Module administration système et réseau GNU/Linux

Le shell Bash

Sommaire

Pré-requis

- Avoir une VM avec une distribution GNU/Linux

Objectifs

- Connaître les fichiers définissant l'environnement du shell Bash

A. Introduction

Shell veut dire *Coquille*, qui entoure le noyau. C'est un interpréteur de commandes qu'on utilise pour lancer des commandes ou programmer une suite de commandes. L'utilisateur discute avec le Shell, qui discute avec le noyau, qui à son tour discute avec le matériel.

Originellement le shell est utilisé sous Unix, il s'est répandu depuis avec différentes versions, la forme la plus simple est sh.

Les versions connues :

- sh : shell Bourne
- ksh : korn shell
- Csh : Shell syntaxe du C
- Tcsh : Csh amélioré
- **Bash** : Bourne Again Shell ← **que l'on retrouve par défaut sur les distributions majeures**
- Zsh : le petit dernier

Les interpréteurs de commandes (ou shells), qui peuvent être le premier contact de l'utilisateur avec l'ordinateur, doivent être assez conviviaux. La plupart utilisent des scripts d'initialisation permettant de configurer leur comportement (complétion automatique, texte d'invite, etc.).

Pour lister les shells disponibles sur le système :

```
$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
/bin/zsh
/usr/bin/zsh
```

B. Environnement shell

1. Généralité

Le programme shell **/bin/bash** utilise une collection de fichiers de démarrage pour aider à la création d'un environnement de travail. Chaque fichier a une utilisation spécifique et peut affecter différemment la connexion et les environnements interactifs.

Un shell interactif de connexion est lancé après une identification positive (demande nom d'utilisateur et mot de passe) avec /bin/login (depuis une console virtuelle ou depuis une connexion distante SSH par exemple) en lisant le fichier **/etc/passwd**.

Un shell interactif, sans login, est lancé soit à la ligne de commande (c'est-à-dire

/bin/bash) soit le plus couramment depuis une application comme gnome-terminal ou terminator dans un environnement graphique.

Un shell non-interactif est présent habituellement lorsqu'un script shell tourne. Il est non interactif parce qu'il exécute un script et n'attend pas d'entrée utilisateur entre les commandes.

Les fichiers suivants sont nécessaires pour s'assurer que l'environnement correct est lu pour chacune des façons dont le shell peut être appelé: **/etc/profile**, **/etc/bashrc**, **~/.bash_profile** et **~/.bashrc**.

Note : même si le principe est le même sur toutes les distributions, ces fichiers peuvent différer dans leur nom et dans leur contenu suivant la distribution employées. C'est notamment le cas entre Red Hat (et ses dérivées) et Debian (et ses dérivées).

2. /etc/profile

Ce fichier est commun à plusieurs shells (sh,ksh,bsh,bash..). C'est un script shell qui est exécuté lors de la connexion à un terminal texte. Il contient des variables d'environnement de base (que nous verrons dans ce cours) de tous les processus et seul l'administrateur système peut le modifier. On ne doit pas y mettre de variables spécifiques bash - ça, c'est dans le bashrc qu'on les mets.

Ce script est interprété lors de la connexion de l'utilisateur.

3. ~/.bash_profile, ~/.bash_login, ~/.profile

Après le exécution de /etc/profile, Bash recherche un de ces fichiers dans cet ordre :

1. ~/,bash_profile,
2. ~/.bash_login,
3. ~/.profile.

Le fichier trouvé, il exécute les commandes contenues dans celui-ci.

Ce fichier a la même fonction que /etc/profile, si ce n'est qu'il peut être modifié par l'utilisateur pour changer son propre environnement.

Comme /etc/profile, celui-ci n'est exécuté qu'à la connexion ; les modifications ne sont prises en charge qu'en se déconnectant puis se reconnectant.

4. ~/.bashrc

Le fichier ~/.profile n'est exécuté qu'à la connexion. Si un utilisateur s'est connecté dans un environnement graphique et qu'il lance un terminal, ce fichier ne sera pas lu. Cela pose des problèmes pour les éléments définis dans ~/.profile.

Le fichier ~/.bashrc est alors utilisé à cette fin, puisqu'il est interprété au démarrage de chaque nouveau shell Bash intéractif.

En bash, **de manière générale,** le shell de connexion exécute en premier le script **/etc/**

profile, puis le script **~/.profile** et enfin le script **~/.bashrc**.

Dans tous le cas, c'est dans ce dernier fichier, que l'on peut définir de nouveaux alias, exporter automatiquement de nouvelles variables, paramétrer l'auto-complétion, définir les variables PS1 et PS2, paramétrer le nombre de commandes à historiser, etc etc.

5. /etc/bashrc

Il a la même fonction que le fichier précédent mais à l'échelle du système et n'est modifiable que par l'administrateur. Il est invoqué par le fichier ~/.bashrc.

6. ~/.bash_logout

Ce script n'est utilisé qu'à la déconnexion de l'utilisateur. On y inclut par exemple des commandes de nettoyage automatique (suppression des fichiers de travail temporaires ou effacement de l' affichage).

7. Les variables d'environnements

Il arrive parfois que certaines informations doivent être mise à disposition d'un grand nombre de programme.

Par exemple, nous avons vu que lorsque vous exécutez une commande (telle qu'un simple « ls »), c'est en réalité un programme qui est exécuté ; or ce programme est contenu dans un fichier, qui doit être lu pour que le système puisse exécuter le programme.

Mais comment le système fait-il pour trouver le fichier en question ?

L'information indiquant à quels endroits chercher les fichiers contenant les programmes des diverses commandes doit donc être aisément accessible, et comme certaines commandes en appelle d'autres, cette information doit également être lisible par tout le monde.

Ce genre d'information largement diffusée dans le système est contenu dans ce que l'on appelle des ***variables d'environnement***. Une variable est une zone de la mémoire, identifiée par une étiquette (un nom) et contenant quelque chose (un nombre, un mot...) Ces variables sont fixées, certaines au démarrage du système, d'autres lors de la connexion d'un utilisateur. Examinons l'une d'entre elle. Exécutez la commande suivante :

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/
snap/bin
```

La commande `echo` permet d'afficher quelque chose. Dans la commande précédente, nous lui demandons d'afficher le contenu d'une variable dont le nom est `PATH`. Notez l'utilisation du symbole dollar ('$') devant le nom de la variable, nécessaire lorsque l'on souhaite accéder au contenu de la variable.

Le résultat de la commande **echo $PATH** est manifestement là une suite de répertoires, donnés par leur chemin absolu, et séparés par le caractère deux-points (' **:** '). Nous avons là précisément la réponse au problème posé au début de ce paragraphe : la variable d'environnement `PATH` contient une liste de répertoires dans lesquels chercher les fichiers

contenant les programmes des commandes que l'on souhaite exécuter.

Un simple which ls nous donne le chemin de la commande dont le répertoire parent est bien présent dans la variable $PATH :

```
$ which ls
/bin/ls
```

Les variables d'environnements ne se limitent pas à PATH. Voici comment les lister :

```
$ env
```

Il existe aussi les variables du shell que l'on peut obtenir avec la commande **set** (attention la liste est longue).

On peut jouer sur ces variables en les paramétrant dans le fichier ~/.bashrc. Par exemple si vous souhaitez augmenter le nombre de lignes dans le fichier ~/.bash_history (le fichier qui est appelé par la commande history), il vous faut trouver la ligne HISTSIZE et modifier la valeur. Afin que les changements du fichier ~./bashrc soient pris en compte, il vous suffit de saisir :

```
$ source ~/.bashrc
```

Vous pouvez modifier le prompt PS1 qui ressemble à ça :

```
stag@formationldnr:~$
```

Pour obtenir cela, la variable est celle-ci :

```
PS1='\[\e]0;\u@\h:  \w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\
[\033[01;34m\]\w\[\033[00m\]\$ '
```

Explications :

- pour la ligne grisée, c'est spécifique à Debian et ses dérivées. Pour faire simple, on peut par le biais du chroot, faire tourner une Debian dans une Debian. La ligne grisée permet donc d'indiquer si nous somme sur le système ou bien dans un chroot.
- Ce qui est en noir, c'est ce qui définit les informations et la mise en forme de votre prompt.

\[\033[01;32m\]\u@\h\[\033[00m\]
\[\033 : Ouverture de la balise pour définir la couleur et la mise en forme du texte
[01;32m\] : 01 pour dire que le texte suivant est en gras et 32 pour indiquer que sa couleur sera verte.
\u : l'utilisateur courant
\h : le nom de l'hôte
@ : c'est pour décorer
\w : le répertoire courant
$: le symbole qui indique que l'utilisateur est sans privilèges. Si nous étions sur le terminal en tant que root, nous choisirions #.

Bon ça fait un peu compliqué comme ça au premier abord, donc si vous souhaitez personnaliser votre prompt rendez-vous sur http://bashrcgenerator.com/

Pour tester votre PS1 temporairement sans modifier le fichier ~/.bashrc, il vous suffira de copier la ligne générer sur le site puis dans un terminal :

```
export ligne_copiée
```

Vous mettrez le code obtenu dans le fichier ~/.bashrc. Il vous faudra commenter la ligne suivante déjà présente dans le fichier en mettant un dièse # devant :

```
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
```

N'oubliez surtout pas de faire un backup du fichier avant de le modifier.

Vous trouverez ici une liste de variables du shell dont certaines seront abordées dans le cours sur le scripting.

Une astuce qui peut vous servir pour certaines commandes répétitives dont certaines sont déjà définies dans la majeure parties des distributions : les **alias**.

Pour lister les alias définies :

```
$ alias
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$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

On connaît déjà la plupart de ses commandes. Si on saisit « ll », le résultat sera équivalent à la commande « ls -alF ». En fait les alias sont des sortes de raccourcis.

Pour les paramétrer, soit on passe par ~/.bashrc soit dans ~/.bash_aliases dans le cas d'Ubuntu (qui est appelé par ~/.bashrc).

Par exemple si je veux que la commande « rm » soit interactive sans avoir besoin de le préciser (en me demandant une confirmation de suppression par exemple), il suffit d'ajouter dans un des deux fichiers cités plus haut :

```
alias rm='rm -i'
```

Ainsi lorsque je ferai un rm sur un fichier ou un répertoire, il me sera demandé une confirmation sans besoin de préciser l'option -i. Un autre exemple :

```
alias Doc='cd ~/Documents'
```

Pour que les alias soient pris en compte :

```
$ source ~/.bashrc
```

Pour aller dans le répertoire Documents de mon répertoire personnel, je n'aurai qu'à saisir :

```
$ Doc
```

On évitera de faire des raccourcis du genre :

```
alias ll='rm -rf'
```

On évitera aussi de donner des noms d'alias déjà pris par des commandes.

Les alias sont très pratiques mais on les réservera pour des commandes courtes. Pour le reste, on privilégiera les scripts.

Vous trouverez d'autres exemples sur le web.