Aide-Mémoire UNIX

Nicolas KIELBASIEWICZ *

9 février 2009

Le système d'exploitation UNIX est de plus en plus présent aujourd'hui. Comme tous les projets Open Source, on le trouve sous différentes formes : BSD, System V, Linux / GNU, ... et dans de multiples distributions : Debian, Mandrake, Mandriva, Fedora, Red Hat, FreeBSD, Ubuntu, ...

Ce document s'adresse en particulier aux débutants, mais je l'ai écrit dans l'esprit d'un aide-mémoire contenant tout ce dont vous pourriez avoir besoin et dont j'ai eu moi-même besoin à un moment ou à un autre. Par la suite, je parlerai préférentiellement de LINUX , qui est l'environnement libre donc gratuit.

Table des matières

*Unité de Mathématiques Appliquées, École Nationale Supérieure de Techniques Avancées

1 Préambule

1.1 Les gestionnaires de fenêtres

Lorsqu'on démarre une machine sous LINUX , on a la possibilité lors de la boite de login/passwd de choisir une interface graphique. Il en existe pléthore, mais voici les plus courantes :

1.1.1 Les gestionnaires minimalistes

Voilà ceux que j'ai rencontré, mais il en existe beaucoup d'autres.
- **fvwm**
- **wmaker**
- **twm**

Le coté esthétique et graphique est très peu développé comparativement à Windows, mais cela rend ces gestionnaires plus efficaces car les ressources sont consacrées à l'utilisation et non à l'interface graphique, idem pour l'espace disque nécessaire à l'installation. Pour ma part, je trouve que **wmaker** est le plus agréable. Je l'utilise personnellement quand la machine sur laquelle je suis est sous Windows et dispose de l'émulateur Cygwin afin de travailler sous LINUX , mais il y en a certainement d'autres qui répondront peut-être davantage à vos besoins.

1.1.2 Les gestionnaires complets

Il en existe essentiellement 2 :
- **kde**
- **gnome**

Il s'agit d'interfaces graphiques complètes et sont peut-être plus faciles à appréhender par les débutants qui connaissent Windows, car ils en sont assez proches, avec des menus, des icônes. Bien qu'utilisant UNIX depuis plusieurs années maintenant, je reste attaché à **kde** quand je me logue directement sous LINUX . Cela représente pour moi un confort relativement proche à Mac OS X, les possibilités graphiques avancées en moins (ce qui fait tout l'intérêt d'un Mac OS X en somme par rapport au reste du monde ;-)). L'inconvénient de ces gestionnaires complets, c'est qu'ils sont beaucoup plus gourmands en ressources et jusqu'à la version 4.0, **kde** était encore plus lourd que **gnome**.

1.1.3 Le terminal X

Quel que soit le choix du gestionnaire de fenêtres, si vous avez décidé d'installer un UNIX sur votre machine, vous serez amené à un moment ou à un autre à utiliser un terminal X (”Terminal”, ”xterm”,

"console", ...). Par comparaison, c'est un peu comme un invite de commande MS-DOS, mais en beaucoup plus développé.

1.2 Le Shell

1.2.1 Qu'est-ce qu'un Shell ?

Le Shell est l'interpréteur de commande du système UNIX . C'est un langage de programmation utilisé pour exécuter des commandes ou pour sous forme de programmes exécutables ou scripts.

1.2.2 Les différents Shells

Il existe essentiellement deux interpréteurs Shells en UNIX :
- le *Bourne Shell*, ou **sh** ;
- le *C Shell*, ou **csh**.

Chacun de ces deux Shells possède plusieurs déclinaisons plus ou moins évoluées :
- **tcsh**, ... pour le *C Shell* ;
- **bash**, **ksh**, **zsh**, ... pour le *Bourne Shell*

Au lancement d'un terminal, un certain nombre de commandes et de mise en place de variables d'environnement est chargé. Il est possible d'intéragir et de personnaliser ce démarrage à l'aide, par exemple, des fichiers .bashrc, .cshrc, .zshrc, ..., selon le Shell utilisé.

Chaque Shell à sa spécialité et ses spécificités. Les Shells les plus courants sont **bash** et **tcsh**. Personnellement, j'utilise plutôt **zsh**, juste pour la simplicité de la mise en oeuvre de la complétion automatique et de l'association de commandes avec des extensions de fichiers. Donc, à vous de voir quel Shell vous convient le mieux, cela viendra avec la pratique.

1.2.3 Connaître le Shell

- lire le fichier /etc/passwd, à l'aide de la commande **more** par exemple. La ligne correspondant à votre login vous fournira l'information demandée ;
- exécuter la commande **echo $SHELL** (pour tous les dérivés de **sh**) ou **echo $shell** (pour tous les dérivés de **csh**). Si vous utilisez la mauvaise commande, elle répondra une ligne blanche pour les dérivés de sh. En soi, c'est déjà une information. Précisons ici que la commande **echo** sert de manière générale à afficher le contenu des variables ou des chaînes de caractères sur la sortie standard, qui est généralement l'écran.

J'ajouterai ici qu'il existe une autre variable globale particulièrement intéressante, $PATH , qui contient les chemins complets des répertoires contenant les diverses commandes et autres executables que l'on peut utiliser.

1.2.4 Changer le Shell

- lancer le Shell directement à l'aide d'une des commandes **bash**, **zsh**, **csh**, **tcsh**, ... modifiera le Shell immédiatement. Cette modification sera annulée à la fermeture du terminal ;
- exécuter la commande **passwd -s** ou **chsh** modifiera le Shell de manière permanente.

Maintenant que nous avons vu le background d'UNIX et d'un terminal, il est temps d'explorer les commandes les plus utilisées, en commençant par la navigation dans une arborescence et la manipulation de fichiers et répertoires.

1.2.5 Personnaliser le Shell

Je vais parler ici de quelques commandes utiles pour pouvoir paramétrer son environnement dans le fichier .bashrc, .cshrc, ...

- **alias nom_alias='com1'** # crée un raccourci à la commande com1. Par exemple, si on utilise régulièrement la commande **ls -l** dont on verra plus loin la signification, on a intéret à en définir un alias appelé ll, par la commande **alias ll='ls -l'**. A noter que le nom de cet alias est assez standard.
- **setenv VAR val** et **export VAR=val** sont des commandes dédiées à la définition et à la propagation de variables globales, comme $PATH, à laquelle on pourra par exemple ajouter des chemins vers des répertoires personnels pour configurer certains programmes installés en local. La commande **setenv** est utilisée en *C Shell*, alors que la commande **export** est utilisée en *Bourne Shell*.

2 Naviguer et manipuler

2.1 Accéder à l'aide

- **man nom_com**
- les options -h, -help, –help peuvent être définies.

2.2 Les méta-caractères

Ce sont des caractères spéciaux concernant les noms de fichiers. Il sont au nombre de 6, 5 étant communs au *Bourne Shell* et au *C Shell*.

? Il correspond à un caractère quelconque.
* Il correspond à un ou plusieurs caractères quelconques.
[et] Les crochets encadrent les diverses possibilités que peut prendre un caractère.
- Il intervient seulement entre [et] pour désigner l'ensemble des caractères entre les deux bornes.
! métacaractère spécifique au *Bourne Shell*, il désigne la négation, l'exclusion.
∼ métacaractère spécifique au *C Shell*, il désigne le répertoire racine du compte utilisateur.
\ et ' ' Pour annuler un métacaractère, on le fait précéder de \. Dans une chaine de caractères encadrées par ' ', les métacaractères sont également considérés comme des caractères normaux.

2.3 Les répertoires

2.3.1 Créer un répertoire

- **mkdir nom_dir**
- **mkdir dir1/nom_dir**
- **mkdir -p dir1/nom_dir** # crée également le sous-répertoire nom_dir à dir1. La seule différence avec la commande précédente est que si dir1 n'existe pas, il est aussi créé.

2.3.2 Se déplacer dans un répertoire

- **pwd** # affiche le chemin complet du répertoire courant
- **cd nom_dir**
- **cd** ou **cd ∼** # permet d'aller directement dans le répertoire personnel. Rappelons que ∼ n'est défini que pour les dérivés de csh
- **cd ..** # permet d'aller dans le répertoire parent
- **cd /** # permet d'aller dans le répertoire racine (le répertoire contenant toute l'arborescence UNIX)

2.3.3 Lister le contenu d'un répertoire

- **ls nom_dir** # affiche la liste des noms des fichiers et des sous-répertoires de nom_dir
- **ls -l nom_dir** # affiche les informations détaillées sur les fichiers et les sous-répertoires de nom_dir
- **ls -a nom_dir** # affiche la liste des noms de fichiers et des sous-répertoires de nom_dir, ainsi que les fichiers cachés (dont le nom commence par .)
- **ls** ou **ls .** # liste le contenu du répertoire courant

2.3.4 Copier/déplacer un répertoire

- **cp -R dir1 dir2** # copie le répertoire dir1 dans dir2. Si ce dernier existe, il est créé et contient un sous-répertoire dir1
- **cp -R dir1/ dir2** # copie le contenu du répertoire dir1 dans dir2, éventuellement créé.
- **mv dir1 dir2** # déplace dir1 dans dir2. Si dir2 existe, dir1 en devient un sous-répertoire.

2.3.5 Supprimer un répertoire

- **rmdir nom_dir** # si le répertoire est vide
- **rm -r nom_dir** # si le répertoire n'est pas vide

2.4 Les fichiers

2.4.1 Créer/éditer un fichier

Je ne parlerai pas ici des éditeurs de texte qui permettent bien évidemment de créer et éditer des fichiers.

- **touch nom_fichier** # permet de créer un fichier vide
- **nom_com > nom_fichier** # écrit le résultat de la commande nom_com dans le fichier. SI le fichier existe, il est écrasé, sinon, il est créé
- **nom_com ≫ nom_fichier** # concatène le résultat de la commande nom_com à la fin du fichier. Si par contre le fichier n'existe pas, il est créé ;
- **cat fich1 fich2** # concatène le contenu de fich1 et de fich2 et affiche le résultat à l'écran. Pour écrire dans un fichier, il suffit alors d'utiliser une des 2 commandes de redirection > ou ≫

2.4.2 Copier/déplacer un fichier

- **cp file1 dir1** # copie le fichier file1 dans le répertoire dir1
- **mv file1 dir1** # déplace le fichier file1 dans le répertoire dir1

2.4.3 Supprimer un fichier

- **rm file1**

2.4.4 Les liens

Ce sont des fichiers particuliers qui pointent vers d'autres fichiers.

- **ln nom_file nom_lien** # crée un lien physique de nom_file. Un lien physique, c'est un peu comme une copie, sauf que les modifications sont instantanément propagées dans les liens physiques. Par ailleurs, pour pouvoir réellement supprimer le fichier nom_file, il faut aussi supprimer tous les liens physiques associés
- **ln -s nom_file nom_lien** # crée un lien symbolique de nom_file, un peu comme un raccourci sous Windows. Si on supprime nom_file, alors le lien sera mort, et le fichier aura réellement disparu.

2.4.5 Effectuer des recherches

- **locate nom_file** # recherche un fichier
- **find nom_dir -name pattern** # recherche dans l'arborescence à partir de nom_dir tous les fichiers contenant pattern

2.4.6 Manipuler des fichiers

- **grep pattern nom_file** # affiche toutes les lignes de nom_file contenant pattern
- **grep -v pattern nom_file** # affiche toutes les lignes de nom_file ne contenant pas pattern
- **wc nom_file** # donne le nombre de lignes, de mots, et de caractères contenus dans nom_file
- **wc -l nom_file** # donne le nombre de lignes contenues dans nom_file
- **wc -c nom_file** # donne le nombre de caractères contenus dans nom_file
- **wc -w nom_file** # donne le nombre de mots contenus dans nom_file
- **sort -n nom_file** # permet de trier les lignes d'un fichier suivant l'ordre arithmétique croissant
- **sort -d -r nom_file** # permet de trier les lignes d'un fichier suivant l'ordre des répertoires téléphoniques (-d) et dans l'ordre décroissant (-r)

2.5 Les droits d'accès

2.5.1 Lire les droits d'accès d'un fichier ou d'un répertoire

La commande **ls -l nom_file** est le moyen le plus rapide de connaître les droits d'accès d'un fichier. Il s'agit de décrypter cette séquence de 10 caractères.

- le premier caractère est soit vide (c'est-à-dire un tiret) soit la lettre d (pour caractériser un répertoire), soit la lettre l (pour caractériser un lien) ;
- les 9 autres caractères sont à lire par groupe de 3. Le premier concerne le possesseur du fichier (dénommé u), le second groupe concerne les autres utilisateurs membres du groupe dans lequel se trouve le possesseur (dénommé o), et le dernier groupe concerne tous les autres utilisateurs (dénommé a).

  Dans chaque groupe, le premier caractère est soit vide soit la lettre r, le second soit vide soit la lettre w, et le troisième soit vide soit la lettre x. La signification de ces droits dépend selon si c'est un fichier ou un répertoire :

  r pour un fichier, il s'agit des droits de lecture. Pour un répertoire, cela se traduit par le droit de lister le contenu ;

  w pour un fichier, il s'agit des droits d'écriture du fichier. Pour un répertoire, cela se traduit par le droit de modifier le contenu, c'est-à-dire copier/supprimer/renommer des éléments qu'il contient ;

  x pour un fichier, il s'agit des droits d'exécution. Pour un répertoire, il s'agit des droits d'accès.

2.5.2 Modifier les droits d'accès d'un fichier ou d'un répertoire

Pour modifier manuellement les droits d'accès d'un fichier ou d'un répertoire, on utilise la commande :

chmod 644 nom_file

Que signifie ces 3 chiffres ? Ils représentent en octal les valeurs des droits pour respectivement le possesseur, le groupe du possesseur, et les autres utilisateurs. Dans cet exemple 644 correspondra donc à rw-r–r– .

On peut également utiliser la commande de la façon qui suit :

chmod +x nom_file
chmod o-w nom_file

On ajoute/retire un droit particulier à un ou plusieurs des 3 groupes, en utilisant u, o, et a pour les caractériser, + pour ajouter et - pour retirer, et r, w ou x pour le droit concerné.

On peut également modifier les droits d'accès par défaut à la création d'un fichier ou d'un répertoire. on utilise pour cela la commande umask :

umask 022

Cette fois-ci, les chiffres ont une valeur complémentaire par rapport à la commande **chmod** . Le premier chiffre indique ce que l'on interdit. Ainsi, 0 correspond à donner tous les droits, 1 correspond à donner les droits en lecture et en écriture, ... Dans notre exemple, 022 signifie donc -rwxr-xr-x.

De la même manière, le premier des trois chiffres concerne l'utilisateur, le second le groupe de l'utilisateur, et le troisième, le reste du monde.

2.6 Archiver et compresser

Voici des exemples d'utilisation classique des commandes tar, gzip, gunzip :
- **tar -cvf archive.tar file1 file2 ... fileN dir1 ... dirP** # crée une archive archive.tar
- **tar -xvf archive.tar** # extrait l'ensemble des fichiers contenus dans archive.tar
- **gzip archive.tar** # compresse archive.tar et crée l'archive compressée archive.tar.gz
- **gunzip archive.tar.gz** # décompresse archive.tar.gz

3 La gestion des processus

3.1 Les processus

Quand on exécute une commande ou qu'on lance un programme, on crée un processus. Par exemple, sous Windows, quand une application plante, la commande CTRL+ALT+SUPPR nous affiche la liste des processus et permet de supprimer celui qui ne répond pas afin de relancer la machine ou de récupérer la main. Et bien, sous Linux, c'est pareil (disons plutôt que c'est Windows qui fonctionne comme un UNIX)
- **top** # affiche les tâches en cours d'exécution
- **ps -ux** # permet d'afficher les processus en cours d'exécution lancé par l'utilisateur et d'obtenir en particulier leur numéro.
- **ps -aux** # permet d'afficher tous les processus en cours d'exécution
- **kill -9 num_proc** # permet de tuer le processus dont le numéro est num_proc

3.2 Cas des processus d'impression

- **lpr file.eps** # imprime le fichier postscript file.eps sur l'imprimante par défaut
- **lpr -PprinterName file.eps** # imprime le fichier postscript file.eps sur l'imprimante de nom printerName
- **lpq -PprinterName** # affiche l'éta t de l'imprimante printerName et les tâches d'impression en attente ou en cours
- **lprm -PprinterName processNum** # supprime la tâche d'impression processNum sur l'imprimante printerName

Il existe également la commande **a2ps** qui permet d'imprimer n'importe quel type de fichier autre que postscript, avec une syntaxe identique à **lpr**. Toutefois, les évolutions récentes en termes de gestion d'imprimante (CUPS par exemple) permettent d'utiliser la commande **lpr** pour imprimer n'importe quel type de fichier.

4 Communiquer avec l'extérieur

Il existe essentiellement 4 moyens de communications permettant de se connecter à distance sur une autre machine ou sur un autre compte utilisateur :

ftp
telnet
rlogin
ssh

Il existe par ailleurs diverses commandes permettant d'échanger des données entre différentes machines ou comptes d'une même machine. En voici trois :

ftp
rsync
scp

4.1 ftp

La commande **ftp** permet de se connecter à distance sur une machine et d'effectuer des opérations de transferts de fichiers.

- **ftp nom_serveur** # pour se connecter à un serveur ftp. Il va ensuite demander un login et un mot de passe. On est alors dans le mode ftp.
- **cd dir1** # permet d'aller dans un répertoire de la machine destination
- **lcd dir1** # permet d'aller dans un répertoire de la machine locale
- **pwd** # affiche le répertoire courant dans la machine destination
- **lpwd** # affiche le répertoire courant dans la machine locale
- **put file1** # copie le fichier file1 de la machine locale vers la machine destination
- **get file1** # copie le fichier file1 de la machine destination vers la machine locale

4.2 ssh

Dans son utilisation la plus courante, la commande **ssh** permet de se connecter à distance sur une machine pour pouvoir l'utiliser comme si on était directement connecté à cette machine. Très utile donc pour travailler sur le compte du bureau depuis chez soi. Voilà les options les plus courantes :

-t permet de forcer l'allocation d'un terminal.

-X permet d'activer le transfert X11.

-l username pour spécifier un nom d'utilisateur local sur la machine à laquelle on veut se connecter.

De manière concrète, l'utilisation des options -t et -X permet de gérer l'affichage. Ainsi, l'interface d'un programme exécuté sur la machine distante s'affichera comme si on se trouvait sur la machine locale. Il existe certaines configurations du protocole ssh où on utilise l'option -Y, en lieu et place de -t -X. Il y a quand même une différence, mais très subtile et technique, donc je n'en parlerai pas.

De manière générale, la syntaxe d'une commande ssh est donc

ssh -t -X login@sshserver machine.

S'il s'agit d'une connexion entre deux machines d'un même réseau, on peut se contenter de

ssh -t -X login@machine.

4.3 scp et rsync

Voici deux commandes destinées à l'échange de données. La première, **scp** , est la version sécurisée de la commande de copie **cp** . Elle est destinée en pratique à "copier" un fichier. La seconde, **rsync** est

une commande de copie particulière, dans la mesure où il s'agit d'une copie miroir, c'est-à-dire que les fichiers copiées conservent toutes leurs informations (date de création, modification, ...). En pratique, elle est très utilisée pour faire des sauvegardes. Si on fait une copie d'un répertoire d'une machine locale à une machine distante, dans un répertoire contenant une ancienne version, alors cette commande ne transférera que les fichiers ou répertoires modifiés, voire, suivant certaines options, de supprimer de la machine destination les fichiers qui n'existent plus sur la machine locale.

- **scp login@machine :chemin/file1 dir1** # permet de copier le fichier file1 qui se trouve dans le répertoire chemin sur la machine distante, dans le répertoire dir1 de la machine locale
- **scp file1 login@machine :chemin/** # permet de copier le fichier file1 du répertoire courant local dans le répertoire chemin de la machine distante
- **rsync -a -e ssh -v dir1 login@machine :chemin/** # permet de mirrorer le répertoire local dir1 sur la machine distante. L'option -a indique qu'on veut conserver toutes les informations lors du transfert (date de création, date de modification, droits d'accès, ...). L'option -e ssh indique que le transfert de données se fait par protocole ssh. L'option -v indique qu'on veut le mode "verbeux", affichant un maximum d'informations durant le transfert.
- **rsync -a -e ssh -v –delete –exclude=dir1 login@machine :chemin/ .** # permet de mirrorer le répertoire chemin de la machine distante dans le répertoire courant de la machine locale, en supprimant les fichiers locaux n'existants plus dans chemin sur la machine distante (option –delete). Le mirroring ne concerne pas le sous répertoire dir1 (option –exclude=dir1)

Index