
- `apt-get upgrade` : pour mettre à jour tous les paquets de votre système, sans installer de paquets supplémentaires ou en supprimer (il est nécessaire d'avoir fait la mise à jour de la liste de paquets avant) ;
- `apt-get install foo` : pour installer le paquet `foo` et toutes ses dépendances (les paquets nécessaires à son bon fonctionnement) ;
- `apt-get remove foo` : pour supprimer le paquet `foo` de votre système ;
- `apt-get --purge remove foo` : pour supprimer le paquet `foo` et ses fichiers de configuration de votre système ;
- `apt-get dist-upgrade` : pour mettre à jour votre système entier, en permettant si nécessaire l'installation de paquets supplémentaires ou la suppression de paquets.

NOTE

Toutes ces opérations se font avec le compte administrateur.

La suite d'outils `apt` inclut aussi le programme `apt-cache` pour questionner les listes de paquets. Les options les plus courantes de `apt-cache` sont les suivantes :

- `apt-cache search mot` : pour trouver les paquets dont la description contient mot ;
- `apt-cache show paquet` : pour afficher des informations détaillées sur un paquet ;
- `apt-cache depends paquet` : pour afficher les dépendances d'un paquet ;

À savoir

Les utilitaires décrits dans les lignes suivantes sont spécifiques aux distributions basées sur Debian : Ubuntu, Xandros, Linux Mint (cf https://www.debian.org/misc/children-distros.fr.html pour la liste complète). Les distributions basées sur Red Hat ou SUSE possèdent d'autres utilitaires tout à fait équivalents.

- `apt-cache showpkg paquet` : pour afficher des informations détaillées des versions disponibles pour un paquet et les paquets ayant des dépendances inverses sur lui.

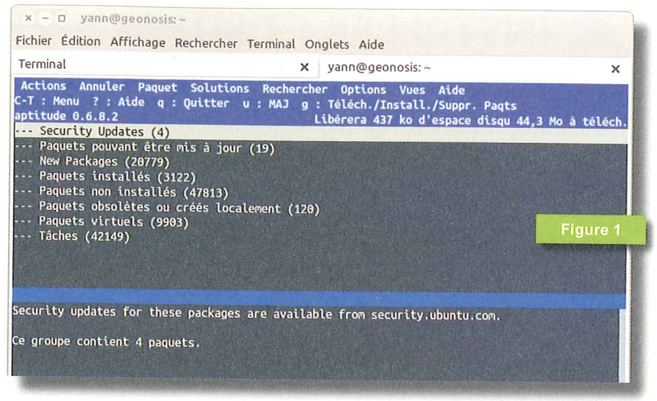


Figure 1

`aptitude` est le gestionnaire de paquet recommandé pour la gestion quotidienne des paquets en mode console. Il utilise une interface graphique en mode texte (figure 1).

Son utilisation est pratique et rapide grâce à la visualisation et la recherche de paquets. Son utilisation peut aussi se faire en ligne de commandes. La syntaxe utilisée est très similaire à celle utilisée avec `apt-get`.

- `aptitude install foo` pour installer le paquet `foo` ;
- `aptitude remove foo` pour supprimer le paquet `foo` ;
- `aptitude purge foo` pour supprimer le paquet `foo` ainsi que ses fichiers de configuration ;
- `aptitude search foo` pour rechercher les paquets contenant la chaîne de caractère `foo` ;
- `aptitude show foo` pour montrer les détails du paquets `foo`.

4 GÉRER LES SERVICES

Service est traduit en anglais par daemon (« *Disk And Execution MONitor* » ou plus récemment « *Deferred Auxiliary Executive MONitor* »). Il s'agit d'un programme réalisant des tâches de fond du système. Les services sous Linux couvrent de nombreuses fonctionnalités : gestion des imprimantes, gestion des pilotes sons, gestion des tâches périodiques, système de messagerie inter-applications, etc. La plupart du temps c'est transparent pour l'utilisateur. Néanmoins, il peut arriver d'avoir besoin de connaître l'état d'un service ou de vouloir le modifier sans relancer la machine.

On peut par exemple lister les services actifs sur la machine par la commande :

Terminal
```
yann@geonosis:~$ service --status-all
 [ + ]  acpid
 [ - ]  anacron
 [ + ]  apache2
 [ - ]  apparmor
 [ ? ]  apport
 [ + ]  atd
 [ + ]  avahi-daemon
 [ + ]  binfmt-support
 [ ? ]  binfmt-support.dpkg-new
[...]
```


Le signe `+` indique que le service est en fonctionnement, le signe `–` qu'il est arrêté. On peut vérifier l'activité d'un service par la commande suivante :

Terminal
```
yann@geonosis:~$ service  timidity status
* timidity is running
```

NOTE

Certains services sont gérés maintenant par le nouveau système Upstart. Ils seront affichés par la commande service avec le caractère ?.

On peut modifier l'état d'un service à l'aide de la commande :

Terminal
```
yann@geonosis:~$ sudo service apache2 stop
* Stopping web server apache2
```

Cela a pour effet d'arrêter le serveur web Apache. La liste des actions possibles est la suivante : `start, stop`, `force-stop`, `restart`, `reload`, `force-reload`, `status`.

Il est possible de désactiver un service pour qu'il ne soit pas exécuté au démarrage. La commande `update-rc.d` permet cette action. Par exemple, la commande :

Terminal
```
yann@geonosis:~$ sudo update-rc.d apache2 disable
```

permet de désactiver le serveur web au démarrage. La commande suivante le réactive :

Terminal
```
yann@geonosis:~$ sudo update-rc.d apache2 enable
```

Depuis 2006, sur Ubuntu, la méthode de gestion des services système utilise progressivement le système « Upstart » qui permet une plus grande souplesse. Cependant, cette migration n'est pas encore complète et vous trouverez dans le dossier `/etc/init.d/` les scripts d'initialisation des services. Il est donc possible d'utiliser le système Upstart à la place de `service` à l'aide de la commande `initctl`.

Comme pour `service`, `inictl` s'utilise comme suit :

Terminal
```
$ sudo initctl ACTION NomService
```

les actions applicables au service sont les suivantes :

- `start` : démarrer le service ;
- `stop` : arrêter le service ;
- `restart` : relancer le service ;
- `reload` : recharger le service ;
- `status` : connaître l'état du service.

Quelle que soit l'action menée sur un service, au prochain démarrage de la machine celui-ci devrait retrouver le `status` qui lui a été défini par défaut.

La commande suivante nous renvoie la liste triée des services gérés par Upstart :

Terminal
```
yann@geonosis:~$ initctl list | sort
at-spi2-registryd start/running, process 3048
dbus start/running, process 2959
gnome-keyring-gpg stop/waiting
gnome-keyring-ssh stop/waiting
[...]
```

Pour désactiver définitivement un service au démarrage avec le système upstart, il suffit de renommer le fichier de configuration.

Si vous souhaitez désactiver samba, on renommera le fichier `/etc/init/smb.conf` :

Terminal
```
$ sudo mv /,etc/init/ smb.conf /etc/init/smb.conf .noexec
```

5 GÉRER LES PROCESSUS

Un processus ou job est un programme en cours d'exécution. Celui-ci a besoin de ressources matérielles pour fonctionner (mémoire, processeur, périphériques, etc.). Il possède des caractéristiques statiques (invariantes au cours sa vie) : un numéro unique, le PID (*Process Identifier*), un propriétaire déterminant les droits d'accès du processus aux ressources, un processus parent dont il hérite la plupart des caractéristiques, un terminal d'attache pour les entrées/sorties. Il possède aussi des caractéristiques dynamiques comme la priorité, un environnement d'exécution, un état, etc.

Un processus est toujours créé par un autre processus appelé processus parent. Ainsi, tout processus a un processus parent sauf le tout premier. Ce tout premier processus est appelé `init` et son identifiant est égal à 1 (PID = 1).

On peut classer les processus en 2 types : les processus utilisateurs, issus du shell de connexion et les processus daemon assurant un service lancé au démarrage de la machine.

La commande `ps` vous permet d'obtenir la liste des processus en exécution au moment où vous lancez la commande. Cette liste est statique et n'est pas actualisée en temps réel. Lançons la commande `ps` sans argument :

Terminal
```
yann@geonosis:~$ ps
  PID TTY          TIME CMD
 1822 pts/11   00:00:00 grep
11605 pts/11   00:00:00 bash
11954 pts/11   00:00:00 ps
```

On distingue quatre colonnes : `PID` vue précédemment, `TTY` (console depuis laquelle a été lancé le processus), `TIME` qui représente la durée d'occupation du processeur par le processus et `CMD`, le programme qui a généré ce processus.

Par défaut, `ps` affiche uniquement les processus de l'utilisateur. Avec `ps -ef`, vous pouvez avoir la liste de tous les processus lancés par tous les utilisateurs sur toutes les consoles :

Terminal
```
yann@geonosis:~$ ps -ef | wc -l
384
```

Nous avons sur notre système 384 processus lancés par des utilisateurs.

La commande `ps u` affiche plus d'informations concernant chaque processus et notamment son état :

Terminal
```
yann@geonosis:~$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
yann      1285  0.0  0.0  30188  1496 pts/21   Ss+  févr.23  0:01 bash
```

Un processus peut prendre plusieurs états, qui correspondent chacun à une lettre :

- `R` (*Run*) : indique que le processus est en cours d'exécution, et qu'il mobilise le processeur ;
- `S` (*short Sleep*) : indique que le processus est en veille, mais prêt à fonctionner ;
- `T`(*sTopped*) : indique que le processus est suspendu, il ne mobilise pas le processeur ;
- `I` (*Idle*) : indique que le processus est suspendu depuis plus de vingt secondes, il est en attente ;
- `Z` (*Zombie*) : indique que le processus s'est exécuté, qu'il n'a plus de raisons de vivre, mais vient quand même hanter le système (ni vivant , ni vraiment mort → zombie).

Il est possible de regrouper les processus sous forme d'arborescence. Plusieurs processus sont des « enfants » d'autres processus, cela vous permet de savoir qui est à l'origine de quel processus. Pour cela, on utilise la commande suivante :

Terminal
```
yann@geonosis:~$ ps -ejH
  PID  PGID   SID TTY          TIME   CMD
[...]
 2104  2104  2104 ?        00:00:00   lightdm
 2127  2127  2127 tty7     1-01:12:17  Xorg
 2466  2104  2104 ?        00:00:00   lightdm
 2866  2866  2866 ?        00:00:01   init
 2959  2959  2959 ?        00:00:55   dbus-daemon
 2985  2985  2985 ?        00:04:19   ibus-daemon
 3023  2985  2985 ?        00:00:00   ibus-dconf
```

La commande `pstree` permet aussi de visualiser l'arborescence des processus et d'une manière plus graphique (figure 2, page suivante).

La commande `ps` est pratique, mais reste limitée, car elle n'affiche pas les informations en temps réel.

Pour avoir une visualisation dynamique des processus en exécution, nous allons utiliser la commande `top`. Elle affiche la liste des processus, avec le PID, le nom du processus, le pourcentage d'utilisation du processeur (CPU), et le temps d'utilisation cumulé du processeur, la consommation mémoire, etc. (figure 3).

L'en-tête indique le nombre total de processus, ainsi que la quantité dans chaque état `R`, `S`, `T` et `Z`. On trouve aussi la moyenne de la charge du système (*load average*) : la première charge est la moyenne

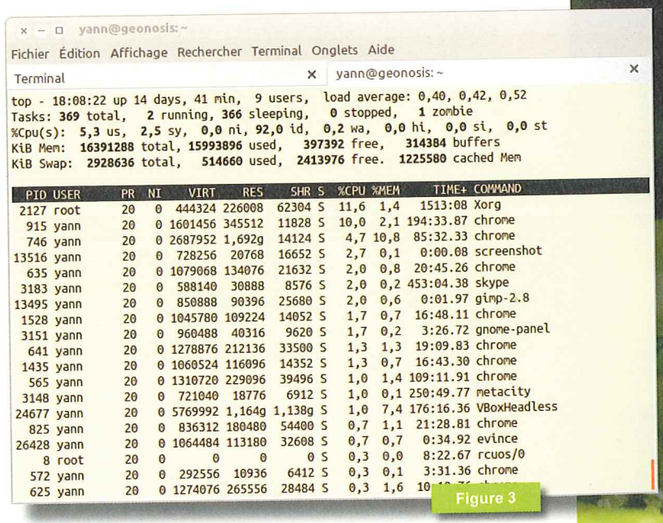


Figure 3

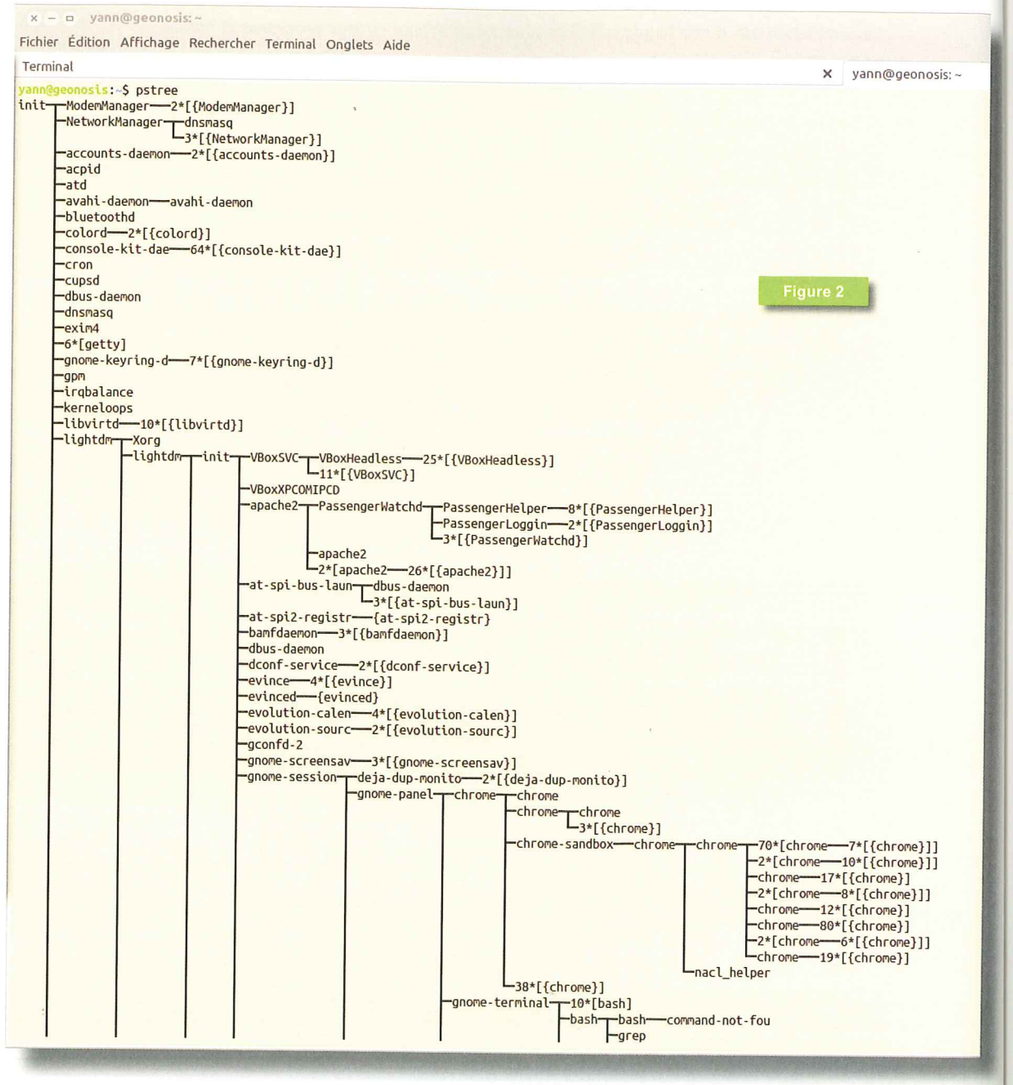


Figure 2

des cinq dernières minutes, la deuxième est celle des dix dernières minutes et la troisième celle des quinze dernières minutes. L'utilisation du processeur (*CPU usage*) est le pourcentage d'utilisation du processeur par l'utilisateur, le système et le pourcentage qu'il passe au repos (*idle*).

Mais quelle est l'utilité de toutes ces informations ? Eh bien, cela vous permet d'avoir une vue globale de l'utilisation de votre système. Ensuite, dans le cas d'une application qui viendrait à ne plus répondre, on va pouvoir utiliser son PID afin de la terminer, et ainsi libérer des ressources pour d'autres programmes sans devoir redémarrer le système.

Pour communiquer avec un processus, on utilise des signaux. Ce sont de petits messages que le processus va exécuter. Il existe de nombreux signaux, mais ceux que nous utiliserons le plus couramment sont :

- ⇨ **TERM** (15), utilisé en général pour quitter des applications (équivalent à [Ctrl]+[C]) ;
- ⇨ **KILL** (9), signal qui ne peut être ignoré par le processus. Il lui ordonne de s'arrêter immédiatement, quelles qu'en soient les conséquences ;
- ⇨ **HUP** (1) est surtout utilisé pour redémarrer un processus lancé par le système ;
- ⇨ **SIGSTOP** (19) : on suspend l'exécution du processus (équivalent à [Ctrl]+[Z]).

La liste complète des signaux est donnée par la commande **kill -l**.

Pour transmettre un signal à un processus, on peut utiliser son PID ou son nom. Dans le premier cas, on utilise la commande **kill**, dans le second **killall**.

La commande **kill** s'utilise comme suit :

Fichier

```
kill -nom_du_signal PID
kill -numero_du_signal PID
```

Sans option, elle envoie le signal **TERM** (15) au processus. Pour l'utilisation, on commence par récupérer le PID du processus à terminer et on l'utilise avec la commande **kill** :

Terminal

```
yann@geonosis:~$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
yann     14181  7.1  0.1  895948 30356 pts/11   Sl   18:29   0:00 geany
[...]
yann@geonosis:~$ kill 14181
```

Si l'application ne veut pas se fermer, il faut être un peu plus agressif et utiliser le signal **KILL** (9) :

Terminal

```
yann@geonosis:~$ kill -9 14181
```

ou encore

Terminal

```
yann@geonosis:~$ kill -KILL 14181
```

Dans le cas ou l'on connaît le nom exact du processus, on utilise **killall** de le même manière :

Terminal

```
yann@geonosis:~$ killall geany
```

ou encore

Terminal

```
yann@geonosis:~$ killall -KILL geany
```

Par défaut, une commande s'exécute en avant-plan (en anglais *foreground*). Lorsque l'utilisateur saisit **geany**, le shell crée un processus enfant pour exécuter l'éditeur graphique **geany** et attend qu'il se termine. Il est alors impossible de travailler avec le

shell. Une fois l'application enfant terminée, elle rend la main au shell. Les processus parent et enfant s'exécutent de manière séquentielle et une seule commande est exécutée à la fois.

Sous Unix, une commande peut aussi s'exécuter en arrière-plan (en anglais *background*). Pour cela, on utilise le caractère « et commercial » ou esperluette (« & ») à la fin de la commande. Lorsque l'utilisateur saisit `geany &`, le shell crée un processus enfant, mais n'attend pas qu'il se termine pour reprendre la main et afficher le prompt. Les deux processus, parent et enfant, s'exécutent alors « simultanément ». Cela permet de lancer plusieurs applications à partir d'un même shell.

Terminal

```
yann@geonosis:~$ geany&
[1] 15360
yann@geonosis:~$ gedit&
[2] 15427
yann@geonosis:~$ jobs
[1]-  En cours d'exécution   geany &
[2]+  En cours d'exécution   gedit &
yann@geonosis:~$
[1]-  Fini                   geany
[2]+  Fini                   gedit
```

Les `[1]` et `[2]` correspondent à des numéros de job, attribués par le shell, les autres nombres (15360 et 15427) correspondent aux PID. Le shell Bash permet en effet de suspendre, reprendre ou passer en arrière-plan un processus, ce qui nécessite une identification supplémentaire en sus de l'identifiant attribué par le système. Ainsi, si vous lancez d'autres jobs en arrière-plan alors que le premier est toujours en cours d'exécution, le shell leur attribuera les numéros 2, 3, 4, etc.

Lorsque les processus d'arrière-plan se terminent, vous obtenez des informations en retour. Le signe `+` permet d'identifier le dernier processus à avoir été lancé dans le terminal ; ce sera le job considéré par défaut lorsque vous effectuerez les manipulations suivantes.

Il est possible de ramener un processus au premier plan, on utilise la commande `fg` (pour *foreground*). Si un seul processus s'exécute en arrière-plan, `fg` peut être utilisé sans argument ; en revanche, si plusieurs sont en cours d'exécution, le shell sélectionnera le dernier processus lancé en arrière-plan, à moins de préciser en argument le numéro du job souhaité précédé du caractère `%` :

Terminal

```
yann@geonosis:~$ gedit&
[1] 15607
yann@geonosis:~$ geany&
[2] 15617
yann@geonosis:~$ jobs
[1]-  En cours d'exécution   gedit &
[2]+  En cours d'exécution   geany &
yann@geonosis:~$ fg %2
geany
```

Le job reprend alors le contrôle du terminal, autrement dit son comportement est tel que si vous n'aviez pas utilisé de `&`.

Inversement, la commande `bg` (pour *background*), permet d'envoyer un job en arrière-plan :

Terminal

```
yann@geonosis:~$ gedit&
[1] 15778
yann@geonosis:~$ geany&
[2] 15787
yann@geonosis:~$ fg %2
geany
^Z
[2]+  Arrêté                 geany
yann@geonosis:~$ bg
[2]+ geany &
yann@geonosis:~$
```

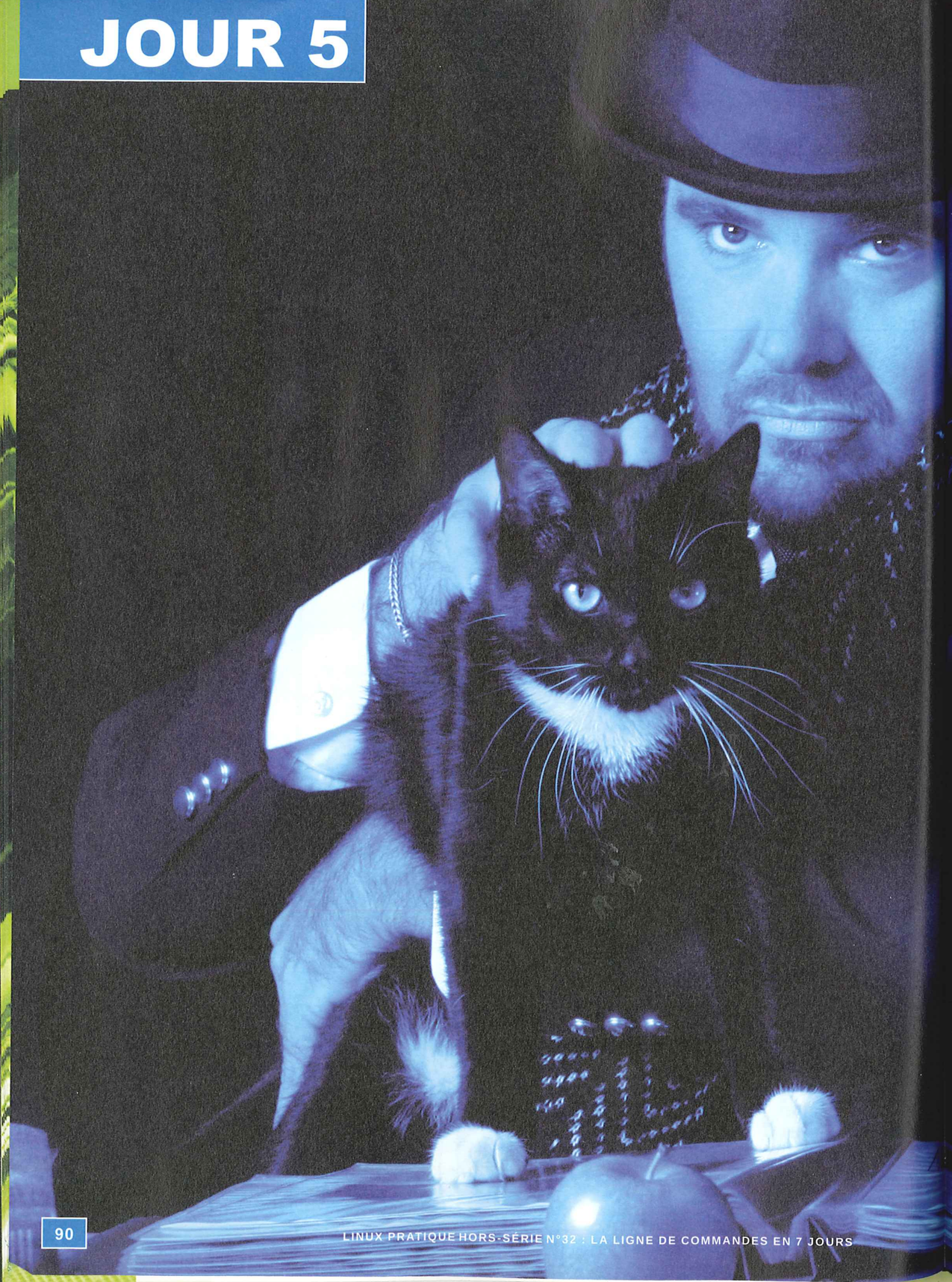
Ici nous reprenons donc la main et le processus `geany` continue de s'exécuter en arrière-plan. Notez bien l'utilisation de la combinaison de touches [Ctrl]+[Z] qui permet de suspendre un processus (pour le reprendre ultérieurement).

À savoir

Pour visualiser les numéros de chaque job en cours d'exécution, on utilise la commande `jobs`, éventuellement accompagnée de l'option `-l` qui affiche en plus le PID.

Récapitulatif

- Une expression rationnelle, ou expression régulière est une chaîne de caractères ou motif qui permet de décrire un ensemble de chaînes de caractères possibles selon une syntaxe précise.
- `aptitude` est le gestionnaire de paquets recommandé pour la gestion quotidienne des paquets en mode console.
- Un service est un programme réalisant des tâches de fond du système.
- Les commandes `service` et `initctl` permettent la gestion des services.
- Un processus ou job est un programme en cours d'exécution. Il a besoin de ressources matérielles pour fonctionner (mémoire, processeur, périphériques, etc.).
- La commande `ps` vous permet d'obtenir la liste des processus en exécution au moment où vous lancez la commande.
- La commande `top` permet d'avoir une visualisation dynamique des processus en exécution.
- Pour transmettre un signal à un processus, on peut utiliser son PID ou son nom. Dans le premier cas, on utilise la commande `kill`, dans le second `killall`.
- Sous Unix, une commande peut aussi s'exécuter en arrière-plan (en anglais *background*) en utilisant le caractère « et commercial » ou esperluette (« & ») à la fin de la commande.



JOUR 5

CONFIGURER ET UTILISER VOS CONNEXIONS RÉSEAU

Sébastien Chazallet

Quelle serait l'utilité d'une machine sans l'accès au réseau ? De sa configuration à son monitoring, présentation de quelques outils indispensables.

JOUR 5

1 ADRESSAGE LOGIQUE

1.1 Notion d'interface

Pour pouvoir utiliser un réseau, une machine nécessite une interface de connexion qui peut être un câble RJ45, un point d'accès WiFi, un modem téléphonique ou encore un modem ADSL. Une machine peut éventuellement appartenir à plusieurs réseaux si elle a plusieurs interfaces.

Chaque interface porte un nom technique :

- *lo* qui est la boucle locale ;
- *eth0* pour la première interface Ethernet (câble réseau RJ45) ;
- *eth1* si l'on a une deuxième carte réseau Ethernet dans la machine ;
- *wlan0* si l'on a une connexion Wi-Fi ;
- *vboxnet* si l'on a des machines virtuelles avec lesquelles on peut échanger via un réseau virtuel
- ...

Elle est également identifiée physiquement de manière unique par une adresse MAC (couche de liaison) et logiquement par une adresse IP (couche réseau) qui est unique sur le réseau. Si l'adresse MAC est automatiquement donnée par l'interface réseau elle-même, l'adresse logique doit être spécifiée et cela se passe dans le fichier **/etc/network/interfaces**.

Pour visualiser votre configuration, vous pouvez utiliser la commande **ifconfig** (ou **/sbin/ifconfig**) :

Terminal
```
$ /sbin/ifconfig -a
```

L'argument **-a** permet de voir toutes les interfaces, y compris celles n'étant pas activées (câble réseau retiré ou Wi-Fi désactivé).

1.2 Configuration Ethernet statique

Voici comment configurer statiquement un réseau, en IPv4 :

Fichier
```
auto eth0
iface eth0 inet static
  address 192.168.0.42
  netmask 255.255.255.0
  broadcast 192.168.0.255
  network 192.168.0.0
  gateway 192.168.0.1
```

La première ligne permet de déclarer que le réseau doit être monté au démarrage de la machine. La seconde ligne permet de décrire sa configuration, en précisant :

- l'adresse de la machine sur le réseau ;
- le masque (permet de spécifier l'étendue du réseau) ;
- l'adresse de broadcast (optionnelle, permet d'envoyer une donnée à toutes les machines du réseau, sans traverser les routeurs) ;
- l'adresse du réseau (optionnelle, partie de l'adresse IP commune à toutes les machines du réseau) ;
- l'adresse de la passerelle (optionnelle, adresse du routeur du réseau).

Il est aussi possible de réaliser sa configuration par l'utilisation de cette commande :

Terminal
```
$ sudo ifconfig eth0 netmask 255.255.255.0 192.168.0.1
```

Pour donner une adresse IPv6, il faudra ajouter ces lignes :

Fichier
```
iface eth0 inet6 static
  pre-up modprobe ipv6
  address fe80::12bf:1000:45:1010:7f:ad02
  netmask 64
  gateway fe80::12bf:1000:45:1010:7f:ad02
```

Ou, via la ligne de commandes :

Terminal
```
$ sudo ifconfig eth0 inet6 add fe80::12bf:1000:45:1010:7f:ad02
```

1.3 Configuration Ethernet dynamique

Le principal souci avec l'utilisation d'adresses statiques réside dans le fait que rien n'empêche d'avoir deux machines avec la même adresse sur un réseau. De plus, on peut aussi avoir plus de machines que d'adresses disponibles, mais qu'elles ne soient pas toutes allumées en même temps.

Attribuer définitivement une adresse à une machine n'est donc pas la solution idéale. C'est un problème que connaissent bien les fournisseurs d'accès internet, par exemple, qui ont parfois plus de clients que d'adresses IP en stock.

La solution consiste donc à permettre d'attribuer dynamiquement une adresse à une machine au moment où celle-ci entre sur un réseau. C'est le rôle des serveurs DHCP (*Dynamic Host Configuration Protocol*).

Si au moins un serveur DHCP existe sur un réseau, il est possible de configurer son interface réseau ainsi :

Fichier
```
auto eth0
iface eth0 inet dhcp
  hostname myHostname
```

Il est possible de préciser le nom d'hôte que l'on souhaite avoir sur le réseau, le reste est automatique.

La configuration en est donc largement simplifiée, tous les paramètres utiles du réseau étant centralisés par le serveur DHCP. Elle peut également se faire via la ligne de commandes, en spécifiant IPv4 ou IPv6 :

Terminal
```
$ sudo ifconfig eth0 dhcp start
$ sudo ifconfig eth0 inet6 dhcp start
```

1.4 Valider sa configuration

Une fois que l'on a configuré une interface, il faut redémarrer le service réseau pour la prendre en compte :

Terminal
```
$ sudo /etc/init.d/networking restart
```

Pour vérifier que cette configuration a été réalisée convenablement, on peut utiliser la commande suivante :

Terminal
```
$ /sbin/ifconfig eth0
$ ip address show eth0
```

Si l'on ne voit pas d'adresse IPv4 ou IPv6, c'est que l'attribution de l'adresse n'a pas fonctionné. On n'a donc pas accès au réseau.

ATTRIBUTION D'UNE ADRESSE IP PAR LE SERVEUR DHCP

Lorsqu'elle se connecte sur le réseau, la machine cliente envoie en *broadcast* (donc à tout le monde sur le réseau) un message du type *DCHP DISCOVER* en indiquant son adresse MAC.

- Tout serveur DCHP à l'écoute du réseau, s'il lui reste des adresses disponibles, renvoie directement à la machine client (identifiée par son adresse MAC) un message de type *DHCP OFFER*, lequel contient les données suivantes :
  - sa propre adresse IP ;
  - l'adresse IP qu'il propose à la machine cliente ;
  - le masque qu'il propose à la machine cliente .
- La machine cliente prend la première offre qu'elle reçoit et diffuse sur le réseau un message du type *DHCP REQUEST* qui contient sa propre adresse IP et celle du serveur DHCP dont il vient d'accepter l'offre :
  - le serveur DHCP à l'origine de l'offre va assigner l'adresse IP à la machine cliente ;
  - les autres serveurs DHCP sauront que leur offre n'a pas été retenue.
- Le serveur DHCP dont l'offre a été retenue diffuse un message de type *DHCP ACK* (*acknowledgement*) pour valider l'adresse à la machine cliente et lui envoie des données complémentaires, comme l'adresse de la passerelle ou celles des serveurs DNS.

1.5 Configuration modem

La plupart des modems modernes actuels font également routeur et serveur DHCP. Il suffit donc de configurer proprement le réseau en adressage dynamique pour avoir accès à internet.

Par contre, si l'on dispose d'un modem ADSL connecté en USB sur sa machine, il convient de le configurer soi-même. La majorité des modèles fonctionnent en utilisant le protocole PPPOE (*Point-to-Point Protocol Over Ethernet* ou protocole point à point sur Ethernet, utilisant la couche de liaison). Pour configurer un tel modem, il suffit d'utiliser l'utilitaire **pppoeconf**, qu'il convient d'installer ainsi :

Terminal
```
$ sudo aptitude install pppoeconf
```

La configuration sera enregistrée dans le fichier **/etc/ppp/peers/dsl-provider** et les informations d'authentification le seront dans les fichiers **/etc/ppp/pap-secrets** et **/etc/ppp/chap-secrets**.

La connexion pourra être démarrée et stoppée par l'utilisation des commandes suivantes :

Terminal
```
$ sudo pon dsl-provider
$ sudo poff dsl-provider
```


Il est à noter que certains modems n'utilisent pas le protocole PPPOE. Il faudra alors regarder leurs spécificités et possiblement installer un paquet qui leur est dédié, comme c'est le cas du Speedtouch d'Alcatel qui nécessite d'installer le paquet éponyme (il utilise PPPOA, protocole point à point sur le protocole « mode de transfert asynchrone »).

Enfin, il existe aussi des modems téléphoniques utilisant le protocole PPP (utilisant également la couche de liaison) et, comme leur nom l'indique, utilisant le réseau téléphonique). Pour ceux-ci, il faut utiliser l'utilitaire **pppconfig**, lequel s'installe ainsi :

Terminal
```
$ sudo aptitude install pppconfig
```

La connexion pourra être démarrée ou stoppée à l'aide des commandes **pon** et **poff**.

2 ROUTAGE

2.1 Réseau local

Lorsqu'une machine fait partie d'un réseau local, elle peut atteindre toutes les machines de ce réseau local à l'aide de la couche de liaison. Il lui suffit de connaître l'adresse MAC de la machine à atteindre.

En IPv4, chaque machine dispose d'une table ARP, qui va lier chaque adresse IP du réseau local à une adresse MAC. La commande suivante (paquet **arp-scan** pour debian) permet de visualiser cette table :

Terminal
```
$ arp -a
```

Comme ces adresses IP peuvent changer, une machine peut envoyer une requête de type ARP à l'aide de la commande suivante :

Terminal
```
$ arp 192.168.1.47
Address                  HWtype  HWaddress           Flags Mask            Iface
PC-2.local               ether   **:**:**:**:**:**   C                     eth0
```

Il est possible d'intervenir directement sur la table ARP en y ajoutant une entrée :

Terminal
```
$ arp 192.168.1.49 **:**:**:**:**:**
```

EN IPv6, il existe NDP (*Neighbor Discovery Protocol*) qui est une amélioration du protocole ARP auxquels ont été ajoutés d'autres types de services.

2.2 Table de routage

Lorsque l'on souhaite atteindre une machine qui n'est pas sur le réseau local, par exemple en allant sur un site internet à l'aide de son navigateur, on se doit de transiter de réseau en réseau jusqu'à atteindre la machine destinataire. Pour ce faire, on passe par des routeurs.

Pour visualiser la table de routage de sa machine, il suffit d'utiliser la commande suivante :

Terminal
```
$ netstat -nr
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic   MSS Fenêtre irtt Iface
0.0.0.0         192.168.1.254   0.0.0.0         UG        0 0          0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

Dans le résultat que l'on voit ici, on peut déduire que si l'adresse est de la forme *192.168.1.XXX*, on attaquera directement la machine destinataire, dans tous les autres cas, on attaquera la *passerelle 192.168.1.254*.

2.3 Routeur

Un routeur est simplement une machine connectée à au moins deux réseaux différents et étant déclarée comme passerelle sur au moins un de ces réseaux. C'est le cas par exemple d'un modem ADSL moderne, tel que la FreeBox.

Ainsi, lorsqu'il reçoit un paquet sur une de ces interfaces réseaux, il peut le faire passer sur un autre, en suivant les règles de sa propre table de routage.

Ce routeur a lui aussi une passerelle et en répétant la même opération, le paquet réseau va circuler de proche en proche sur un certain nombre de réseaux jusqu'à arriver à la machine destinataire.

Ainsi, pour atteindre une machine ayant une certaine adresse IP, on va utiliser un certain nombre de serveurs. Les commandes **`ping`** ou **`ping6`** permettent de tester l'accessibilité d'une machine via le réseau (respectivement IPv4 ou IPv6) via le protocole ICMP ou ICMPv6 :

Terminal

```
$ ping inspyration.fr
PING inspyration.fr (213.246.53.26) 56(84) bytes of data.
64 bytes from 40965hd53026.ikexpress.com (213.246.53.26): icmp_
req=1 ttl=51 time=45.1 ms
[...]
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 40.473/41.982/45.163/1.743 ms
```

La commande permet d'accéder à plusieurs informations :

- l'IP liée à l'adresse (via DNS) ;
- le nom de domaine principal (via reverse DNS, ici **`40965hd53026.ikexpress.com`**) ;
- le numéro de la requête ICMP (**`icmp_req`**) ;
- le nombre de routeurs intermédiaires (**`ttl`**) ;
- le temps aller-retour (ici **`45,1 ms`**) ;
- le nombre de paquets envoyés, reçus et perdus ;
- le temps minimal, moyen, maximal et la déviation standard.

Ces données peuvent permettre de détecter un éventuel problème soit sur le réseau, soit avec la machine distante (temps trop long, déviation standard trop élevée). Il faut cependant savoir que certains pare-feu bloquent cette commande.

Les commandes **`traceroute`** ou **`traceroute6`** permettent de suivre plus précisément le chemin aller que les données suivent en allant de la machine émettrice à la machine réceptrice et peut par exemple permettre de diagnostiquer un problème de congestion sur l'un des routeurs.

3 NOMS DE DOMAINES

3.1 Nom canonique

Une *adresse IP* n'identifie pas une machine. En effet, comme on l'a déjà évoqué, d'une part une machine peut avoir plusieurs *interfaces réseau* et donc plusieurs *adresses IP* et d'autre part rien n'empêche l'interface réseau d'une machine de changer d'*adresse IP* dans le temps.

Pour identifier une machine sur un *réseau local*, on utilise un *nom canonique* (celui que l'on verra sur un partage réseau). Il est stocké dans le fichier **`/etc/hostname`** (qui ne contient rien d'autre).

Il est possible de retrouver cette information ainsi :

Terminal

```
$ hostname
```

3.2 Nom complet

Pour identifier une machine sur un réseau plus large qu'un simple réseau local, il faut utiliser un nom de domaine complet. Ce nom complet est celui du nom canonique de la machine accolé à celui du réseau, par exemple **`ma_machine.mon_domaine.com`**.

Le nom complet de la machine se trouve dans le fichier **`/etc/hosts`**. Ce dernier se présente ainsi :

Fichier

```
127.0.0.1       ma_machine.mon_domaine.com localhost
ma_machine autres_vhosts
```

3.3 DNS

Lorsque l'on cherche à joindre une machine distante sur un large réseau, on n'utilise pas son adresse IP, mais son nom de domaine. Cependant, pour lui transmettre des données, on ne peut le faire qu'en connaissant cette adresse IP. On utilise donc les services d'un serveur DNS.

Un serveur DNS permet d'enregistrer l'adresse IP correspondant à un nom de domaine (ainsi que d'autres informations). Lorsqu'une nouvelle entrée est enregistrée dans l'un d'entre eux, elle est propagée vers les autres, de manière à rendre l'information accessible à tous.

Pour utiliser ces services, il faut entrer l'adresse IP de l'un d'entre eux (sauf si l'on est en DCHP, qui fait cela pour nous). Pour ceci, il faut aller dans le fichier **`/etc/resolv.conf`** et définir les adresses IP des serveurs DNS de cette manière :

Fichier

```
nameserver xxx.xxx.xxx.xxx
nameserver yyy.yyy.yyy.yyy
```

À partir de là, on peut utiliser la commande suivante pour obtenir une adresse IP à partir d'un nom de domaine (potentiellement avec les options **`-4`** pour IPv4 ou **`-6`** pour IPv6) :

Terminal

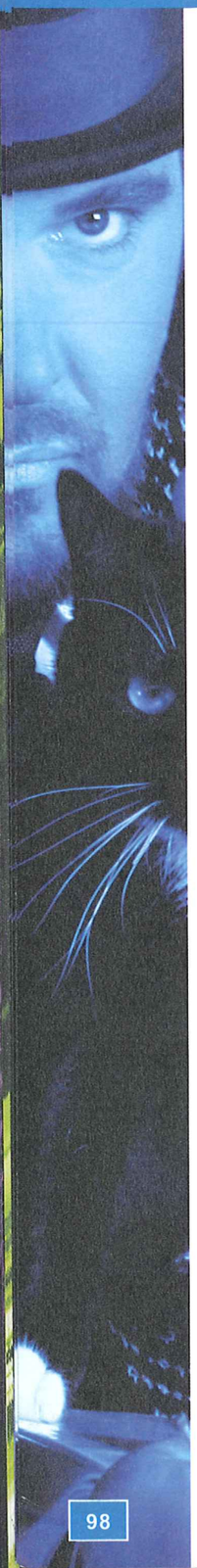
```
$ host www.inspyration.fr
www.inspyration.fr is an alias for inspyration.fr.
inspyration.fr has address 213.246.53.26
```

3.4 Alias

Comme on l'a vu précédemment (cf 3.2), le fichier **`/etc/hosts`** peut contenir plusieurs noms. Le premier est le nom de domaine, les autres sont des alias. Ceci permet à une machine d'être vue sous plusieurs noms

À savoir

Le système de nom de domaine se lit ainsi : dans `ma_machine.mon_domaine.com`, on distingue un nom de domaine de premier niveau qui est `com`, un nom de domaine qui est `nom_domaine` et un nom de sous-domaine qui est `ma_machine`, lequel en l'occurrence correspond à une machine spécifique (ou hôte), mais qui pourrait correspondre à un hôte virtuel.



de domaines différents (et les serveurs utilisent cette spécificité, par exemple pour héberger différents sites web sur le même serveur physique).

Lorsque l'on passe en paramètre de la commande **host** une adresse IP, on obtient le reverse DNS, c'est-à-dire le nom de domaine principal lié à cette adresse IP :

Terminal

```
$ host 213.246.53.26
26.53.246.213.in-addr.arpa domain name pointer 40965hd53026.ikexpress.com.
```

Pour cette machine, **40965hd53026** correspond à son nom canonique. Elle dispose cependant de plusieurs dizaines de noms de domaines différents.

4 PROBLÉMATIQUES DE SÉCURITÉ

4.1 État du réseau

Qui dit réseau, dit ouverture vers le monde extérieur et donc besoin de sécuriser ce qui entre et ce qui sort.

Avant de sécuriser le réseau, il faut savoir ce qu'il s'y passe. La commande suivante permet de récupérer des statistiques sur les paquets réseaux liés à la machine :

Terminal

```
$ netstat -s
```

On peut ainsi voir s'il n'y a pas trop de perte de paquets réseau (ce qui peut indiquer une mauvaise configuration), par exemple.

La commande suivante permet de savoir quelles sont les connexions actives :

Terminal

```
$ netstat -a | less
```

Une ligne telles que celles-ci indique que l'on a sur la machine un serveur qui est à l'écoute du réseau :

Terminal

```
tcp        0      0 localhost:mysql         *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
tcp        0      0 localhost:postgresql    *:*                     LISTEN
tcp        0      0 localhost:8000          *:*                     LISTEN
```

Dans une seconde partie, la commande liste aussi les sockets actives, ce qui est hors cadre.

4.2 Tester le réseau

Il existe un outil formidable nommé **nmap**. Il permet de scanner l'ensemble des ports sur une machine distante et réaliser certains tests lui permettant de récupérer le maximum d'informations (version du système d'exploitation, liste des services actifs et leurs versions...).

Couplé à une base de connaissance des failles connues, cela permet de mettre en évidence d'éventuels problèmes de sécurité à résoudre (lesquels peuvent être résolus par mise à jour du système ou d'un logiciel en particulier ou encore par le renforcement de fichiers de configuration).

Il est à noter qu'il existe au moins deux interfaces graphiques pour cet outil, nommées nmapsi4 et ZenMap.

Terminal

```
$ nmap -A machine.nom_domaine.com
```

4.3 Pare-feu

Sécuriser correctement un réseau passe par l'utilisation d'un pare-feu. Les logiciels **iptables** et **ip6tables** permettent de décrire des règles permettant de filtrer le trafic réseau sur une machine. Si cette machine est un routeur, cela permet de contrôler l'échange des paquets entre les différents réseaux.

Les commandes sont relativement simples, mais l'utilisation du logiciel requiert une parfaite connaissance de la manière dont le réseau fonctionne, des différents protocoles et des types de règles. Par conséquent, son utilisation est réservée à des connaisseurs.

Il existe également des outils tels que **apf-firewall** qui valent le détour et peuvent se révéler plus simples à utiliser en première approche.

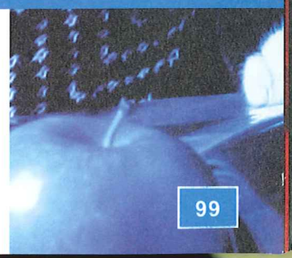
Récapitulatif

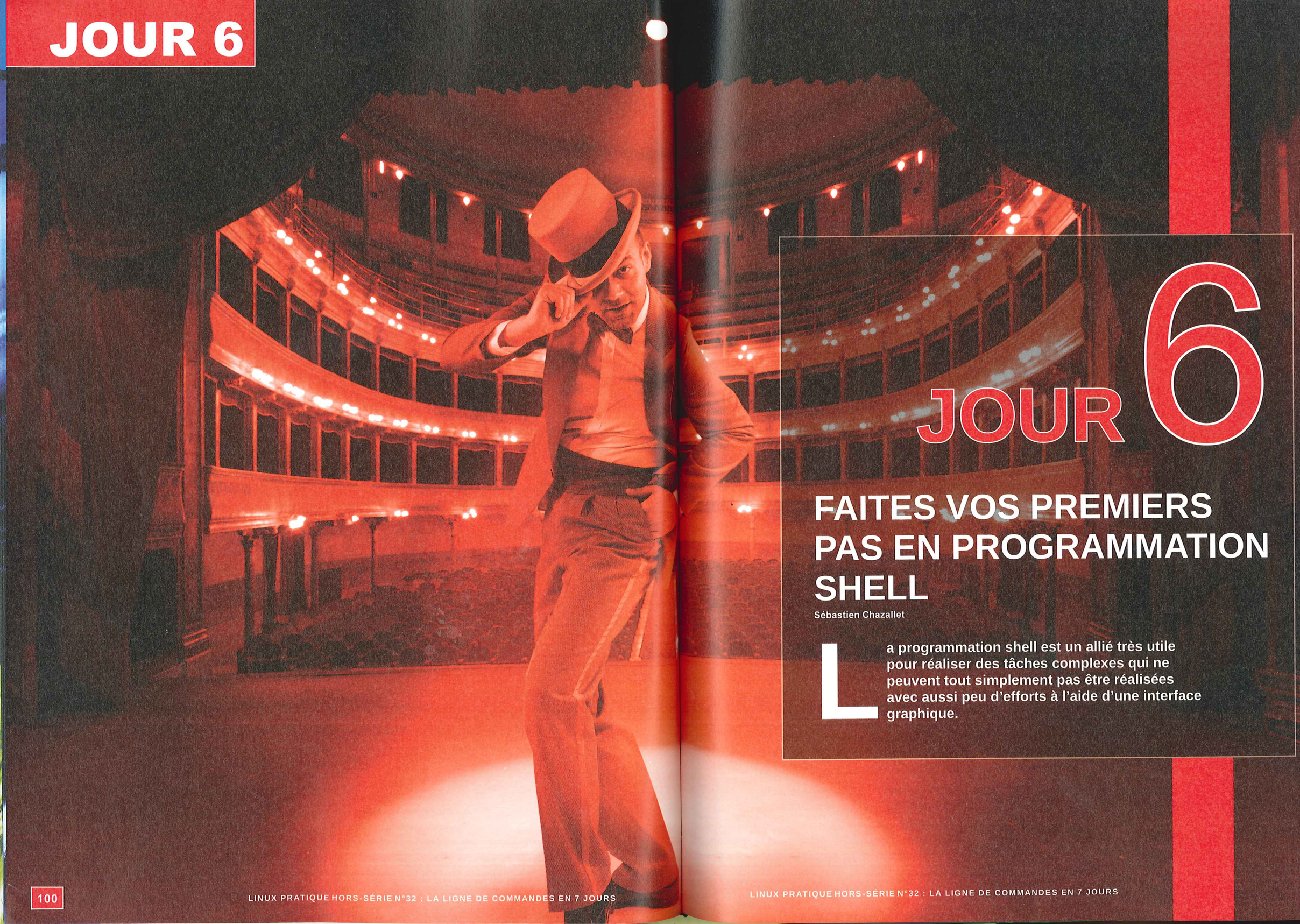
- On peut configurer son réseau par l'utilisation de la commande **ifconfig** ou par la modification du fichier **/etc/network/interfaces**.
- On peut visualiser la liste des machines du réseau local en utilisant la commande **arp**.
- On peut visualiser sa table de routage en utilisant **netstat -nr**.
- La commande **ping** offre beaucoup plus d'informations que simplement le temps de réponse du réseau.
- La commande **traceroute** permet de connaître le chemin emprunté par les paquets pour aller jusqu'à une destination spécifique.
- La commande **host** permet de connaître l'IP associé à un nom de domaine ou le nom de domaine principal d'une IP.
- Le logiciel **nmap** permet de faire un bilan de sécurité d'une machine vis-à-vis du réseau.
- Les logiciels **iptables** et **ip6tables** permettent de sécuriser le réseau.

À savoir

Dans le répertoire /etc/network/ se trouvent 4 sous-répertoires nommés ifdown.d, if-up.d, if-post-down.d, if-pre-up.d. Comme leur nom le laisse supposer, ces répertoires sont destinés à contenir des scripts, lesquels vont être lancés respectivement à l'arrêt de la connexion réseau, lors de son activation, après son arrêt ou avant son activation.

Placer des scripts dans ce répertoire est un moyen d'exécuter des actions à chaque fois que l'on se connecte ou déconnecte d'un réseau.





JOUR 6

FAITES VOS PREMIERS PAS EN PROGRAMMATION SHELL

Sébastien Chazallet

La programmation shell est un allié très utile pour réaliser des tâches complexes qui ne peuvent tout simplement pas être réalisées avec aussi peu d'efforts à l'aide d'une interface graphique.

JOUR 6

Le shell a deux modes de fonctionnement. Le premier mode est celui que vous avez utilisé jusqu'à présent en suivant ce guide, c'est-à-dire le mode interactif. Vous ouvrez un terminal et à chaque commande que vous tapez, une interprétation est réalisée et la commande est exécutée. Vous pouvez alors constater son résultat.

Dans le mode batch, le shell va interpréter un fichier textuel qui va contenir les commandes à exécuter. Dans le principe, ce n'est pas plus complexe que ceci.

ÉCRIRE UN SCRIPT

1.1 Fichier script

Pour écrire un script, il suffit de créer un fichier textuel, de préférence avec une extension **.sh** :

Terminal
```
$ touch exemple.sh
```

Puis de le rendre exécutable :

Terminal
```
$ chmod +x exemple.sh
```

On peut ensuite l'éditer à l'aide de son éditeur préféré. La première chose à faire est d'indiquer l'interpréteur à choisir. Pour savoir exactement ce qu'il convient de mettre, on peut utiliser ceci :

Terminal
```
$ which bash
/bin/bash
```

On va donc prendre ce résultat et le faire précéder par **#!**. Voici donc un script de type hello world :

Fichier
```
#!/bin/bash
echo "Hello World!"
```

Voici comment l'exécuter :

Terminal
```
$ ./exemple.sh
Hello World!
```

Si on le souhaite, on peut passer en mode débogage (ce qui reste léger, mais est toujours bon à prendre) :

Terminal
```
$ bash -x exemple.sh
+ echo 'Hello World!'
Hello World!
```

Toutes les lignes commençant par **+** sont les informations de débogage. Elles permettent de visualiser les commandes exécutées. Si l'on rajoute des instructions **echo** pour afficher l'état de variables, par exemple, cela peut aider à visualiser exactement l'état du programme à chaque étape de son exécution.

Enfin, pour exécuter son programme sans le **./** devant, il faut que ce programme soit dans un des répertoires du PATH ou que vous en rajoutiez un nouveau. À noter que rajouter **.** dans le PATH n'est vraiment pas une bonne idée.

1.2 Gérer le retour d'une commande

Un script peut utiliser n'importe quelle commande accessible dans un terminal interactif.

Toutes les commandes renvoient un résultat sous la forme d'un entier positif ou nul. Lorsqu'il est nul, c'est que la commande s'est terminée proprement. Dans le cas contraire, le résultat indique un numéro d'erreur dont la logique est propre à la commande.

Si dans une console interactive on voit immédiatement le résultat et on peut s'adapter, il faut également faire en sorte que notre script puisse savoir si la commande a échoué ou non, afin d'adapter son comportement. Par exemple, si une commande échoue, il faut qu'il puisse afficher un message et sortir afin d'éviter d'exécuter les commandes suivantes, ce qui pourrait poser des soucis :

Fichier
```
if [ $? -eq 0 ] ; then
        echo "succès, on continue"
else
        echo "Erreur, on s'arrête"
        exit 42
fi
```

La variable **$?** permet de récupérer la valeur de retour de la dernière commande précédemment exécutée.

Dans cet exemple, j'ai choisi arbitrairement de quitter mon programme avec une erreur **42** (ce qui compte est d'avoir un numéro d'erreur différent pour chaque façon de quitter). Si un autre script utilise le mien, il récupérera cette valeur non nulle.

Cet exemple montre aussi la manière dont on construit un bloc conditionnel. La syntaxe utilise les mots clés **if then elif else fi**, l'utilisation des crochets et l'opérateur d'égalité pour les nombres qui est **-eq**.

Voici comment comprendre le bloc précédent : si le résultat de la commande précédente est nul, dire que tout va bien. Sinon, dire que l'on a une erreur et arrêter le programme. Et le bloc conditionnel s'arrête avec le mot clé **fi** (**if** lu de droite à gauche, un idiome souvent utilisé). L'exemple suivant utilisera **elif**.

1.3 Gérer les arguments

L'utilisateur de votre commande devra pouvoir passer des arguments et vous devrez vous assurer que vous en avez le bon nombre. Voici un exemple permettant de vous assurer que vous avez exactement deux arguments :

Fichier
```
if [ $# -lt 2 ] ; then
        echo "too few arguments"
        exit 1
elif [ $# -gt 2 ] ; then
        echo "too many arguments"
        exit 2
fi
```

La condition travaille sur le nombre d'arguments, contenu dans la variable **$#**. Le bloc conditionnel se comprend ainsi : si le nombre d'arguments est strictement plus petit que deux, alors informer l'utilisateur qu'il y a trop peu d'arguments et quitter. Sinon, si le nombre d'arguments est plus grand que deux, alors informer l'utilisateur qu'il y a trop d'arguments et quitter.

On aurait tout à fait pu enchaîner un **else** final, mais ce n'est pas obligatoire. Chaque bloc conditionnel doit avoir un bloc **if - then**, puis peut avoir 0 à n blocs **elif** et 0 ou 1 bloc **else**.

Concernant l'écriture des conditions, on a vu jusqu'ici les opérateurs d'égalité, d'infériorité stricte et de supériorité stricte pour les nombres.

À noter qu'il existe aussi :

- **-ge** pour plus grand ou égal ;
- **-le** pour plus petit ou égal ;
- et enfin, **-ne** pour différent.

On peut également utiliser des opérateurs logiques tels que **&&** pour ET et **||** pour OU :

Fichier
```
if [ $# -eq 2 ] && [ $1 = $2 ] ; then
        echo "the two files must be distinct"
        exit 3
fi
```

Comme on le voit, pour comparer des chaînes de caractères, on utilise simplement les opérateurs **=** ou **!=**.

Dans l'exemple ci-dessus, on vérifie que les deux noms de fichiers passés en paramètres sont bien distincts.

On voit la manière dont bash permet d'accéder aux arguments : ici, **$1** est le nom du premier argument, **$2** du second argument.

1.4 Gérer des fichiers

Des vérifications peuvent être réalisées sur les fichiers. On peut par exemple vérifier l'existence d'un chemin.

Fichier
```
if [ ! -e $MY_FILE ] ; then
        echo "$MY_FILE does not exists"
        exit 4
fi
```

Dans ce cas, si le chemin n'existe pas, on affiche un message et l'on quitte. On peut noter que la condition a été renversée par l'utilisation d'un point d'exclamation.

Voici comment vérifier que le chemin est un fichier :

Fichier
```
if [ -f $MY_FILE ]
```

Ou un répertoire :

Fichier
```
if [ -d $MY_FILE ]
```

Ou encore un lien symbolique :

Fichier
```
if [ -L $MY_FILE ]
```

On peut aussi vérifier les conditions d'accès (pour l'utilisateur courant, celui qui exécute le script) :

Fichier
```
if [ -r $MY_FILE ]
if [ -w $MY_FILE ]
if [ -x $MY_FILE ]
```

Il va sans dire que **r** signifie lecture, **w** pour écriture et **x** pour exécution. Enfin, il est possible de savoir si un fichier est plus récent (**nt** pour *newer than*) ou plus ancien (**ot** pour *older than*) :

Fichier
```
if [ $MY_FILE1 -nt $MY_FILE2 ]
if [ $MY_FILE1 -ot $MY_FILE2 ]
```

Tester ces informations au préalable permet d'éviter de lancer une commande qui est destinée à échouer à cause de problèmes d'accès. Cela permet aussi de prévoir tous les cas d'utilisation et d'améliorer ses messages d'erreurs, au minimum.

1.5 Saisies utilisateur

On a vu que l'on pouvait totalement gérer l'affichage via l'utilisation de **echo**. On peut aussi demander à l'utilisateur de saisir des informations :

Fichier
```
read -p 'Nom d'utilisateur' username
```

La commande **read** permet de demander une saisie à l'utilisateur, en utilisant le flux d'entrée standard. L'option **-p** permet d'afficher un message préalable, expliquant à l'utilisateur la nature attendue de la saisie. Sa réponse sera une chaîne qui sera stockée, dans cas de l'exemple, dans la variable **username**.

On peut vérifier que la saisie a bien eu lieu, voire boucler tant que ce n'est pas le cas :

Fichier
```
while [ -z $username ] ; do
    read -p "Nom d'utilisateur: " username
done
```

Le **-z** permet de vérifier si la chaîne est vide (tandis que le **-n** permet de vérifier qu'elle n'est pas vide).

La boucle **while** permet d'exécuter une portion de code tant qu'une condition est vraie. À noter qu'il existe également le mot clé **until** qui permet de boucler jusqu'à ce que la condition soit vraie (soit le contraire de **while**) :

Fichier
```
until [[ -n $password ]] ; do
    read -sp "Mot de passe: " password
    echo ""
done
```

Dans ce dernier exemple, on notera la présence de **-s** qui permet de ne pas afficher à l'écran la saisie de l'utilisateur.

2 LOGIQUE AVANCÉE

2.1 Boucler sur une donnée

Plutôt que de travailler sur des conditions, on peut boucler directement sur des valeurs :

Fichier
```
for script in 'exemple1.sh' 'exemple2.sh'
do
    chmox +x $script
    echo "le fichier $script a été rendu exécutable"
done
```

Au niveau de la syntaxe, le corps de la boucle est toujours délimité par les mots clés **do** et **done**. La variable de boucle est ici **script**. Elle va prendre les différentes valeurs placées après le mot clé **in**, au fur et à mesure des itérations.

Au lieu d'utiliser des données statiques, on peut utiliser le retour d'une commande :

Fichier
```
for script in 'ls *.sh 2>/dev/null'
do
    chmox +x $script
    echo "le fichier $script a été rendu exécutable"
done
```

Petit détail : on redirige la sortie d'erreur vers **/dev/null** pour s'en débarrasser. En effet, si l'on a un message, cela signifie qu'il n'y a en fait rien à faire. C'est un moyen plus court que de faire la commande au préalable et de vérifier le code de retour.

Voici un exemple permettant de boucler sur l'ensemble des arguments de la commande :

Fichier
```
for arg in $*
do
    echo $arg
done
```

2.2 Choix multiple

On a vu la structure qu'il est nécessaire de mettre en place pour traiter une condition, à l'aide des mots clés **if**, **then**, **elif**, **else** et **fi**. Lorsque l'on souhaite répéter un test sur de multiples valeurs, il est plus simple de passer par l'utilisation de **case** :

Fichier
```
read -n 1 p ÐQue décide-t-on [O/n/_] " choix
case $choix in
_)
    echo "Choix reporté"
    ;;
o|O )
    echo "Oui"
    ;;
n|N )
    echo "Non"
    ;;
*)
    echo "Choix non compris"
esac
```

Il y a plusieurs détails à aborder sur l'analyse de cet exemple. Tout d'abord, on peut noter l'utilisation du **-1** pour limiter le nombre de caractères à 1 lors de la saisie de l'utilisateur. Ensuite, la syntaxe utilise le mot clé **in** comme précédemment. Après ce mot clé, on trouve les différentes options.

Ces options, sont, dans notre exemple, le caractère souligné, les lettres **o** ou **O**, les lettres **n** ou **N** et enfin, toutes les autres possibilités, ce qui est représenté par le caractère étoile. Le caractère **|** sert à permettre à plusieurs choix de suivre le même chemin logique dans l'exécution du code. Le caractère parenthèse fermante permet de séparer ces groupes de choix du bloc de code qui sera exécuté si l'un de ces choix est fait par l'utilisateur.

Chacun de ces blocs se termine par deux points-virgules.

2.3 Fonctions

Le shell permet de déclarer des fonctions, à l'aide de la syntaxe suivante :

Fichier
```
function say_hello
{
    echo 'Hello World!'
}
```

Pour appeler une fonction, il suffit de procéder ainsi :

Fichier
```
say_hello
```

Si vous avez déjà pratiqué un tant soit peu un autre langage de programmation, vous devez vous demander où se trouvent les variables. En fait, il n'y a pas besoin de les déclarer dans la fonction, puisque par défaut, les variables d'une fonction vont être rendues disponibles dans **$1**, **$2**, etc., comme c'est le cas pour un programme. On dispose aussi du nombre d'arguments **$#**.

Fichier
```
function say_hello
{
    echo "Hello $1"
}
```

De plus, on est en bash, il n'y a donc pas de parenthèses ni de virgules pour gérer les arguments. Comme pour l'appel d'un programme, on va simplement les séparer par des espaces :

Fichier
```
say_hello World
```

Il faut également savoir qu'une variable déclarée à l'intérieur d'une fonction est globale, sauf mention contraire et qu'une fonction peut accéder à l'ensemble des variables déclarées dans le contexte global :

Fichier
```
ma_variable_globale="_"

function a
{
    ma_variable_globale="${ma_variable_globale}A"
}
function b
{
    ma_variable_globale="${ma_variable_globale}B"
}
```

L'exécution du code suivant :

Fichier
```
echo $ma_variable_globale
a
echo $ma_variable_globale
b
echo $ma_variable_globale
```

va provoquer le résultat suivant :

Terminal
```
$ ./exemple1.sh
_
_A
_AB
```

L'utilisation de **local** n'est pas celle d'un mot clé, mais d'une commande :

Fichier
```
    local ma_variable_locale="valeur"
```

Cette commande a des options. Il est par exemple possible d'utiliser **-i** pour préciser qu'il s'agit d'un entier :

Fichier
```
    local -i ma_variable_locale=42
```

Récapitulatif

- Vous pouvez utiliser dans un fichier script l'ensemble des commandes que vous utiliseriez dans un terminal interactif ; **$?** permet de récupérer les codes des retours.
- Le nombre de paramètres est donné par **$#**, les paramètres sont appelés individuellement par leur numéro **$1**, **$2**, ... sachant que **$0** est le nom de la commande et **$*** permet d'itérer dessus.
- La gestion des processus et des flux permet d'enchaîner des commandes et d'écrire dans des fichiers.
- Les conditions sont gérées à l'aide des mots clés **if then elif else fi** et les conditions s'appliquent de manière différente selon que les variables sont des chaînes de caractères, des entiers, ou que l'on souhaite faire des vérifications sur des chemins vers des fichiers.
- Les itérations peuvent être réalisées avec **while**, **until**, ou encore **for** et les conditions multiples avec **case**.
- La commande **read** permet de gérer la sortie utilisateur, son option **-s** permet de masquer la saisie de l'utilisateur (pour les mots de passe).
- La déclaration d'une fonction est minimaliste ; toutes les variables sont globales, elles s'utilisent comme des commandes et leurs arguments sont gérés de la même façon.

JOUR 7

GAGNEZ EN EFFICACITÉ AVEC LES SCRIPTS SHELL

Sébastien Chazallet

Nous voici déjà au dernier jour de notre guide. Il est temps maintenant de voir quelques exemples concrets afin de mettre en œuvre tout ce qui a été vu jusqu'ici et de capitaliser nos connaissances.

JOUR 7

Cet article va présenter un certain nombre de scripts potentiellement utiles dans la vie réelle. Ce n'est pas pour autant qu'ils sont démesurément complexes. On va simplement se contenter d'illustrer concrètement comment utiliser tout ce que vous avez appris jusqu'ici.

1 METTRE AUTOMATIQUEMENT À JOUR SON SYSTÈME

Ce script est valable pour un système basé sur Debian, mais il serait très facilement adaptable pour une Fedora ou tout autre système possédant un gestionnaire de paquets.

Fichier

```
01: #!/bin/bash
02: apt-get update &> /tmp/apt-list.log
03: apt-get upgrade -Ryq > /tmp/apt-info.log 2> /tmp/apt-error.log
04: if [ $? -eq 0 ] ; then
05:     aptitude clean
06: else
07:     cat /tmp/apt-error.log | \
08:       mail -s "[Maintenance] Échec MAJ" user@mail.com
09: fi
```

Voici quelques explications additionnelles :

- ligne 1 : permet de déclarer l'interpréteur à utiliser ;
- ligne 2 : permet de mettre à jour la liste des dépôts ; les deux sorties sont redirigées vers un fichier ;
- ligne 3 : permet de mettre à jour le système ; les deux sorties sont redirigées vers deux fichiers distincts ;
- ligne 4 : on teste si la dernière commande s'est bien passée ;
- ligne 5 : si oui, on supprime les fichiers pendant la mise à jour qui ne sont plus utiles ;
- lignes 7-8 : sinon, on envoie un courriel à l'administrateur de la machine pour le prévenir.

Les avantages de ce script sont que l'on n'envoie un courriel qu'en cas de problème, et non pas à chaque exécution pour éviter le spam. À tout moment, l'administrateur peut se connecter et vérifier le contenu des fichiers de logs pour vérifier que la dernière mise à jour s'est bien passée (ces fichiers sont écrasés à chaque utilisation du script).

Les lignes 3 et 7-8 méritent peut-être une explication supplémentaire. En effet, les options **-R -q** et **-y** de la commande **aptitude** permettent de ne pas avoir à gérer les recommandations, de minimiser la sortie standard et de répondre par l'affirmative à toutes les questions dont la réponse, à l'exception des choix dangereux pour le système.

La commande suivante permet d'envoyer un courriel :

Fichier

```
$ mail -s "Sujet" user@mail.com <<< "corps du message"
```

On voit comment sont passés les principaux paramètres. Le triple chevron permet d'insérer un flux de données, qui est le corps du message. Sans cela, la commande **mail** nécessite de saisir ce message à la main. Dans l'exemple plus haut, le message provient du flux généré par la commande **cat** qui va lire le fichier de logs contenant les erreurs rencontrées par la commande **apt-get upgrade**.

Pour terminer, s'il faut faire tourner ce script tous les jours, à 2 heures du matin, par exemple, il est nécessaire de modifier la crontab de l'administrateur (qui seul peut utiliser la commande **apt-get**) de cette manière :

Terminal

```
# crontab -e
# m    h     dom    mon    dow    command
Ø      2     *      *      *      /path/to/script.sh
```

Il est cependant important de noter qu'en cas de problème qui nécessite une décision de l'utilisateur, le programme restera en attente. Cette solution est donc largement perfectible.

Il fait également savoir qu'il existe d'autres solutions, comme le paquet **unattended-upgrades** par exemple, pour régler ce type de problématique.

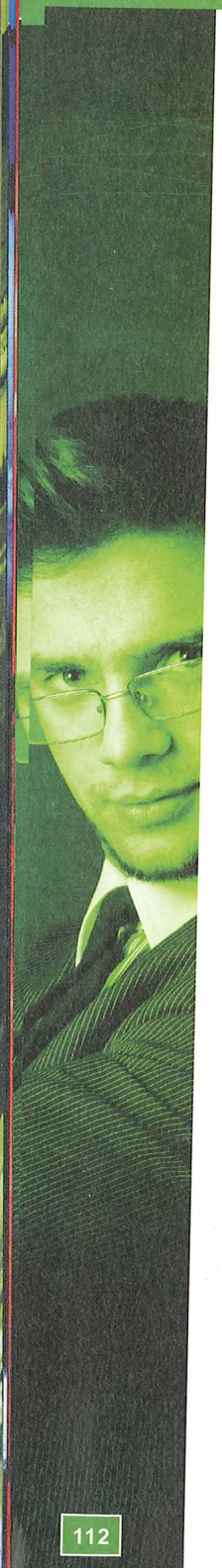
2 SAUVEGARDER UNE BASE DE DONNÉES

L'idée consiste à sauvegarder dans le détail une base de données en mettant la structure dans le fichier et les données dans un autre. Ces dernières sont entrées une par une (une commande SQL par donnée).

Ceci est totalement contre-productif lorsque l'on souhaite faire des sauvegardes et se permettre de restaurer rapidement, mais est l'idéal lorsque l'on veut faire des comparaisons entre deux exports. On peut ainsi immédiatement isoler les changements de structures (colonnes ou tables) et les changements de données.

Fichier

```
01: #!/bin/bash
02: if [ ! $# -eq 1 ] ; then
03:    echo "Usage : './dump.sh db_name'"
04:    exit 1
05: fi
06: BASE=$1
07: USER=""
08: PASS=""
09: while [ -z $USER ] ; do
10:     read -p "Nom d'utilisateur: " USER
11: done
12: read -sp "Mot de passe: " PASS
13: mysqldump \
14:    --default-character-set=utf8\
15:    --comments \
16:    --no-data \
17:    --add-drop-database \
18:    --add-drop-table \
19:    --result-file=struct.sql \
20:    --user=$USER \
21:    --password=$PASS \
22:    --databases \
23:    $BASE
```



Fichier

```
24: if [ $? = "0" ] ; then
25:    echo "Export of structure : success"
26: else
27:    echo "Import of structure : FAIL"
28:    exit 2
29: fi
30: mysqldump \
31:    --default-character-set=utf8\
32:    --comments \
33:    --skip-extended-insert \
34:    --complete-insert \
35:    --no-create-db \
36:    --no-create-info \
37:    --skip-add-locks \
38:    --skip-disable-keys \
39:    --result-file=data.sql \
40:    --user=$USER \
41:    --password=$PASS \
42:    $BASE
43: if [ $? = "0" ] ; then
44:    echo "Export of datas      : success"
45: else
46:    echo "Import of datas      : FAIL"
47:    exit 3
48: fi
49: cp struct.sql struct_'date +%Y%m%d_%H%M%S'.sql
50: cp data.sql data_'date +%Y%m%d_%H%M%S'.sql
```

Voici quelques explications complémentaires :

- lignes 2 à 5 : permet de vérifier que l'on a bien un et un seul argument ;
- lignes 6 à 8 : cet argument est le nom de la base, il est mis dans une variable ; les autres variables sont initialisées à vide ;
- lignes 9 à 11 : on demande le nom de l'utilisateur, la donnée étant obligatoire, on boucle tant qu'on a vide ;
- ligne 12 : on demande le mot de passe, il peut être vide ;
- lignes 13 à 23 : on exporte la structure de la base de données dans le fichier `struct.sql` ;
- lignes 24 à 29 : on vérifie que cela s'est bien passé et l'on gère l'affichage ;
- lignes 30 à 42 : on réalise l'export des données ;
- lignes 43 à 48 : on vérifie que cette opération s'est correctement terminée et on gère l'affichage ;
- lignes 49 et 50 : on duplique les deux fichiers en mettant une référence temporelle dans leur nom, ce qui leur garantit autant l'unicité que le fait de pouvoir en garder trace.

Ce script paraît beaucoup plus long qu'il ne l'est réellement à cause du fait que l'on écrive les deux commandes principales sur plusieurs lignes. Ces commandes peuvent d'ailleurs être parfaitement maîtrisées en allant vérifier leur fonctionnement dans le manuel et ainsi comprendre le tenant et les aboutissants de chaque option.

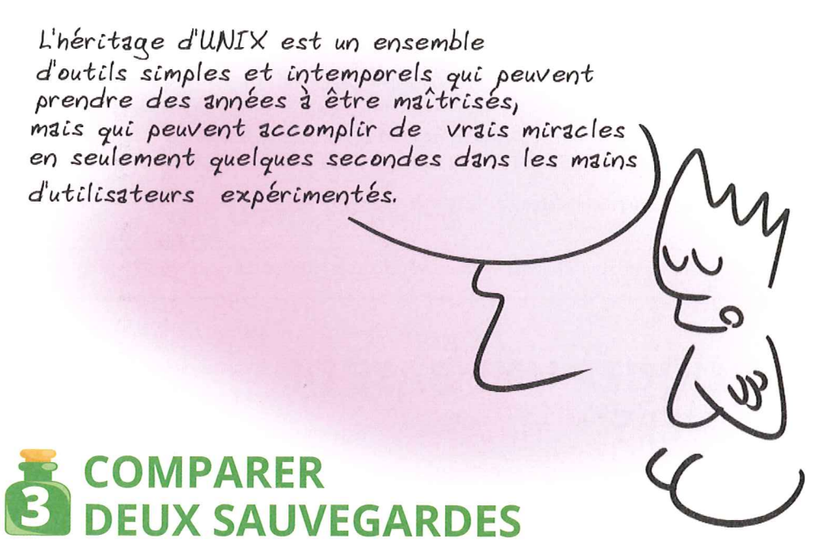


Illustration basée sur le GKND Creator (https://framalab.org/gknd-cre). Œuvre originale de Simon « Cee » Giraudot. Licence Creative Commons By-Sa.

3 COMPARER DEUX SAUVEGARDES

On va poursuivre avec l'idée précédente et mettre au point un programme permettant de mettre en évidence les modifications entre deux versions de la base de données. Il y a plusieurs manières de faire cela. Pour l'exercice et afin de voir différentes commandes, on va afficher le nombre de différences par fichier. Le programme va prendre en paramètre un *timestamp* et faire la comparaison par rapport au dernier fichier généré.

Fichier

```
01: #!/bin/bash
02: if [ ! $# -eq 1 ] ; then
03:    echo "Usage : './dump.sh timestamp'"
04:    exit 1
05: fi
06: STRUCT_FILE="struct_'echo $1'.sql "
07: DATA_FILE="data_'echo $1'.sql "
08: if [ ! -f $STRUCT_FILE ];do
09:     echo "Le fichier $STRUCT_FILE n'existe pas"
10:     exit 1
11: if
12: if [ ! -f $DATA_FILE ];do
13:     echo "Le fichier $DATA_FILE n'existe pas"
14:     exit 2
15: fi
16: echo "structure : 'diff -d $STRUCT_FILE struct.sql | grep
--regex ^[\>\<] | wc -l' différences"
17: echo "données   : 'diff -d $STRUCT_FILE struct.sql | grep
--regex ^[\>\<] | wc -l' différences"
```

Voici quelques explications nécessaires :

- lignes 2 à 5 : on vérifie que l'on a bien un et un seul argument ;
- lignes 6 et 7 : on reconstruit le nom des deux fichiers à explorer ;
- lignes 8 à 15 : on vérifie que les deux fichiers existent bien.
- lignes 16 et 17 : le cœur du programme :

- la commande `diff` va montrer les différences entre les fichiers en précisant des informations comme le fait qu'il s'agisse d'un ajout, d'une modification ou encore d'une suppression (chaque ligne en plus sera préfixée par `>` et chaque ligne en moins par `<`) ;
- le `grep` utilise une expression régulière pour rechercher ces caractères expressément en début de ligne ;
- la commande `wc -l` permet de compter le nombre de lignes ainsi obtenues ;
- enfin, l'ensemble est une sous-commande dont le résultat est injecté dans un affichage.

4 DUPLIQUER UNE BASE DE DONNÉES

Les deux exemples permettent de voir des utilisations originales du shell. Mais on ne peut pas aborder cette partie sans aborder la manière de dupliquer une base de données afin de se créer une base de test similaire à une base de production. Ceci permet par la suite de tester une migration de données ou de faire des tests fonctionnels en se donnant la possibilité de tout casser sans que cela revête la moindre importance.

Fichier

```
01: #!/bin/bash
02: if [ ! $# -eq 1 ] ; then
03:    echo "Usage : './duplicate.sh db_name'"
04:    exit 1
05: fi
06: BASE=$1
06: FILENAME="save_'echo $1'_'date +%Y%m%d_%H%M%S'.sql "
07: USER=""
08: PASS=""
 09: while [ -z $USER ] ; do
10:     read -p "Nom d'utilisateur: " USER
11: done
12: read -sp "Mot de passe: " PASS
13: mysqldump \
14:    --default-character-set=utf8\
15:    --comments \
17:    --add-drop-database \
18:    --add-drop-table \
19:    --result-file=$FILENAME \
20:    --user=$USER \
21:    --password=$PASS \
22:    --databases \
23:    $BASE
24: if [ $? = "0" ] ; then
25:    echo "Export of database : success"
26: else
27:    echo "Import of database : FAIL"
28:    exit 2
29: fi
30: mysql --user=$USER --password=$PASS 'echo $BASE'_test <
$FILENAME
31: if [ $? = "0" ] ; then
32:    echo "Import of database : success"
33: else
34:    echo "Import of database : FAIL"
35:    exit 3
36: fi
```

Voici le détail des opérations importantes et nouvelles :

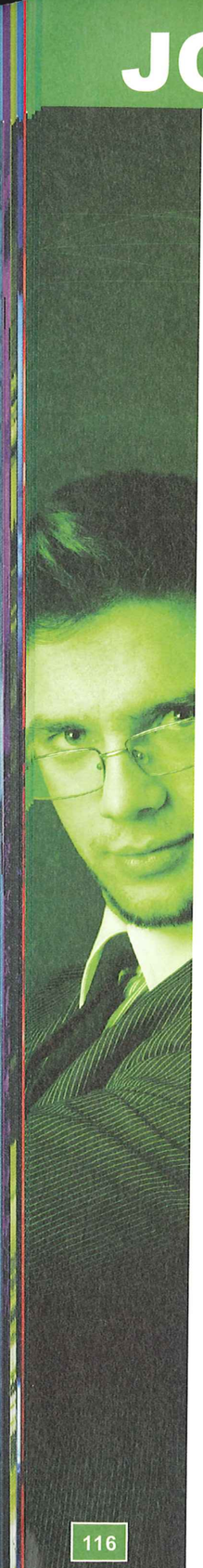
- ligne 6 : on construit le nom du fichier contenant l'export des données à partir du nom de la base de données et de la date ;
- lignes 13 à 23 : on réalise un export classique ;
- ligne 30 : on réalise l'import des données dans la nouvelle base, dont le nom est construit d'après celui de la base initiale.

5 RECHERCHER ET SUPPRIMER

L'idée du script suivant est de rechercher des fichiers et, pour chacun d'entre eux, de demander à l'utilisateur s'il souhaite les supprimer ou non. Pour cela, on va procéder en deux étapes. La première va consister en la définition d'une fonction permettant de supprimer un fichier si l'utilisateur le valide.

Fichier

```
01: #!/bin/bash
02: if [ ! $# -eq 1 ] ; then
03:    echo "Usage : './supprimer.sh db_name'"
04:    exit 1
05: fi
06: function supprimer
07: {
08:     echo $fichier
09:     read -n 1 -p "Voulez-vous supprimer ce fichier ? [o/N] "
choix
10:     case $choix in
11:         o|O )
12:             echo "** $fichier supprimé **"
13:             rm $fichier
14:             ;;
15:         n|N|* )
16:             echo ".. $fichier conservé .."
17:             ;;
18:     esac
19: }
20: for fichier in $(find . -name \*'echo $1'\*) ; do
21:     supprimer $fichier
22: done
```



Voici les points importants :

- lignes 6 à 19 : fonction principale :
  - ligne 9 : on demande à l'utilisateur la confirmation de la suppression après avoir affiché le fichier ; s'il tape `o`, le fichier est détruit. Dans les autres cas, il ne l'est pas ;
  - ligne 10 à 18 : lecture de la réponse de l'utilisateur à l'aide d'un `case` ;
  - ligne 13 : suppression du fichier effective.
- lignes 20 à 22 : boucle principale :
  - on réalise un `find` pour trouver dans le répertoire courant la liste des fichiers à supprimer ; on utilise deux *wildcard* avant et après le mot de recherche ;
  - on doit insérer une commande dans une commande dans une commande, on utilise donc les *quotes* inverses ainsi que le `$()` pour arriver à nos fins ;
  - le cœur de la boucle principale se contente d'appeler la fonction bash.

À noter que le script doit être testé avec prudence. Pour éviter toute mauvaise manipulation, la ligne 13 peut être commentée (ou supprimée).

6 CRÉER DES IMAGES MINIATURES

L'idée du script suivant est de créer des miniatures pour toutes les images d'un répertoire. Ces miniatures sont mises dans un répertoire de destination. Dans le cas où source et destination sont identiques, les images seront effectivement mises dans le même répertoire, mais leur nom sera suffixé par `_mini`.

Fichier

```
01: #!/bin/bash
02: if [ ! $# -eq 2 ] ; then
03:     echo "Usage : './mini.sh src dest'"
04:     exit 1
05: fi
06: if [ ! -d $1 ] || [ ! -d $2 ] ; then
07:     echo "La commande ne prend que des répertoires comme
paramètre"
08:     exit 2
09: fi
10: if [ $1 = $2 ] ; then
11:     suffix="_mini"
12: else
13:     suffix=""
14: fi
15: for source in $( find $1 -maxdepth 1 \( -name \*.jpg -o
-name \*.png \) ) ; do
16:     fichier=$(basename "$source")
```


Fichier

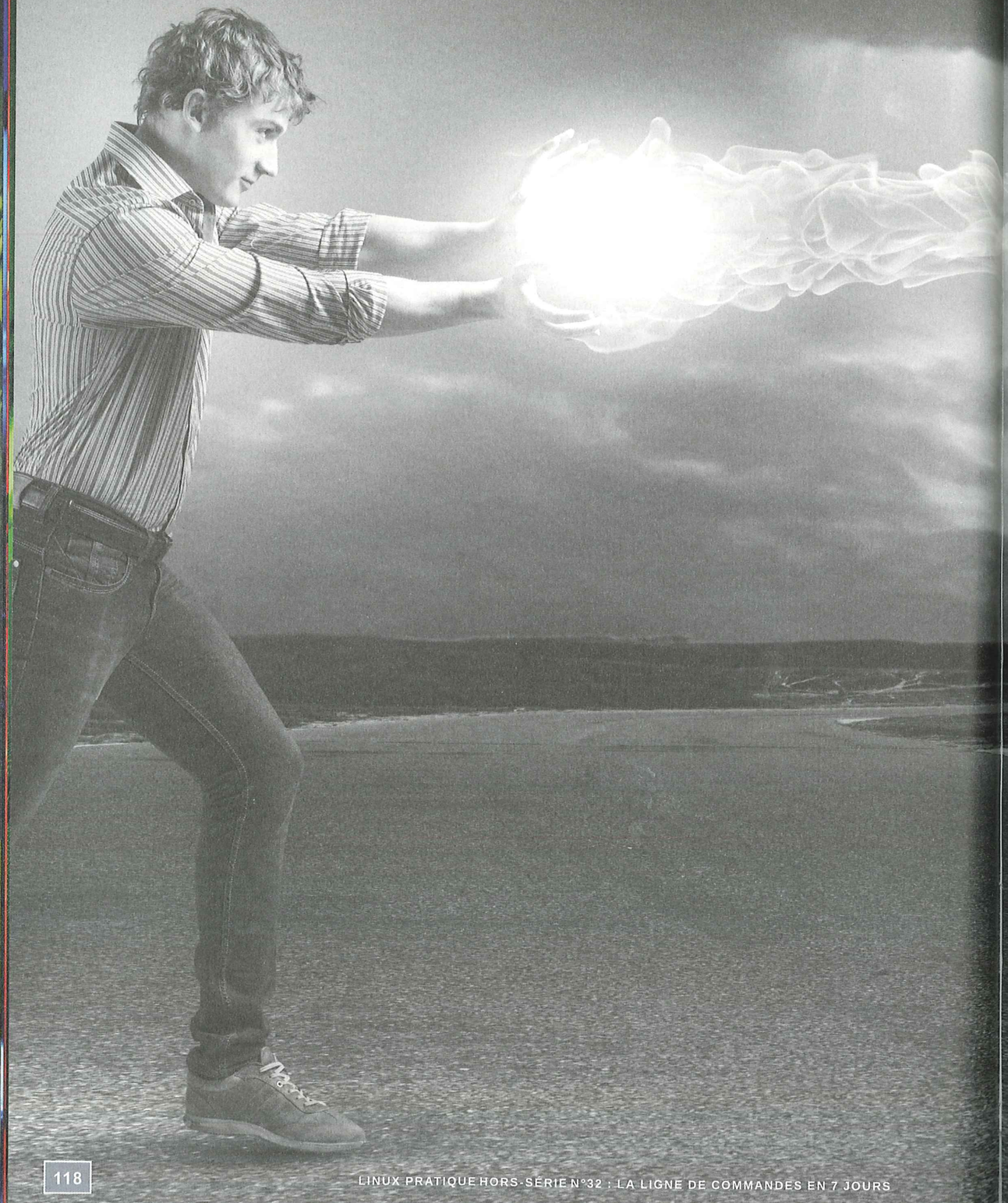
```
17:     destination="'echo $2'/'echo ${fichier%.*}''echo
$suffix'.'echo ${fichier##*.}'"
18:     echo "conversion de $source vers $destination"
19:     convert $source -resize 64x64\! $destination
20: done
```

Voici des éclairages sur les éléments nouveaux :

- lignes 6 à 9 : permet de s'assurer que les deux paramètres sont bien des répertoires ;
- lignes 10 à 14 permet de gérer le suffixe si source et destination sont identiques ;
- ligne 15 : permet d'itérer sur l'ensemble des images du répertoire source :
  - on utilise `maxdepth` pour itérer ne pas descendre dans les sous-répertoires ;
  - on utilise l'option `-o` pour aller chercher deux extensions de fichiers.
- ligne 16 : à partir du chemin complet d'une source, on extrait le nom du fichier ;
- ligne 17 : construction du chemin complet destination correspondant à une source particulière :
  - les règles de nommage sont personnalisables, elles correspondent à l'énoncé ci-dessus ;
  - on récupère l'extension en utilisant `##` qui permet de récupérer la partie droite après le point ;
  - on récupère le nom du fichier sans l'extension à l'aide du pourcentage (un seul) ; on aurait pu utiliser `basename` pour ce faire, mais l'idée est de faire l'analogie avec l'utilisation précédente du dièse ;
  - si le nom du fichier a plusieurs points, seul le dernier point est utilisé comme séparateur étant donné que l'on utilise un seul pourcentage, mais deux dièses ;
- ligne 19 : appel à la commande permettant de redimensionner une image. ▮

Récapitulatif

- Nous avons mis en pratique beaucoup de notions vues dans les jours précédents, en particulier le jour 6.
- Peu importe la tâche à réaliser, ce qui compte est d'utiliser la bonne commande ou le bon ensemble de commandes, avec la ou les bonnes options et de les enchaîner d'une manière logique et organisée.
- Enchaîner les commandes peut se faire par l'utilisation de pipes ou de sous-commandes.
- Un des points essentiels d'un programme bien fait consiste à vérifier tous les cas d'utilisation et à prévoir les problèmes potentiels afin de protéger le système et d'avertir l'utilisateur s'il demande quelque chose qui ne peut pas ou ne doit pas être fait. Entre autres, les arguments doivent être vérifiés.



INDEX

100 COMMANDES ESSENTIELLES À RETENIR

Sébastien CHAZALLET

INDEX

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES
MANIPULATION DE FICHIERS ET RÉPERTOIRES		
`pwd`	Renvoie le chemin absolu du répertoire courant	`$ pwd` `$ pwd -L` # chemin logique (peut contenir des liens symboliques)
`ls` [options] [chemin]	Affiche la liste des fichiers et répertoires du répertoire (par défaut le répertoire courant)	`$ ls /home/moi/*.txt` `$ ls -Al` # liste détaillée récursive
`stat` chemin	Retourne les métadonnées du fichier ou répertoire	`$ stat fichier.txt` `$ stat répertoire`

`cat` fichier	Affiche le contenu d'un fichier ou permet d'écrire dedans	`$ cat fichier.txt`
		`$ cat << EOF > /tmp/fichier.txt`
		# permet de saisir du texte et de l'enregistrer dans le fichier

`file` fichier	Renvoie le type du fichier	`$ file image.jpg`
`cd` [chemin]	Permet de changer de répertoire, par défaut vers le répertoire `HOME`	`$ cd -` # retourne dans le répertoire précédent `$ cd ../../chemin/relatif`

`touch` fichier	Crée un fichier ou met à jour ses métadonnées	`$ touch fichier.txt`
		`$ touch -at 01162000 test.txt`
		# 16/01, 20h

`mkdir` répertoire	Permet de créer un répertoire	`$ mkdir ../répertoire_frère` `$ mkdir -p a/b/c` # crée (s'ils n'existent pas) les répertoires `a`, puis `a/b`, puis `a/b/c`
`rmdir` répertoire	Permet de supprimer un répertoire vide	`$ rmdir ../répertoire_frère` `$ rmdir -p a/b/c` # supprime les répertoires `a/b/c`, puis `a/b`, puis `a` (s'ils ne contiennent pas de fichiers)
`cp` [-r] source destination	Copie un fichier ou un répertoire et son contenu	`$ cp fichier1 fichier2` `$ cp -r répertoire1 répertoire2`
`mv` source destination ou `mv` -t destination source ...	Permet de déplacer ou renommer un fichier ou un répertoire	`$ mv fichier1 répertoire/fichier2` # déplacement ET renommage `$ mv -t photos *.png` # déplacement de toutes les images PNG vers le répertoire photos
`rm` chemin ...	Supprime tous les fichiers ou répertoires listés	`$ rm *.jpg` # supprime toutes les images JPEG du répertoire courant `$ rm -r *~ *.bak *.old` # supprime tous les fichiers indésirables de l'arborescence
`chmod` [options] droits chemin	Modification des permissions sur un fichier ou répertoire	`$ chmod 755 fichier` # lecture et exécution pour tout le monde, écriture pour l'utilisateur propriétaire `$ chmod -R ug+w répertoire` # rajout de la permission d'écriture pour le propriétaire et les membres du groupe propriétaire

`chown` [options] user:group chemin	Modification de l'utilisateur et du groupe propriétaire d'un fichier ou d'un répertoire	`$ chown user: fichier`
		# changement de l'utilisateur propriétaire
		`$ chown -R :group répertoire`
		# changement du groupe propriétaire du répertoire et de son contenu

`ln` fichier1 fichier2	Création d'un lien matériel ou symbolique vers un fichier (de la même partition si matériel)	`$ ln nom_fichier nom_lien_materiel` `$ ln -s nom_fichier1 nom_lien_symbolique`

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES

`diff` [options] fichier1 fichier2	Permet de comparer deux fichiers textuels	`$ diff -rc 6 rep_v1 rep_v2`
		# permet de comparer deux répertoires en donnant 6 lignes de contexte
		`$ diff -pq module_v1.c module_v2.c`
		# permet de faire une comparaison minimale en indiquant le nom des fonctions C dans lesquelles il y a des différences

`cmp` [options] fichier1 fichier2	Permet de comparer deux fichiers octet par octet	`$ cmp -l script_v1 script_v2` # `-l` pour avoir les numéros et valeurs des octets différents

`tar` [-xc][-zj][-v] -f destination source ...	Permet de gérer des archives (`x` extrait et `c` compresse, `z` pour gzip, `j` pour bzip2, `f` pour préciser le nom de l'archive)	`$ tar -xzf archive.tar.gz`
		# extrait l'archive
		`$ tar -cjf archive.tar.bz2 *.py src/`
		# compresse tous les fichiers python du répertoire courant plus le répertoire `src` dans l'archive

RECHERCHE D'UN FICHIER		

`find` rep_base [options]	Recherche de fichiers par le parcours du répertoire de départ (rep_base) et sa descendance	`$ find ~/Documents -name \*.odt`
		# permet de trouver, dans le répertoire `~Documents`, tous les documents dans lesquels se trouve la chaîne `.odt`
		`$ find ~/Documents -regex \.odt$`
		# permet de trouver, dans le répertoire `~Documents`, tous les documents dont le chemin se termine par la chaîne `.odt`

`updatedb`	Mise à jour de la base d'indexation utilisée par la commande `locate`	`$ sudo updatedb`

`locate` chemin	Permet d'effectuer une recherche rapide d'un chemin	`$ locate odt`
		`$ locate --regex \.odt$`
		# permet de trouver dans la table d'index tous les documents dont le chemin se termine par la chaîne `.odt`

`which` commande	Les commandes externes sont fournies par des fichiers, `which` permet de les retrouver	`$ which python` `$ which bash`

`whereis` -bmsu commande	Localise l'exécutable (comme `which`), les sources et la page de manuel d'une commande	`$ whereis python`
		`$ whereis -m bash`
		# `-m` pour les pages de manuel

MANIPULATION DE FLUX		
`more` flux	Affiche le contenu page par page (navigation avec la barre espace)	`$ more fichier` `$ ls -Al | more` # afficher le résultat de la commande `ls` page par page
`less` flux	Affiche le contenu en permettant une navigation souple avec les touches de direction	`$ less fichier` `$ ls -Al | less` # afficher le résultat de la commande `ls` en permettant la navigation
`nl` flux	Affiche le contenu d'un flux en numérotant les lignes	`$ nl fichier` `$ ls -Al | nl` # affiche le résultat de la commande `ls` en numérotant les lignes

INDEX

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES

`head` [-n] flux	Affiche les premières lignes d'un flux	`$ head fichier`
		`$ ls -Al | head -n 42`
		# n'affiche que les 42 premières lignes du résultat de la commande `ls`

`tail` flux	Affiche les dernières lignes, l'option `-f` permet de garder le flux ouvert et de voir les arrivées en temps réel	`$ tail -f /var/log/apache/error.log`
		# affiche le flux de `error.log` dans la sortie standard au moment où celui-ci est généré
		`$ ls -Al | tail -n 42` # n'affiche que les 42 dernières lignes du résultat de la commande `ls`

`grep` motif flux	N'affiche que les lignes du flux correspondant à la recherche	`$ grep -ri function .` # recherche toutes les lignes contenant le mot function dans le répertoire courant et ses sous-répertoires `$ ls -Al | grep --regex ^d` # ne renvoie que les répertoires

`sed` motif flux	Permet de filtrer et modifier le contenu d'un flux	`$ sed -n "/^[A-Z]/p" fichier.txt`
		# retourne toutes les lignes commençant par une majuscule
		`$ ls -Al | sed -n "/^d/p"`
		# ne renvoie que les répertoires

`awk` motif flux	Permet de filtrer et modifier le contenu d'un flux	`$ ls -Al | awk "/^d/"` # ne renvoie que les répertoires
`cut` [options] flux	Permet de découper chaque ligne du flux	`$ cut -d : -f 1 /etc/passwd` # permet d'obtenir uniquement la première colonne du fichier `/etc/passwd` (les colonnes étant séparées par des deux-points)
`sort` [options] flux	Permet de trier le contenu d'un flux, l'option `-r` permet d'inverser le tri	`$ du -ach | sort -h` # le `-h` permet de comprendre la différence entre o, ko, Mo, Go...
`wc` [-wlmc]	Permet de compter des éléments sur un flux	`$ wc -c fichier` # nombre d'octets du fichier `$ wc -m fichier` # nombre de caractères du fichier
GESTION DES UTILISATEURS		
`su` [-] [user]	Permet de changer d'utilisateur (le tiret permet de relire les scripts d'initialisation de son environnement)	`$ su -` # passer en root avec sa configuration `$ su postres -s /bin/bash` # passer en utilisateur `postgres` en gardant sa configuration et en choisissant le shell à exécuter `$ su - postgres -c "createuser -s pg_user"` # permet d'exécuter une commande en utilisant l'utilisateur `postgres`

`sudo` [-u user]	Permet d'exécuter une commande en tant qu'autre utilisateur	`$ sudo cat /etc/apt/sources.list`
		# exécuter la commande `cat` au nom du root
		`$ sudo -u postgres createuser -s pg_user` # permet d'exécuter une commande en utilisant l'utilisateur `postgres`

`visudo`	Permet de paramétrer la liste des utilisateurs habilités à utiliser la commande `sudo`	`$ visudo`

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES
`adduser` [options] ou `adduser` user group	Permet de créer un utilisateur ou d'ajouter un utilisateur à un groupe	`# adduser guest cups` # permet à l'utilisateur `guest` d'utiliser l'imprimante (en le rajoutant au groupe `cups`) `# adduser django --system --home=/opt/django --shell /bin/false --group` # crée un utilisateur technique
`passwd` user	Permet de saisir le mot de passe d'un utilisateur	`# passwd invite`
`deluser` user	Permet de supprimer un utilisateur	`# deluser invite`
`addgroup` group	Permet de créer un groupe	`# addgroup famille`
`delgroup` group	Permet de supprimer un groupe	`# delgroup famille`
`users`	Donne la liste des utilisateurs connectés au système	`# users`
`who`	Permet de donner des informations sur les utilisateurs connectés au système	`# who -a` # affiche tous les utilisateurs connectés `$ who am i` # affiche les informations sur le compte que l'on utilise
`w`	Permet de savoir quels sont les utilisateurs connectés et leur activité	`# w`
`finger`	Permet de donner des informations sur les utilisateurs connectés au système	`# finger -l` # sur plusieurs lignes
`last`	Permet de donner la liste des derniers utilisateurs connectés au système	`# last`
`lastb`	Permet de donner la liste des dernières tentatives de connexion échouées	`# lastb`
`env`	Permet de gérer les variables d'environnement	`$ env` # affiche toutes les variables d'environnement `$ env -u VAR` # permet de supprimer la variable d'environnement nommée `VAR`
GESTION DES DISQUES		
`mount` disque [rep]	Permet de monter un système de fichiers	`$ mount /dev/sdb /media/usb_drive` # monte le disque `SDB` dans le répertoire `usb_drive`
`umount` disque	Permet de démonter un système de fichiers	`$ umount /dev/sdb` # démonte le disque `SDB`
`parted` [options] disque	Permet de gérer les partitions d'un disque	`# parted /dev/sda` # lance le logiciel de partition sur `SDA`
`df`	Renvoie la quantité d'espace disponible (il existe aussi `pydf`)	`$ df -h`
`du` [options] [répertoire]	Permet de visualiser l'espace occupé par les répertoires du répertoire passé en argument	`$ du -sh` `$ du -ach | sort -h` # permet de voir les répertoires les plus volumineux

INDEX

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES
`cdrecord` [options] image ou `cdrecord` [options] vodim	Grave une image sur un support	`$ cdrecord -v speed=4 dev=0,0,0 image.iso` # `speed` permet de fixer la vitesse et `dev` permet de donner l'adresse du graveur
`mkisofs` [options] image rep	Permet de créer une image ISO	`$ mkisofs -r -o image.iso ~/Photos` # crée une image ISO à partir du contenu du répertoire `Photos`
GESTION DU SYSTÈME		
`lsb_release`	Renvoie des informations sur le système d'exploitation	`$ lsb_release -idcr` # renvoie toutes les informations utiles en une commande
`uname`	Renvoie des informations sur le noyau	`$ uname -a` # renvoie toutes les informations `$ uname -r` # renvoie uniquement la version du noyau
`tty`	Affiche l'identifiant du terminal sur lequel on travaille	`$ tty`
`uptime`	Renvoie des statistiques sur la charge système	`$ uptime`
`dmesg`	Renvoie les informations affichées sur la console principale au démarrage de la machine	`$ dmesg`
`vmstat`	Renvoie des statistiques sur les ressources du système (processus, mémoire, CPU...)	`$ vmstat`
`top`	Affiche en temps réel les informations sur les processus en cours d'exécution (il existe aussi `htop`)	`$ top`
`free`	Renvoie les quantités de mémoire libre et utilisée du système	`$ free`
`swapoff`	Permet d'arrêter d'utiliser la swap. Force la recopie de tout ce que la swap contient en mémoire vive, peut être long.	`$ swapoff -a` `$ swapoff -a && swapon -a` # permet de vider le swap
`swapon`	Autorise l'utilisation de la swap.	`$ swapon -a`
`ps`	Retourne les statistiques sur les processus en cours au moment de l'exécution de la commande (non en temps réel)	`$ ps ax` `$ ps auxf | grep -v grep | grep -i -e VSZ -e apache` # permet de visualiser tous les processus dont le motif correspond à `apache`
`kill` [signal] PID	Permet d'envoyer un signal à un processus identifié par son PID	`$ kill -15 19874` # numéro du signal `$ kill -TERM 19874` # nom du signal
`pkill` motif	Permet d'envoyer un signal à un processus identifié par un motif	`$ pkill gedit` # va tuer tous les processus `gedit`, mais aussi, par exemple un processus lancé par la commande `vim gedit_preferences`
`update-rc.d`	Gère les services automatiquement lancés ou pouvant l'être	`# update-rd.d cupsys enable` # active le service `cupsys` `# update-rd.d cupsys disable` # désactive le service `cupsys`

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES
`service`	Gère les démons	`# sudo service apache2 reload` # demande à apache de relire ses fichiers de configuration `# sudo service –status-all` # donne le statut de tous les services configurés
`dpkg`	Interface de bas niveau permettant de gérer un paquet en particulier	`$ dpkg -i paquet.deb` # installe un paquet `$ dpkg -c paquet.deb` # liste les fichiers contenus dans la paquet
`apt-cache` [options] [motif|paquet]	Permet de gérer les métadonnées des paquets	`$ apt-cache search motif` # recherche un paquet dans la base d'index `$ apt-cache show paquet` # affiche les fichiers contenus dans un paquet
`apt-get` [options] paquet ...	Interface permettant de gérer les paquets de son système	`$ apt-get install ipython3-notebook` # installe un paquet `$ apt-get purge ipython-notebook` # supprime un paquet ainsi que ses éventuels fichiers de configuration
`aptitude` [options] [paquet ...]	Interface de haut niveau, utilisant `apt-get`	`$ aptitude` # démarre un client curse `$ aptitude reinstall ipython3-notebook` # réinstalle un paquet
COMMANDES RÉSEAU		
`hostname`	Permet d'afficher ou modifier le nom canonique de la machine	`$ hostname` # donne le nom d'hôte de la machine `# hostname nouveau_nom` # modifie le nom d'hôte de la machine
`ifconfig`	Permet d'afficher ou de modifier la configuration réseau	`$ ifconfig -a` # donne la configuration réseau de toutes les interfaces, connectées ou non `# ifconfig eth0 inet6 dhcp start` # configure l'interface `eth0` en dynamique
`iwconfig`	Idem que précédemment, pour les interfaces sans fil	`$ iwconfig wlan0` # donne la configuration de `wlan0` `# iwconfig wlan0 inet6 dhcp start` # configure l'interface `wlan0` en dynamique

`ping`
ou
`ping6`
	Permet de tester l'accessibilité d'une machine et de recevoir des statistiques	`$ ping python.org`
		`$ ping6 python.org`

`traceroute`
ou
`traceroute6`
	Renvoie le chemin aller entre la machine émettrice et la machine réceptrice	`$ traceroute python.org`
		`$ traceroute6 python.org`

`netstat` [options]	Renvoie des statistiques sur l'état du réseau	`$ netstat -nr` # donne la table de routage `$ netstat -lptn` # affiche les connexions actives
`host` [ip ou nom_domaine]	Renvoie l'IP associée à un nom de domaine ou le nom de domaine principal associé à une IP	`$ host www.inspyration.fr` # donne l'IP associée au nom de domaine `$ host 213.246.53.26` # donne le nom de domaine principal associé à l'IP
`whois` nom_domaine	Renvoie les informations sur le propriétaire d'un nom de domaine	`$ whois python.org`

COMMANDE ET SYNTAXE	RÔLE	EXEMPLES
`wget` [options] url	Télécharge le fichier spécifié par l'URL fournie	`$ wget http://dam.inspyration.fr/remoulins/accueil_01.jpeg`
`arp [options] [ip]`	Permet de visualiser ou mettre à jour la table ARP	`$ arp -a` # affiche la table ARP `$ arp 192.168.1.47` # envoie une requête ARP à la machine identifiée par l'adresse IP pour mettre à jour la table ARP
`nmap` [options] machine	Permet de tester les interfaces réseau d'une machine	`$ nmap -A machine.nom_domaine.com`
DIVERS		

`history`	Donne la liste numérotée des commandes	`$ history`
		`$ history | grep ./manage.py`
		# recherche dans l'historique

`!n`
ou
`!motif`
	Permet de rappeler la commande numéro n (dans l'historique)	`$ !42` # rappelle la commande 42
		`$ !vim` # rappelle la dernière commande utilisant le motif

`alias` alias[='commande']	Permet de visualiser ou définir un alias	`$ alias ll` # affiche l'alias `$ alias ll='ls -Al –color=auto'` # crée ou modifie l'alias
`unalias`	Permet de supprimer un alias	`$ unalias ll`
`echo` [chaine]	Affiche du texte à l'écran	`$ echo "Texte à l'écran: $variable"`
`export`	Permet d'exporter une variable	`$ export MUSIC_PATH='~/music'` `$ export CURRENT_PATH=`pwd``
`read`	Permet de lire une ligne sur l'entrée standard	`$ read -p "D'accord ? [O/n]" -n 1 variable` # permet à l'utilisateur de saisir un caractère `$ read -sp "Mot de passe" password` # permet à l'utilisateur de saisir un mot de passe
`date` [options]	Permet de renvoyer la date actuelle en la formatant ou de la mettre à jour	`$ date -u 25120001` # fixe la date à l'heure d'ouverture des cadeaux `$ mv fichier.txt fichier_`date +%Y%m%d_%H%M%S`.txt` # horodatage d'un fichier
`cal` [options]	Affiche un calendrier	`$ cal -y 2020` # affiche le calendrier de l'année 2020
`bg`	Met en arrière-plan un processus (quitté par [CTRL] + [Z])	`$ bg`
`fg`	Remet au premier plan un processus	`$ fg` `$ fg 2` # remet au premier plan le second processus (voir `jobs` pour voir les numéros)
`jobs`	Permet de visualiser la liste des processus en arrière-plan	`$ jobs`
`whatis` commande	Permet de connaître rapidement l'utilité d'une commande	`$ whatis ls` # affiche le rôle de la commande `ls`
`man` commande	La dernière, mais la plus utile : permet d'ouvrir le manuel d'une commande	`$ man ls` # affiche le manuel de la commande `ls`

Les éléments indiqués entre crochets sont facultatifs. La liste des options de chaque commande n'est pas exhaustive ; n'ont été mentionnées ici que les plus utiles. ▌