PATH Mini-HowTo

Esa Turtiainen (etu@dna.fi) Version 0.4, 15 Novembre 1997

L'objectif de ce HOWTO est de traiter l'utilisation des variables d'environnement sous Unix, et en particulier de la variable PATH. Adaptation française par Mathieu Lafon (Mathieu.Lafon@insalien.org), réalisée le 22 Octobre 1998.

1 Introduction

Ce document aborde les astuces et les problèmes relatifs aux variables d'environnement sous Unix/Linux, et plus spécialement à la variable PATH. PATH est une liste de répertoires dans lesquels le système recherche les commandes à exécuter. Ce document s'appuie sur la distribution Debian Linux 1.3.

Remarque: Ce document est en phase de développement (bêta). Vous pouvez m'envoyer vos commentaires ou vos corrections.

Les commentaires sur la traduction sont à envoyer à Mathieu Lafon (Mathieu.Lafon@insalien.org).

2 Droits d'auteur

3 Généralités

Tous les processus sous Unix possèdent un *environnement*. C'est une liste de variables contenant un nom et une valeur, les deux sous la forme de chaînes (pouvant contenir la majorité des caractères). Tous les processus Unix possèdent un processus parent, celui qui les a créés. Les processus fils héritent de l'environnement de leurs parents. Ils peuvent ensuite y faire quelques modifications avant de le passer à leurs propres processus fils.

Une variable importante de l'environnement est la variable PATH qui se présente sous la forme d'une liste de répertoires séparés par le caractère deux-points (':'). Ces répertoires sont parcourus pour rechercher les commandes. Si vous essayez de lancer la commande `bidule`, tous les répertoires contenus dans PATH seront examinés (dans l'ordre), à la recherche de l'exécutable `bidule` (un fichier avec le bit exécutable positionné). Si un tel fichier est trouvé, il sera exécuté.

Dans ce document, j'utilise le terme de *commande* pour un programme exécutable qui est appelé sans indication de son chemin, utilisant donc le mécanisme de PATH.

Sous Linux, même les appels de bas niveau pour lancer des processus (la famille des `exec`) se basent sur la variable PATH pour trouver les exécutables : vous pouvez donc utiliser le mécanisme de PATH n'importe où, où vous voulez exécuter une commande. Si un appel de `exec` reçoit le nom d'un fichier qui ne contient pas de '/', il cherchera dans la variable d'environnement PATH. Même si cette variable n'existe pas, les répertoires `/bin` et `/usr/bin` seront examinés à la recherche de cette commande.

Pour créer ou modifier l'environnement, on utilisera `export` avec `sh` ou `setenv` avec `csh`. Par exemple :

```
sh:
    export PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:.

csh:
    setenv PATH /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:.
```

Les programmes C peuvent utiliser la fonction `setenv()` pour modifier l'environnement. Perl, quand à lui, conserve l'environnement dans le tableau associatif %ENV, et vous pouvez donc modifier PATH avec :

```
$ENV{PATH}="/bin"
```

La commande `env` est le moyen le plus facile pour connaître les variables de l'environnement courant. Elle peut également être utilisée pour les modifier.

Pour trouver plus d'information sur les commandes d'accès à l'environnement, vous pouvez regarder les pages de manuel de `environ`, `execl`, `setenv`, le fichier info `env`, ainsi que la documentation des shells.

Quand Linux démarre, le premier processus a être lancé est `init`. C'est un processus particulier car il n'a pas de parent. De plus, il s'agit de l'ancêtre de tous les autres processus. Son environnement restera celui des autres processus tant qu'ils ne le modifieront pas. La plupart le modifieront.

Le programme init lance un groupe de processus spécifiés dans le fichier `/etc/inittab`. Ces processus travaillent dans un environnement directement hérité de `init`. Ce sont d'habitude des processus comme `getty`, le programme qui écrit 'login:' à l'écran. Si vous lancez une connexion PPP ici, vous devez savoir que vous travaillez avec l'environnement de init. L'initialisation du système est souvent effectuée par un script lancé à cet endroit. Dans le cas de la Debian 1.3, il s'agit de `/etc/init.d/rc` qui est chargé de lancer à son tour, les scripts d'initialisation.

Le système comprend plusieurs démons qui peuvent ou non utiliser l'environnement par défaut. La plupart de ceux-ci sont lancé par les scripts d'initialisation et possèdent donc l'environnement de init.

Quand un utilisateur se connecte, l'environnement est modifié par les paramètres contenus dans les programmes, les scripts d'initialisation communs à tous, et ceux spécifiques à l'utilisateur. C'est assez compliqué et la situation n'est pas complètement satisfaisante. En effet, le comportement est totalement différent suivant que l'utilisateur se connecte à partir du terminal texte, de `XDM` ou du réseau.

4 init

`init` est le processus parent de tous les autres processus du système. Ceux-ci héritent de son environnement et même de sa variable PATH dans le rare cas où aucun autre PATH n'est indiqué.

Le PATH de init est fixé dans le code source du programme. Il s'agit de :

```
/usr/local/sbin:/sbin:/bin:/usr/sbin:/usr/bin
```

Notez qu'il ne contient pas le répertoire `/usr/local/bin`.

Tous les programmes qui sont lancés à partir de `/etc/inittab` travaillent avec l'environnement de `init`, et en particulier les scripts d'initialisation contenus dans `/etc/init.d` (dans le cas de la Debian 1.3).

Tout ce qui est lancé par les scripts d'initialisation possède par défaut l'environnement de `init`. Par exemple, `syslogd`, `kerneld`, `pppd` (lorsqu'il est lancé au démarrage), `gpm`, et ce qui est le plus important, `lpd` et `inetd` possèdent l'environnement de `init` et ne le modifient pas.

Un certain nombre de programmes sont lancés par les scripts de démarrage mais avec une variable PATH explicitement fixée dans le script. Les exemples de tels programmes sont `atd`, `sendmail`, `apache` et `squid`.

D'autre programmes, par exemple `cron`, sont lancés par les scripts mais modifient totalement la variable PATH.

5 Connexion

Sur un terminal texte, il y a le programme `getty` qui attend le login de l'utilisateur. Il est chargé d'écrire 'login:' et quelques autres messages. Il travaille avec l'environnement de `init`. Lorsque l'utilisateur commence à se connecter au moyen de `getty`, ce dernier invoque le programme `login`. Celui-ci installe alors l'environnement utilisateur et lance le shell.

Le programme login fixe le PATH comme défini dans le fichier `/usr/include/paths.h`.

Il s'agit, pour les utilisateurs normaux (_PATH_DEFPATH) de :

```
/usr/local/bin:/usr/bin:/bin:.
```

Et pour root (_PATH_DEFPATH_ROOT) de :

```
/sbin:/bin:/usr/sbin:/usr/bin
```

Le PATH des utilisateurs normaux ne contient aucun répertoires `sbin`. Cependant, il contient le répertoire courant '.', qui est considéré comme dangereux pour l'utilisateur root. Même `/usr/local/bin` n'est pas disponible pour root.

Le PATH obtenu lors du login est souvent modifié par l'initialisation du shell. Cependant, il est possible d'utiliser d'autres programmes que des shells dans `/etc/passwd`. Par exemple, j'utilise la ligne suivante pour lancer PPP quand je me connecte avec le nom d'utilisateur etu-ppp. Dans ce cas, `pppd` possède exactement le PATH du login.

```
etu-ppp:viYabVlxPwzDl:1000:1000:Esa Turtiainen, PPP:/:/usr/sbin/pppd
```

6 Shells

Les processus utilisateurs sont souvent des processus fils du shell indiqué pour cet utilisateur dans le fichier `/etc/passwd`. Les fichiers d'initialisation de ces shells modifient souvent la variable PATH.

Lors de la connexion, le nom du shell est précédé d'un '-'. Par exemple, dans le cas de `bash`, on aura `-bash`. Cela indique au shell qu'il est en présence d'un login shell et qu'il doit dans ce cas exécuter les fichiers d'initialisation spécifiques à la connexion. Dans le cas contraire, on aura une initialisation plus légère. De plus, le shell détermine s'il est interactif ou non, c'est à dire si les commandes viennent d'un terminal (tty) ou d'un fichier. Cela modifie également l'importance de l'initialisation si bien qu'un shell non interactif et qui n'est pas lancé avec une connexion effectue vraiment très peu d'initialisation (`bash` n'exécute aucune initialisation dans ce cas là).

6.1 bash

Pour un login shell normal, `bash` parcourt le fichier `/etc/profile`, commun à tous, où les variables d'environnement, dont PATH, peuvent être fixées pour les utilisateurs de `bash`. Cependant, ce fichier n'est pas relu lorsque le système se trouve face à un shell non interactif. Le cas le plus important est `rsh`, où la commande est exécutée sur la machine voisine : le fichier `/etc/profile` n'est pas lancé et le PATH provient du démon de `rsh`.

`bash` accepte les arguments `-login` et `-i` qui sont utilisés pour obtenir respectivement un login shell et/ou un shell interactif.

L'utilisateur peut redéfinir les paramètres contenus dans `/etc/profile` en créant un fichier `~/.bash_profile`, `~/.bash_login` ou `~/.profile`. Il faut noter que seul le premier fichier sera exécuté même si cela diffère des habitudes de `csh`. En particulier, `~/.bash_login` ne sera pas forcement exécuté pour un login shell, car si `~/.bash_profile` existe, ce dernier sera prioritaire.

Si `bash` est lancé par `sh` (qui est un lien symbolique sur `bash`), il se comporte comme le Bourne shell original : il ne parcourt que les fichiers `/etc/profile` et `~/.profile` et uniquement dans le cas d'un login shell.

6.2 tcsh

Pour un login shell, `tcsh` exécute dans l'ordre les fichiers suivants :

- `/etc/csh.cshrc`
- `/etc/csh.login`
- `~/.tcshrc`
- `~/.cshrc` (si `~/.tcshrc` n'existe pas)
- `~/.history`
- `~/.login`
- `~/.cshdirs`

Attention. `tcsh` peut être compilé pour exécuter les scripts de connexion (`login`) avant les scripts `cshrc`.

Les shells non interactifs n'exécutent que les scripts `*cshrc`. Les scripts `*login` peuvent être utilisés pour ne fixer le PATH que lors d'une connexion.

7 Modifier l'identité de l'utilisateur

7.1 su

La commande `su` sert à indiquer la nouvelle identité à utiliser (sous réserve de connaître le mot de passe), root étant la valeur par défaut.

Normalement, `su` lance un sous-shell avec la nouvelle identité. Avec l'argument '-' (plus récemment `-l` ou `-login`), `su` lance le shell comme un login shell. Cependant, il n'utilise pas le programme `login` pour cela mais encore un autre PATH intégré au programme pour simuler le login (termes employés dans le code source). Il s'agit de :

pour les utilisateurs normaux :

```
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:.
```

pour l'utilisateur root :

```
/sbin:/bin:/usr/sbin:/usr/bin:/usr/bin/X11:/usr/local/sbin:/usr/local/bin
```

`su` réalise également quelques changements mineurs dans l'environnement.

7.2 sudo

Il y a un groupe de commandes qui permettent une utilisation plus sûr des commandes du super utilisateur. Elles permettent un meilleur suivi (au sens où l'on garde une trace de chaque exécution - NdT), des restrictions sur les utilisateurs et utilisent des mots de passe individuels. La plus utilisée est sûrement `sudo`.

```
$ sudo env
```

Cette commande exécute `env` en tant que super utilisateur (si `sudo` est configuré pour le permettre).

La commande `sudo` a encore une autre approche en ce qui concerne la gestion du PATH. Elle modifie les répertoires où chercher la commande à exécuter pour que le répertoire courant soit toujours le dernier. Cependant, elle ne modifie pas la variable PATH, seulement quelques variables comme SUDO_USER.

8 Serveurs

La majorité des serveurs ne devrait pas lancer n'importe quelle sorte de processus. Pour des raisons de sécurité, leur PATH doit donc être minimal.

La plus grosse exception est l'ensemble des services qui autorisent une connexion sur le système à partir du réseau. Cette section décrit comment se trouve l'environnement dans ces cas précis. En effet, une commande exécuté à distance avec `rsh` aura un PATH différent d'une commande exécuté avec `ssh`. De la même façon, une connexion à l'aide de `rlogin`, `telnet` ou `ssh` est différente.

8.1 inetd

La plupart des serveurs ne possèdent pas de processus chargé d'attendre en permanence l'arrivée d'une requête. Ce travail est laissé à un super serveur (Internet super server), appelé `inetd`. Le programme `inetd` est à l'écoute permanente du réseau et lance le serveur approprié en fonction du port sur lequel arrive la requête. Son comportement est défini dans le fichier `/etc/inetd.conf`.

`inetd` est démarré par les scripts de démarrage du système. Il hérite donc du PATH de `init`. Il ne le modifie pas et tous les serveurs lancés par `inetd` possèdent donc le PATH de `init`. Un exemple de tel serveur est `imapd`, le serveur du protocole IMAP.

D'autre exemples de processus lancés par `inetd` sont `telnetd`, `rlogind`, `talkd`, `ftp`, `popd`, certains serveurs http, etc...

Souvent, l'utilisation de `inetd` est compliquée par l'utilisation du programme tcpd, chargé de lancer le véritable serveur. C'est un programme qui effectue quelques vérifications du point de vue sécurité avant de lancer le véritable serveur. Il ne touche pas au PATH (information non vérifiée).

8.2 rsh

Le démon de `rsh` utilise le PATH défini par _PATH_DEFPATH (`/usr/include/path.h`), c'est à dire, le même que celui utilisé par le programme `login` pour connecter les utilisateurs normaux. L'utilisateur root obtiendra le même PATH que les autres.

En réalité, `rshd` exécute la commande désirée en se servant de la commande suivante :

```
shell -c ligne_de_commande
```

Où `shell` n'est pas un login shell. Il est préférable que tous les shells mentionnés dans `/etc/passwd` prennent en compte l'option `-c` pour pouvoir leur envoyer ce genre de ligne de commande.

8.3 rlogin

`rlogin` invoque login pour effectuer la procédure de connexion. Si vous vous connectez avec `rlogin`, vous aurez le même PATH qu'avec `login`. La plupart des autres façons de se connecter à un ordinateur sous Linux n'utilisent pas `login`. Notez la différence avec `rsh`.

La commande de `login` utilisée est de la forme :

```
login -p -h nom_de_l_hote nom_d_utilisateur
```

L'option `-p` conserve l'environnement à l'exception des variables HOME, PATH, SHELL, TERM, MAIL et LOGNAME. L'option `-h` indique le nom de l'ordinateur sur lequel doit se faire la connexion.

8.4 telnet

Le programme `telnet` est similaire à `rlogin` : il utilise le programme `login` et la ligne de commande utilisée est de la même forme.

8.5 ssh

`ssh` possède sa propre variable PATH, à laquelle il ajoute le répertoire où se trouve `ssh`. Cela implique souvent que le répertoire `/usr/bin` se retrouve en double :

```
/usr/local/bin:/usr/bin:/bin:.:/usr/bin
```

La variable PATH ne contient pas `/usr/bin/X11` et le shell invoqué par `ssh` n'est pas un login shell. Ainsi, la commande

```
ssh hote_distant xterm
```

ne marchera pas et rien de ce qui est contenu dans `/etc/profile` ou `/etc/csh.cshrc` ne pourra changer cela. Vous devrez toujours utiliser des chemins absolus, par exemple `/usr/bin/X11/xterm`.

`ssh` cherche des variables d'environnement de la forme VARIABLE=VALEUR dans le fichier `/etc/environment`. Malheureusement, cela provoque des problèmes avec XFree86.

9 XFree86

9.1 XDM

XDM est la manière la plus courante pour se connecter à partir d'un terminal graphique. Même s'il ressemble à `login`, il se comporte, en interne, d'une manière totalement différente.

Les fichiers de configuration se trouvent dans le répertoire `/etc/X11/xdm` et sont exécutés pendant les différentes étapes de la connexion. Xstartup (et Xstartup_0 pour l'écran 0) contient les commandes à exécuter juste après la connexion. Ces commandes sont lancés en tant que root.

Le PATH qui est utilisé pour les utilisateurs se trouve dans `/etc/X11/xdm/xdm-config`. Ce sont les lignes :

```
DisplayManager*userPath: /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
DisplayManager*systemPath: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11
```

C'est le PATH par défaut pour les utilisateurs normaux (`userPath`), et pour l'utilisateur root (`systemPath`) respectivement. Il est très important que le répertoire `/usr/bin/X11` soit accessible pour les utilisateurs sous X. En effet, si un utilisateur se connecte à une autre machine pour lancer une application X, il faut qu'il aie `/usr/bin/X11` dans son PATH car la machine hôte ne saura pas qu'il dispose d'un terminal X.

Après Xstartup, XDM lance `/etc/X11/Xsession` en tant qu'utilisateur final. La configuration locale est contenue dans le fichier `/etc/environment` qui est parcouru, s'il existe, par Xsession. Xsession étant exécuté par `/bin/sh`, `/etc/environment` doit donc être un script `sh`. Cela interfère avec `ssh` qui suppose que `/etc/environment` est un fichier qui ne contient que des lignes de la forme VARIABLE=VALEUR.

9.2 xterm -ls

Par défaut, le PATH de toutes les commandes lancés à partir des menus du gestionnaire de fenêtre est celui hérité de XDM. Pour en utiliser un autre, il faut le définir explicitement. Pour lancer un terminal X avec un PATH "normal", on doit utiliser des options spéciales. Pour `xterm`, l'option `-ls` (login shell) doit être utilisé pour obtenir un login shell avec le PATH défini dans les fichiers d'initialisation du shell en question.

9.3 Menus et boutons du gestionnaire de fenêtre

Le gestionnaire de fenêtre hérite de l'environnement de XDM. Tous les programmes lancés par lui héritent donc de cet environnement.

L'environnement du shell de l'utilisateur n'affecte pas les programmes qui sont lancés par les menus ou les boutons. Par exemple, si un programme est lancé par un `xterm` (`xterm -ls`), il possède l'environnement par défaut du login shell, par contre s'il est lancé par un menu, il aura l'environnement du gestionnaire de fenêtre.

10 Commandes "à retardement" cron et at

10.1 cron

C'est le programme `cron` qui exécute périodiquement les commandes spécifiées dans `/etc/crontab` et dans les crontabs des utilisateurs. La Debian 1.3 possède en plus un mécanisme pour exécuter les commandes de `/etc/cron.daily`, `/etc/cron.weekly` et `/etc/cron.monthly`, respectivement tous les jours, toutes les semaines et tous les mois.

cron est lancé par les scripts de démarrage mais il change son PATH en une valeur assez étrange :

```
/usr/bin:/binn:/sbin:/bin:/usr/sbin:/usr/bin
```

IL S'AGIT SUREMENT D'UN BOGUE DANS CRON. Il s'agit en fait du PATH de init (/usr/bin:/bin) qui est copié ici, mais sans le 0 terminal (chaîne en convention C - NdT)! Ce bogue n'existe pas sur tous les systèmes.

Dans la crontab, on peut définir un PATH spécifique pour l'exécution des commandes. Pour la Debian 1.3, il s'agit de :

```
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

De cette façon, le PATH de crond n'est jamais utilisé dans les programmes utilisateurs. Tous les scripts de /etc/cron.* obtiennent par défaut le PATH de la crontab. Celui ci est utilisé même si le programme n'est pas exécuté en tant que root.

10.2 at

La commande at est utilisée pour lancer un programme à une heure fixée.

Le programme atd est lancé avec le PATH de init. Cependant, les programmes sont toujours lancés avec l'environnement utilisateur grâce à sh. Les spécificités de sh s'appliquent donc ici. Reportez vous au chapitre sur bash.

11 Quelques exemples

11.1 magicfilter

magicfilter est un outil standard permettant de manipuler les fichiers à destination de l'imprimante. Il analyse le type du fichier à imprimer et lance un filtre approprié pour l'imprimer de la meilleure façon. Les scripts utilisés pour filtrer sont lancés par lpd, lui même lancé par le script /etc/init.d/lpd lancé par init. Le PATH est donc identique à celui de init et ne contient donc pas /usr/bin/X11.

Si vous voulez envoyer des fichier PDF (Portable Data Format) à magicfilter, vous pouvez utiliser /usr/bin/X11/xpdf. Mais vous ne devez pas oublier d'indiquer le chemin absolu. Sinon, magicfilter ne trouvera pas xpdf. La plupart des programmes utilisés avec magicfilter, ne nécessitent pas forcement un chemin explicite car ils se trouvent souvent dans /bin ou /usr/bin.

11.2 Impression à partir d'applications X

Au cas où vous utilisez la variable d'environnement PRINTER pour sélectionner l'imprimante à utiliser, vous devez savoir que dans certains cas, certaines applications X risquent de ne pas la connaître.

Vous vous souvenez sûrement que si la session X a été lancé par XDM, le gestionnaire de fenêtre ne se sert pas de vos scripts de login. Toutes les applications X que vous lancez à partir d'un xterm possèdent donc la variable PRINTER. Par contre, la même application lancée à partir d'un menu ou d'un bouton ne possédera pas cette variable.

Parfois, la variable PRINTER peut être héritée à un niveau encore plus bas. Par exemple, une application auxiliaire de Netscape pourra connaître votre variable PRINTER même si Netscape ne la connaît pas.

12 Questions de sécurité

Le mécanisme de PATH est souvent un gros problème du point de vue sécurité. Utiliser une erreur dans la définition du PATH est une manière fréquente de pirater un système. Il est facile pour un pirate de fabriquer des chevaux de Troie, s'il arrive à forcer root ou un autre utilisateur à exécuter ses propres programmes.

Une erreur fréquente par le passé (?) était de laisser le répertoire courant '.' dans le PATH de l'utilisateur root. Un pirate malveillant peut alors créer son propre programme `'ls'` dans son répertoire. Ensuite, si root fait :

```
# cd ~pirate
# ls
```

il exécute le programme du pirate...

De la même façon, cela s'applique à tous les programmes exécutés par root. Aucun important démon ne devrait exécuter quoi que ce soit qui puisse être modifié par un utilisateur. Dans certains systèmes, `/usr/local/bin` peut contenir des programmes jugés moins sûr, mais le répertoire est retiré du PATH de root. Cependant, si on sait qu'un démon exécute `bidule` avec 'PATH=/usr/local/bin:...', il est possible de tromper le démon en lui faisant exécuter `/usr/local/bin/bidule` à la place de `/bin/bidule`. Dans ce cas, n'importe qui pouvant écrire dans `/usr/local/bin` peut sûrement pirater le système.

Il est donc très important de faire attention à l'ordre dans lequel les répertoires sont placés dans le PATH. Si `/usr/local/bin` se trouve avant `/bin`, il y a un risque. Alors que s'il se trouve après, il est impossible de lancer la commande modifiée `/usr/local/bin/bidule` à la place de `/bin/bidule`.

Sous Linux, vous devez vous souvenir que la recherche dans le PATH est faîte dans tous les mécanismes d'appels du système d'exploitation. N'importe où, où le chemin d'un exécutable est donné, vous pouvez utiliser le nom de la commande seul qui sera alors cherchée au moins dans `/bin` et `/usr/bin`, et vraisemblablement dans beaucoup d'autres endroits.

13 Comment résoudre les problèmes ?

La commande la plus simple pour avoir accès à l'environnement est `/usr/bin/env`.

Il est egalement possible d'utiliser le répertoire `/proc` pour trouver le PATH de n'importe quel programme. Vous devez d'abord connaître le numéro de processus du programme. Utilisez la commande `ps` pour l'obtenir. Par exemple, si `xterm` est le processus numéro 1088, vous pouvez voir son environnement avec :

```
# more /proc/1088/environ
```

Cela ne marche pas avec des processus comme `xdm`. Pour accéder à l'environnement d'un processus du système ou d'un autre utilisateur, vous devez être root.

Pour deboguer Netscape, vous pouvez créer le script suivant :

```
$ cat > /tmp/test
#!/bin/sh
/usr/bin/env > /tmp/env
^d
$ chmod +x /tmp/test
```

Ensuite, arrangez vous pour que votre programme soit appelé à la place d'une application auxiliaire, par exemple RealAudio (`audio/x-pn-realaudio`). Lorsque vous essayerez d'accéder à un lien RealAudio (quelque chose comme `http://www.realaudio.com/showcase`), Netscape lancera votre programme factice et sauvera l'environnement dans `/tmp/env`.

14 Méthodes pour que tous les utilisateurs aient le même PATH

Le réglage le plus important est à faire dans les fichiers commun d'initialisation des logins shells : `/etc/csh.login` pour `tcsh` et `/etc/profile` pour `bash`.

Ceux qui n'obtiennent pas le bon PATH à partir de ces fichiers sont : `rsh`, `ssh`, les éléments des menus du gestionnaire de fenêtres sous X ne lançant pas explicitement de login shell, les commandes lancés à partir de `inittab`, les travaux de `cron`, les travaux des démons comme `magicfilter` (lancé par `lprd`), les scripts CGI (WWW), etc...

Si le PATH est fixé dans `/etc/csh.cshrc`, il sera utilisé si `rsh` ou `ssh` lance des commandes sur une machine distante où l'utilisateur utilise `tcsh/csh`. Par contre, il n'est pas possible de régler le PATH si l'utilisateur utilise `bash/sh`. Voici une méthode pour ne garder le PATH que dans un seul fichier, par exemple `/etc/environnement-commun`, dans lequel on écrit :

```
${EXPORT}PATH${EQ}/bin:/usr/bin:/sbin:/usr/sbin:/usr/bin/X11:/usr/local/bin:/usr/games:.
```

On peut ensuite l'utiliser à partir de `/etc/csh.login` (pour `tcsh` et `csh`)

```
set EQ=" " set EXPORT="setenv "; source /etc/environnement-commun
```

A partir de `/etc/profile` (pour `bash`, mais pas pour le vrai `sh`)

```
EQ='=' EXPORT="export " . /etc/environnement-commun
```

Et à partir de `/etc/environment` (pour XDM)

```
EQ='=' EXPORT="export " . /etc/environnement-commun
```

Cette méthode marchera la plupart du temps, sauf que `ssh` se plaindra des lignes contenues dans `/etc/environment` (ainsi que des variables EQ et EXPORT). De plus, `rsh` n'aura toujours pas le bon PATH s'il passe par `bash`.

15 Remerciements

Une des raisons pour commencer l'écriture de ce document a été la grosse frustration de Ari Mujunen. Juha Takala m'a donné de précieux commentaires.