BASH ou le Bourne Again Shell

Qu'est-ce qu'un shell ?

Ouais, c'est quoi donc le shell ? Et bien, le shell est fondamentalement un environnement utilisateur de lignes de commande. En substance, il s'agit d'une application qui s'exécute lorsque l'utilisateur se connecte et lui permet d'exécuter des applications supplémentaires. À certains égards, le shell est très similaire à une interface utilisateur graphique, en ce sens qu'il fournit un cadre pour l'exécution des commandes et lancer des programmes. Il existe une panoplie de shells incluse avec une installation complète de Slackware. Toutefois dans ce livre, nous allons seulement discuter de bash (1), le Bourne Again Shell. Les utilisateurs avancés peuvent envisager d'utiliser le puissant zsh (1), et les utilisateurs familiers avec les systèmes UNIX plus anciens peuvent apprécier ksh. Le masochiste pourrait vraiment choisir le csh, mais les nouveaux utilisateurs devraient s'en tenir à bash.

Variables d'environnement

Tous les shells facilitent l'exécution de certaines tâches pour l'utilisateur en permettant de garder une trace à travers les variables d'environnement. Une variable d'environnement est tout simplement un nom court pour certains bits d'information que l'utilisateur souhaite stocker et utiliser plus tard. Par exemple, la variable d'environnement PS1 spécifie à ***bash*** comment formater son invite de commande. D'autres variables peuvent servir à définir le fonctionnement de certaines applications. Par exemple, la variable LESSOPEN indique à ***less*** d'exécuter le très utile pré-processeur `lesspipe.sh` dont nous avons discuté, et LS_OPTIONS active les options de couleur d'affichage pour ***ls***.

Il est très aisé de définir votre propre variable d'environnement. ***bash*** contient deux fonctions natives permettant de gérer cela : ***set*** et ***export***. En outre, une variable d'environnement peut être supprimée en utilisant ***unset***. (Pas de panique si vous supprimez une variable d'environnement par mégarde. La déconnexion du terminal suivie d'une reconnexion restaurera les variables d'environnement par defaut.) Une variable d'environnement peut être référencée en plaçant tout simplement le symbole ($) devant la variable.

```
darkstar:~$ set FOO=bar
darkstar:~$ echo $FOO
bar
```

La différence principale entre ***set*** et ***export*** est que ***export*** exportera (naturellement) la variable dans tous les sous-shells subséquents. (Un sous-shell est tout simplement un shell s'exécutant à l'intérieur d'un autre shell, shell parent.) Cette différence est notable lorsque vous utilisez la variable PS1 qui contrôle l'invite de commande ***bash***.

```
darkstar:~$ set PS1='FOO '
darkstar:~$ export PS1='FOO '
FOO
```

Une des variables d'environnement très importante dans ***bash*** et d'autres shells est PATH. PATH est

tout simplement la liste des répertoires consultés lorsqu'une application a besoin de s'exécuter. Par exemple, ***top***(1) se trouve dans ***/usr/bin/top***. Vous pouvez l'exécuter simplement en spécifiant son chemin d'accès complet à elle, mais si `/usr/bin` est dans votre variable PATH, ***bash*** vérifiera ce répertoire si vous ne spécifiez pas de chemin d'accès complet. Ce comportement est bien visible lorsque vous essayez d'exécuter un programme qui n'est pas dans votre PATH d'utilisateur normal, par exemple, ***ifconfig***(8).

```
darkstar:~$ ifconfig
bash: ifconfig: command not found
darkstar:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/games:/opt/www/htdig/bin:.
```

Ci-dessus, vous voyez un PATH typique pour un utilisateur normal. Vous pouvez le modifier comme n'importe quelle autre variable d'environnement. Cependant, si vous vous connectez en tant que root, vous verrez que le PATH de root est différent.

```
darkstar:~$ su -
Password:
darkstar:~# echo $PATH
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:
/usr/games:/opt/www/htdig/bin
```

Les caractères génériques

Les caractères génériques sont des caractères spéciaux qui indiquent au shell de générer des correspondances par rapport à certains critères. Si vous avez une expérience avec MS-DOS, vous reconnaîtrez * comme un caractère générique qui correspond à quoi que ce soit. ***bash*** utilise ce caractère générique et plusieurs autres pour vous permettre de facilement définir exactement ce que vous voulez faire.

Le premier et le plus commun de cette liste de caractères génériques est, bien sûr, *. L'astérisque correspond à n'importe quel caractère ou combinaison de caractères, y compris aucun. Ainsi b* correspond à tout fichier nommé b, ba, bab, babc, bcdb, et ainsi de suite. Un peu moins commun est le caractère ?. Ce caractère correspond à une instance de tout caractère, de sorte que b? correspondrait a bas et bb, mais pas a b ou bab.

```
darkstar:~$ touch b ba bab
darkstar:~$ ls *
b ba bab
darkstar:~$ ls b?
ba
```

Non, le fun ne s'arrête pas là ! En plus de ces deux, nous avons aussi la paire de crochets *"[ ]"* qui nous permet d'affiner le critère de correspondance. Chaque fois que ***bash*** rencontre la paire de crochets, il remplace son contenu. Toute combinaison de lettres ou de chiffres peut être spécifiée dans a paire de crochets tant qu'ils sont séparées par des virgules. En outre, des champs de chiffres et de lettres peuvent être aussi ainsi spécifiés. Ceci est probablement mieux illustré par un exemple.

```
darkstar:~$ ls a[1-4,9]
```

```
a1 a2 a3 a4 a9
```

Comme Linux est sensible à la casse des lettres, les majuscules et les minuscules sont traités différemment. Toutes les lettres majuscules précèdent les lettres minuscules en ordre *“alphabétique”*. De ce fait, lors de l'utilisation des champs de lettres majuscules et minuscules, assurez-vous de ne pas vous tromper.

```
darkstar:~$ ls 1[W-b]
1W 1X 1Y 1Z 1a 1b
darkstar:~$ ls 1[w-B]
/bin/ls: cannot access 1[b-W]: No such file or directory
```

Dans le deuxième exemple, 1[b-W] n'est pas un champ valide. Le shell le traite donc comme un nom de fichier, et comme ce ficher n'existe pas, ***ls*** vous le dit.

La touche de tabulation

Vous pensez toujours pense que l'usage des caractères génériques est ardu ? Vous avez raison. Il existe un moyen encore plus facile lorsque vous manipulez des noms de fichiers longs : la correspondance automatique par la touche de tabulation. La touche de tabulation vous permet de taper juste assez du nom de fichier pour l'identifier de manière unique, puis d'appuyer sur TAB key, et ***bash*** complétera le reste pour vous. Même si vous n'avez pas tapé assez de texte permettant d'identifier un nom de fichier unique, le shell va remplir autant que possible pour vous. Taper TAB une seconde fois affichera la liste de toutes les correspondances possibles.

Redirection d'entrée et de sortie

L'une des caractéristiques de Linux et d'autres systèmes d'exploitation dérivant d'Unix est le nombre de petites applications relativement simples et la possibilité de les empiler pour créer des systèmes complexes. Ceci est réalisé en redirigeant la sortie d'un programme vers un autre, ou en obtenant une entrée à partir d'un fichier ou d'un second programme.

Pour commencer, nous allons vous montrer comment rediriger la sortie d'un programme dans un fichier. Cela se fait facilement avec le caractère '>'. Quand ***bash*** voit le caractère '>', il redirige tout sur la sortie standard (également connu sous l’appellation stdout) vers n'importe quel nom de fichier qui suit.

```
darkstar:~$ echo foo
foo
darkstar:~$ echo foo > /tmp/bar
darkstar:~$ cat /tmp/bar
foo
```

Cet exemple montre ce que ***echo*** ferait si stdout n’était pas redirigé vers un fichier, et ensuite ce qui se passe lorsqu'il est redirigé vers le fichier `/tmp/bar`. Si `/tmp/bar` n'existe pas, il est crée et la sortir de ***echo*** y est stockée. Si `/tmp/bar` existait, son contenu est écrasé. Ceci pourrait ne pas être la meilleure idée si vous voulez garder ce contenu en place. Heureusement, ***bash*** supporte “»” qui va

ajouter la sortie au contenu initial du fichier.

```
darkstar:~$ echo foo
foo
darkstar:~$ echo foo > /tmp/bar
darkstar:~$ cat /tmp/bar
foo
darkstar:~$ echo foo2 >> /tmp/bar
darkstar:~$ cat /tmp/bar
foo
foo2
```

Vous pouvez également rediriger la sortie d'erreur standard (ou stderr) vers un fichier. Ceci est légèrement différent en ce que vous devez utiliser '2>'au lieu de simplement utiliser '>'. (Puisque ***bash*** peut rediriger une entrée, stdout et stderr doivent être identifiables de manière unique. 0 est l'entrée, 1 est stdout, et 2 est stderr. À moins que l'un d'eux est spécifié, ***bash*** estimera que vous voulez redirigez seulement stdout chaque fois que vous utiliserez '>'. 1> aurait aussi bien marché.)

```
darkstar:~$ rm bar
rm: cannot remove `bar': No such file or directory
darkstar:~$ rm bar 2> /tmp/foo
darkstar:~$ cat /tmp/foo
rm: cannot remove `bar': No such file or directory
```

Vous pouvez également rediriger l'entrée standard (connu sous le nom de stdin) avec le caractère '<'. Toutefois, cela n'est pas souvent utilisé.

```
darkstar:~$ fromdos < dosfile
```

Enfin, vous pouvez rediriger la sortie d'un programme en tant qu'entrée pour un autre programme. C'est peut-être la fonctionnalité la plus utile de ***bash*** et d'autres shells. Pour ce faire, il suffit d'utiliser le caractère '|'. (Ce caractère est appelé 'pipe'.)

```
darkstar:~$ ps auxw | grep getty
root      2632  0.0  0.0   1656   532 tty2     Ss+  Feb21   0:00
/sbin/agetty 38400 tty2 linux
root      3199  0.0  0.0   1656   528 tty3     Ss+  Feb15   0:00
/sbin/agetty 38400 tty3 linux
root      3200  0.0  0.0   1656   532 tty4     Ss+  Feb15   0:00
/sbin/agetty 38400 tty4 linux
root      3201  0.0  0.0   1656   532 tty5     Ss+  Feb15   0:00
/sbin/agetty 38400 tty5 linux
root      3202  0.0  0.0   1660   536 tty6     Ss+  Feb15   0:00
/sbin/agetty 38400 tty6 linux
```

Gestion de tâches

bash dispose d'une autre fonctionnalité intéressante, la capacité de suspendre et reprendre des tâches. Cela vous permet de suspendre temporairement un processus en cours, accomplir une autre

tâche, puis la reprendre ou le cas échéant la faire tourner en arrière-plan. En appuyant sur CTRL+z, ***bash*** va suspendre le processus en cours et vous revenez à l'invite. Vous pouvez revenir à ce processus plus tard. En outre, vous pouvez suspendre de la meme facon plusieurs processus et cela indéfiniment. La commande native ***jobs*** permet d'afficher une liste des tâches suspendues.

```
darkstar:~$ jobs
[1]-  Stopped                 vi TODO
[2]+  Stopped                 vi chapter_05.xml
```

Pour revenir à une tâche suspendue, exécutez la commande native ***fg*** pour pouvoir amener la tâche la plus récemment suspendue à l'avant-plan. Si vous avez plusieurs tâche suspendues, vous pouvez ussi bien spécifier un nombre afin d'amener l'un d'eux à l'avant-plan.

```
darkstar:~$ fg # "vi TODO"
darkstar:~$ fg 1 # "vi chapter_05.xml"
```

Vous pouvez également envoyer une tâche en arrière-plan en utilisant (surprise) ***bg***. Cela permettra la poursuite de l'exécution du processus sans garder le contrôle de votre shell. Vous pouvez la ramener au premier plan avec ***fg*** de la même façon que les tâches suspendues.

Les terminaux

Slackware Linux et d'autres systèmes d'exploitation de type UNIX permettent aux utilisateurs d'interagir avec eux de nombreuses façons, mais la plus commune, et sans doute la plus utile, est le terminal. Autrefois, les terminaux étaient des claviers et des moniteurs (parfois même des souris) câblés dans un ordinateur central ou serveur via des connexions série. Aujourd'hui, cependant, la plupart des terminaux sont virtuels, c'est à dire qu'ils n'existent que dans le logiciel. Les terminaux virtuels permettent aux utilisateurs de se connecter à l'ordinateur, sans nécessiter de matériel coûteux et souvent incompatibles. Au contraire, les utilisateurs ont seulement besoin d'exécuter le logiciel leur offrant, en en général, un terminal virtuel hautement personnalisable.

Les terminaux virtuels les plus communs (chaque machine Slackware Linux en a au moins un) sont les *gettys*. ***agetty***(8) lance six instances par défaut sur Slackware, et permet aux utilisateurs locaux (ceux qui ne peuvent physiquement s'asseoir en face de l'ordinateur et taper sur le clavier) de se connecter et d'exécuter des applications. Chacun de ces gettys est disponible sur des périphériques tty différentes qui sont accessibles séparément en appuyant sur la touche ALT et l'une des touches de fonction F1 à F6. L'utilisation de ces gettys vous permet de vous connecter plusieurs fois, en tant que différents utilisateurs si vous le souhaitez, et d'exécuter simultanément des applications dans les shells de ces utilisateurs. Ceci est le plus souvent effectué avec des serveurs qui n'ont pas installé ***X***, mais peut être fait sur n'importe quelle machine.

Sur les ordinateurs de bureau, ordinateurs portables et autres postes de travail où l'utilisateur préfère une interface graphique fournie par ***X***, la plupart des terminaux sont graphiques. Slackware comprend de nombreux différents terminaux graphiques, mais les plus couramment utilisés sont ***konsole*** de KDE, ***Terminal***(1) de XFCE et le vieux de la vieille xterm(1). Si vous utilisez une interface graphique, vérifiez vos barres d'outils ou les menus. Chaque environnement de bureau ou gestionnaire de fenêtres possède un terminal virtuel (souvent appelé émulateur de terminal), et ils sont tous étiquetés différemment. Ccependant, vous les trouverez en général dans le sous-menu *"System"* dans les environnements de bureau. Ils vous donneront accès a un terminal graphique et vous

permettront d'exécuter automatiquement votre shell par défaut.

Personalisation

Vous devriez être a présent assez familier avec bash et vous pouvez avoir même remarqué quelques comportements inhabituels Par exemple, lorsque vous vous connectez à la console, vous êtes présenté avec une invite qui ressemble un peu à ceci.

```
alan@darkstar:~$
```

Cependant, parfois, vous verrez une invite beaucoup moins utile comme celui-ci.

```
bash-3.1$
```

La cause ici est une variable d'environnement spéciale qui contrôle l'invite bash. Quelques shells sont considérés comme des *“login”*“ shells et d'autres sont des shells *“interactifs”*, et les deux types lisent différents fichiers de configuration au démarrage. Les *“login”*“ shells lisent `/etc/profile` et `~/.bash_profile` lorsqu'ils sont exécutés. Les shells *“interactifs”* lisent `~/.bashrc`. Cela comporte certains avantages pour les utilisateurs avancés, mais est une nuisance commune pour de nombreux nouveaux utilisateurs qui veulent le même environnement chaque fois qu'ils exécutent bash et ne se soucient pas de la différence entre les login shells et les shells interactifs. Si cela est votre cas, il suffit d'éditer votre propre fichier ~/.bashrc et ajouter les lignes suivantes. (Pour plus d'informations sur les différents fichiers de configuration utilisés, lisez la section INVOCATION de la page ***bash***).

```
# ~/.bashrc
. /etc/profile
. ~/.bash_profile
```

Lorsque vous utilisez la configuration montrée plus haut, tous vos login shells et shells interactifs auront les mêmes paramètres d'environnement et se comporteront de façon identique. Maintenant, quand on veut personnaliser un shell, nous n'avons plus qu'à éditer le fichier `~/.bash_profile` pour les changements spécifiques à l'utilisateur et `/etc/profile` pour les paramètres globaux. Commençons par configurer l'invite de commande.

Les invites de commande ***bash*** sont de toutes formes, couleurs et tailles, et chaque utilisateur dispose de ses propres préférences. Personnellement, je préfère les invites courtes et simples qui prennent un minimum d'espace, mais j'ai vu et utilisé des invites mutli-ligne à plusieurs reprises. Un ami à moi a même inclus de l'ASCII-art dans son invite bash. Pour modifier votre invite de commande, il vous suffit de modifier votre variable PS1. Par défaut, Slackware tente de configurer votre variable PS1 ainsi :

```
darkstar:~$ echo $PS1
\u@\h:\w\$
```

Oui, ce petit bout de drôles de figures contrôle votre invite de commande bash. Fondamentalement, tous les caractères de la variable PS1 sont incluses dans l'invite, sauf s'il s'agit d'un caractère échappé par un \ ce qui indique à ***bash*** de l'interpréter. Il existe différentes séquences d'échappement et nous ne pouvons pas toutes les discuter, mais je vais expliquer celles-ci. Les premier *“\u”* représente le nom de l'utilisateur actuel. *“\h”* est le nom de la machine auquel le

terminal est attaché. “*\w*” est le répertoire de travail courant, et “*\$*” affiche soit le signe # ou $, en fonction des privilèges de l'utilisateur actuel (root ou non). Une liste complète de toutes les séquences d'échappement rapides est répertoriée dans la page de manuel de ***bash*** sous la section PROMPTING.

Comme nous nous sommes donnés tout ce mal pour discuter de l'invite par défaut, j'ai pensé prendre le temps de vous montrer quelques d'invite d commande et les valeurs de la variables PS1 nécessaires pour les utiliser.

```
Wed Jan 14 12:08 AM
alan@raven:~$ echo $PS1
\d \@\n\u@\h:\w$
HOST: raven - JOBS: 0 - TTY: 3
alan@~/Desktop/sb_3.0:$ echo $PS1
HOST: \H - JOBS: \j - TTY: \l\n\u@\w:\$
```

Pour encore plus d'informations sur la configuration de votre invite de commande bash, y compris les informations sur la configuration d'invites de couleur, reportez-vous à `/usr/doc/Linux-HOWTOs/Bash-Prompt-HOWTO`. Après avoir consulté ce document, vous aurez une idée du potentiel de votre invite de commande ***bash***. Une fois, j'ai même eu une invite qui me donnait des informations météorologiques mises à jour telles que la température et la pression barométrique !

Navigation

Chapitre précédent : L'invite de commandes (shell)

Chapitre suivant : Contrôle des processus

Sources

- Source originale : http://www.slackbook.org/beta

- Publication initiale d'Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson
- Traduction initiale de escaflown

slackbook, bash, task management, terminals

From:
https://docs.slackware.com/ - **SlackDocs**

Permanent link:
https://docs.slackware.com/fr:slackbook:bash

Last update: **2016/03/08 11:23 (UTC)**