Chapitre 03 : Configuration du Shell

3.1 Présentation

Les variables shell sont un composant clé du shell Bash. Ces variables sont critiques car elles stockent des informations système vitales et modifient le comportement du shell Bash, ainsi que de nombreuses commandes. Ce chapitre expliquera en détail ce que sont les variables du shell et comment elles peuvent être utilisées pour configurer le shell.

Vous apprendrez comment la variable **PATH** affecte la façon dont les commandes sont exécutées et comment d'autres variables affectent votre capacité à utiliser l'historique de vos commandes. Vous verrez également comment utiliser les fichiers d'initialisation pour rendre les variables shell persistantes, afin qu'elles soient créées à chaque fois que vous vous connectez au système.

De plus, ce chapitre expliquera comment utiliser l'historique des commandes, et les fichiers utilisés pour stocker et configurer l'historique des commandes.

3.2 Variables du shell

Une ***variable*** est un nom ou un identifiant auquel une valeur peut être attribuée. Le shell et d'autres commandes lisent les valeurs de ces variables, ce qui peut entraîner un comportement modifié en fonction du contenu (valeur) de la variable.

Les variables contiennent généralement une seule valeur, comme **0** ou **bob**. Certains peuvent contenir plusieurs valeurs séparées par des espaces comme **Joe Brown** ou par d'autres caractères, tels que des deux-points ; **/usr/bin:/usr/sbin:/bin:/usr/bin:/home/joe/bin**.

Vous attribuez une valeur à une variable en tapant le nom de la variable immédiatement suivi du signe **égal =** puis de la valeur. Par exemple:

```
name="Bob Smith"
```

Les noms de variables doivent commencer par une lettre (caractère alpha) ou un trait de soulignement et contenir uniquement des lettres, des chiffres et le caractère de soulignement. Il est important de se rappeler que les noms de variables sont sensibles à la casse ; a et A sont des variables différentes. Tout comme pour les arguments des commandes, des guillemets simples ou doubles doivent être utilisés lorsque des caractères spéciaux sont inclus dans la valeur attribuée à la variable afin d'empêcher l'expansion du shell.

Valid Variable Assignments	Invalid Variable Assignments
a=1	1=a
_1=a	a-1=3
LONG_VARIABLE='OK'	LONG-VARIABLE='WRONG'
Name='Jose Romero'	'user name'=anything

Le shell Bash et de nombreuses commandes utilisent largement les variables. L'une des principales utilisations des variables est de fournir un moyen de configurer diverses préférences pour chaque utilisateur. Le shell et les commandes Bash peuvent se comporter de différentes manières, en fonction de la valeur des variables. Pour le shell, les variables peuvent affecter ce que l'invite affiche, les répertoires dans lesquels le shell recherchera les commandes à exécuter et bien plus encore.

3.2.1 Variables locales et environnementales

Une ***variable locale*** n'est disponible que pour le shell dans lequel elle a été créée. Une ***variable d'environnement*** est disponible pour le shell dans lequel elle a été créée et elle est transmise à toutes les autres commandes/programmes démarrés par le shell.

Pour définir la valeur d'une variable, utilisez l'expression d'affectation suivante. Si la variable existe déjà, la valeur de la variable est modifiée. Si le nom de la variable n'existe pas déjà, le shell crée une nouvelle variable locale et définit la valeur :

```
variable=value
```

Dans l'exemple ci-dessous, une variable locale est créée, et la commande **echo** est utilisée pour afficher sa valeur :

```
sysadmin@localhost:~$ name="judy"
sysadmin@localhost:~$ echo $name
judy
```

Par convention, les caractères minuscules sont utilisés pour créer des noms de variables locales et les caractères majuscules sont utilisés pour nommer une variable d'environnement. Par exemple, une variable locale peut être appelée *test* tandis qu'une variable d'environnement peut être appelée *TEST*. Bien qu'il s'agisse d'une convention que la plupart des gens suivent, ce n'est pas une règle. Une variable d'environnement peut être créée directement à l'aide de la commande **export**.

```
export variable=value
```

Dans l'exemple ci-dessous, une variable d'environnement est créée avec la commande **export** :

```
sysadmin@localhost:~$ export JOB=engineer
```

Rappelez-vous que la commande **echo** est utilisée pour afficher la sortie dans le terminal. Pour afficher la valeur de la variable, utilisez le signe **dollar $** suivi du nom de la variable comme argument de la commande echo :

```
sysadmin@localhost:~$ echo $JOB
engineer
```

Nous pouvons voir la différence entre les variables locales et d'environnement en ouvrant un nouveau shell avec la commande **bash** :

```
sysadmin@localhost:~$ bash
To run a command as administrator (user "root"), use "sudo" command.
See "man sudo_root" for details.
sysadmin@localhost:~$ echo $name
sysadmin@localhost:~$ echo $JOB
engineer
```

Notez que le nom de la variable locale est vide, tandis que la variable d'environnement JOB renvoie la valeur de l'ingénieur. Notez également que si vous revenez au premier shell en utilisant la commande exit, les deux variables sont toujours disponibles dans le shell d'origine :

```
sysadmin@localhost:~$ exit
exit
sysadmin@localhost:~$ echo $name
judy
sysadmin@localhost:~$ echo $JOB
engineer
```

Le nom de la variable locale n'est pas disponible dans le nouveau shell car par défaut, lorsqu'une variable est affectée dans le shell Bash, elle est initialement définie comme variable locale. Lorsque vous quittez le shell d'origine, seules les variables d'environnement seront disponibles. Il existe plusieurs manières de transformer une variable locale en variable d'environnement.

Tout d'abord, une variable locale existante peut être exportée avec la commande **export**.

Mr William

```
export variable
```

Dans l'exemple ci-dessous, le nom de la variable locale est exporté vers l'environnement (avec la convention standard des majuscules) :

```
sysadmin@localhost:~$ NAME=judy
sysadmin@localhost:~$ export NAME
sysadmin@localhost:~$ echo $NAME
judy
```

Deuxièmement, une nouvelle variable peut être exportée et assignée à une valeur avec une seule commande, comme illustré ci-dessous avec la variable DEPARTMENT :

```
sysadmin@localhost:~$ export DEPARTMENT=science
sysadmin@localhost:~$ echo $DEPARTMENT
science
```

Troisièmement, la commande **declare** ou **typeset** peut être utilisée avec l'option d'*export* **-x** pour déclarer une variable comme variable d'environnement. Ces commandes sont synonymes et fonctionnent de la même manière :

```
sysadmin@localhost:~$ declare -x EDUCATION=masters
sysadmin@localhost:~$ echo $EDUCATION
masters
sysadmin@localhost:~$ typeset -x EDUCATION=masters
sysadmin@localhost:~$ echo $EDUCATION
masters
```

La commande **env** est utilisée pour exécuter des commandes dans un environnement modifié. Il peut également être utilisé pour créer ou modifier temporairement des variables d'environnement qui ne sont transmises qu'à une seule exécution de commande en utilisant la syntaxe suivante :

```
env [NAME=VALUE] [COMMAND]
```

Par exemple, les serveurs sont souvent réglés sur le temps universel coordonné (UTC), ce qui est utile pour maintenir une heure constante sur les serveurs de la planète, mais peut s'avérer frustrant dans la pratique pour simplement indiquer l'heure :

```
sysadmin@localhost:~$ date
Sun Mar 10 22:47:44 UTC 2020
```

Pour définir temporairement la variable de fuseau horaire, utilisez la commande **env**. Ce qui suit exécutera la commande **date** avec l'affectation de variable temporaire :

```
sysadmin@localhost:~$ env TZ=EST date
Sun Mar 10 17:48:16 EST 2020
```

La variable TZ est définie uniquement dans l'environnement du shell actuel, et uniquement pour la durée de la commande. Le reste du système ne sera pas affecté par cette variable. En fait, réexécuter la commande date vérifiera que la variable TZ est revenue à UTC

```
sysadmin@localhost:~$ date
Sun Mar 10 22:49:46 UTC 2020
```

3.2.2 Affichage des variables

Il existe plusieurs manières d'afficher les valeurs des variables. La commande **set** affichera à elle seule toutes les variables (locales et environnementales). Ici, nous allons diriger la sortie vers la commande **tail** afin que nous puissions voir certaines des variables définies dans la section précédente :

```
sysadmin@localhost:~$ set | tail
DEPARTMENT=science
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments
TERM=xterm
UID=1001
USER=sysadmin
VISUAL=vi
_=set
name=judy
```

Note

L'exemple ci-dessus utilise un tube de ligne de commande, représenté par le | personnage. Le caractère barre verticale peut être utilisé pour envoyer la sortie d'une commande à une autre.

Les tuyaux de ligne de commande seront abordés plus en détail plus tard dans le cours.

Note

La commande **tail** est utilisée pour afficher uniquement les dernières lignes d'un fichier (ou, lorsqu'elle est utilisée avec un tube, la sortie d'une commande précédente). Par défaut, la commande tail affiche dix lignes d'un fichier fourni en argument.

La commande **tail** sera abordée plus en détail plus tard dans le cours.

Pour afficher uniquement les variables d'environnement, vous pouvez utiliser plusieurs commandes qui fournissent presque le même résultat :

```
env
declare -x
typeset -x
export -p
```

```
sysadmin@localhost:~$ env | tail
NAME=judy
MAIL=/var/mail/sysadmin
SHELL=/bin/bash
TERM=xterm
SHLVL=1
EDUCATION=masters
LOGNAME=sysadmin
PATH=/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
/bin:/usr/games:/usr/local/games
LESSOPEN=| /usr/bin/lesspipe %s
_=/usr/bin/env
```

Pour afficher la valeur d'une variable spécifique, utilisez la commande **echo** avec le nom de la variable préfixé par le $ (signe dollar). Par exemple, pour afficher la valeur de la variable PATH, vous exécuterez **echo $PATH** :

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
```

Considère ceci

Les variables peuvent également être entourées d'accolades {} afin de les délimiter du texte environnant. Alors que la commande **echo ${PATH}** produirait le même résultat que la commande **echo $PATH**, les accolades distinguent visuellement la variable, ce qui la rend plus facile à voir dans les scripts dans certains contextes.

Mr William

3.2.3 Désactivation des variables

Si vous créez une variable et que vous ne souhaitez plus que cette variable soit définie, utilisez la commande **unset** pour la supprimer :

```
unset VARIABLE
```

```
sysadmin@localhost:~$ example=12
sysadmin@localhost:~$ echo $example
12
sysadmin@localhost:~$ unset example
sysadmin@localhost:~$ echo $example
```

Avertissement

Ne désactivez pas les variables système critiques telles que la variable PATH, car cela pourrait entraîner un dysfonctionnement de l'environnement.

3.2.4 Variable CHEMIN (PATH)

La variable ***PATH*** est l'une des variables d'environnement les plus critiques pour le shell, il est donc important de comprendre son effet sur la façon dont les commandes seront exécutées.

La variable ***PATH*** contient une liste de répertoires utilisés pour rechercher les commandes saisies par l'utilisateur. Lorsque l'utilisateur tape une commande puis appuie sur la touche Entrée, les répertoires PATH sont recherchés pour un fichier exécutable correspondant au nom de la commande. Le traitement s'effectue à travers la liste des répertoires de gauche à droite ; le premier fichier exécutable qui correspond à ce qui est tapé est la commande que le shell va tenter d'exécuter.

Note

Avant de rechercher la commande dans la variable PATH, le shell déterminera d'abord si la commande est un alias ou une fonction, ce qui peut entraîner la non-utilisation de la variable PATH lorsque cette commande spécifique est exécutée.

De plus, si la commande est intégrée au shell, la variable PATH ne sera pas utilisée.

L'utilisation de la commande **echo** pour afficher le $PATH actuel renverra tous les répertoires à partir desquels les fichiers peuvent être exécutés. L'exemple suivant affiche une variable PATH typique, avec des noms de répertoires séparés les uns des autres par un caractère deux-points : :

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
```

Le tableau suivant illustre l'objectif de certains des répertoires affichés dans le résultat de la commande précédente :

Directory	Contents
`/home/sysadmin/bin`	A directory for the current user `sysadmin` to place programs. Typically used by users who create their own scripts.
`/usr/local/sbin`	Normally empty, but may have administrative commands that have been compiled from local sources.
`/usr/local/bin`	Normally empty, but may have commands that have been compiled from local sources.
`/usr/sbin`	Contains the majority of the administrative command files.
`/usr/bin`	Contains the majority of the commands that are available for regular users to execute.
`/sbin`	Contains the essential administrative commands.
`/bin`	Contains the most fundamental commands that are essential for the operating system to function.

Pour exécuter des commandes qui ne sont pas contenues dans les répertoires répertoriés dans la variable PATH, plusieurs options existent :

- La commande peut être exécutée en tapant le chemin absolu de la commande.
- La commande peut être exécutée avec un chemin relatif vers la commande.
- La variable PATH peut être définie pour inclure le répertoire où se trouve la commande.
- La commande peut être copiée dans un répertoire répertorié dans la variable PATH.

Pour illustrer les chemins absolus et relatifs, un script exécutable nommé my.sh est créé dans le répertoire personnel. Ensuite, ce fichier de script reçoit des « autorisations d'exécution » (les autorisations sont couvertes dans un chapitre ultérieur) :

```
sysadmin@localhost:~$ echo 'echo Hello World!' > my.sh
sysadmin@localhost:~$ chmod u+x my.sh
```

Mr William

Un chemin absolu spécifie l'emplacement d'un fichier ou d'un répertoire depuis le répertoire de niveau supérieur jusqu'à tous les sous-répertoires jusqu'au fichier ou au répertoire. Les chemins absolus commencent toujours par le caractère / représentant le répertoire racine. Par exemple, */usr/bin/ls* est un chemin absolu. Cela signifie le fichier ls, qui se trouve dans le répertoire bin, qui se trouve dans le répertoire *usr*, qui se trouve dans le répertoire / (racine). Un fichier peut être exécuté en utilisant un chemin absolu comme ceci :

```
sysadmin@localhost:~$ /home/sysadmin/my.sh
Hello World!
```

Un chemin relatif spécifie l'emplacement d'un fichier ou d'un répertoire par rapport au répertoire actuel. Par exemple, dans le répertoire */home/sysadmin*, un chemin relatif de *test/newfile* ferait en fait référence au fichier */home/sysadmin/test/newfile*. Les chemins relatifs ne commencent jamais par le caractère /.

L'utilisation d'un chemin relatif pour exécuter un fichier dans le répertoire courant nécessite l'utilisation du . caractère, qui symbolise le répertoire courant :

```
sysadmin@localhost:~$ ./my.sh
Hello World!
```

Parfois, un utilisateur souhaite que son répertoire personnel soit ajouté à la variable PATH afin d'exécuter des scripts et des programmes sans utiliser ./ devant le nom du fichier. Ils pourraient être tentés de modifier la variable PATH comme ceci :

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$ PATH=/home/sysadmin
```

Malheureusement, modifier une variable de cette manière écrase le contenu. Par conséquent, tout ce qui était auparavant contenu dans la variable PATH sera perdu.

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin
```

Les programmes répertoriés en dehors du répertoire */home/sysadmin* ne seront désormais accessibles qu'en utilisant leur nom de chemin complet. Par exemple, en supposant que le répertoire personnel n'a pas encore été ajouté à la variable PATH, la commande **uname -a** se comporterait comme prévu :

```
sysadmin@localhost:~$ uname -a
Linux localhost 3.15.6+ #2 SMP Wed Jul 23 01:26:02 UTC 2014 x86_64 x86_64 x
```

Après avoir attribué le répertoire personnel à la variable PATH, la commande uname -a aura besoin de son nom de chemin complet pour s'exécuter correctement :

```
sysadmin@localhost:~$ PATH=/home/sysadmin
sysadmin@localhost:~$ uname -a
-bash: uname: command not found
sysadmin@localhost:~$ /bin/uname -a
Linux localhost 3.15.6+ #2 SMP Wed Jul 23 01:26:02 UTC 2014 x86_64 x86_64 x
```

Important

Si vous avez modifié la variable PATH comme nous l'avons fait dans l'exemple précédent et que vous souhaitez la réinitialiser, utilisez simplement la commande **exit** pour vous déconnecter :

```
sysadmin@localhost:~$ exit
```

Une fois reconnecté, la variable PATH sera réinitialisée à sa valeur d'origine.

Il est possible d'ajouter un nouveau répertoire à la variable PATH sans écraser son contenu précédent. Importez la valeur actuelle de la variable $PATH dans la variable PATH nouvellement définie en l'utilisant des deux côtés de l'instruction d'affectation :

```
sysadmin@localhost:~$ PATH=$PATH
```

Terminez-le avec la valeur du chemin supplémentaire du répertoire personnel :

```
sysadmin@localhost:~$ PATH=$PATH:/home/sysadmin
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
```

Désormais, les scripts situés dans le répertoire /home/sysadmin peuvent s'exécuter sans utiliser de chemin :

```
sysadmin@localhost:~$ my.sh
Hello World!
```

Mr William

Avertissement

En général, c'est une mauvaise idée de modifier la variable $PATH. Si cela devait changer, les administrateurs le considéreraient comme une activité suspecte. Les forces malveillantes veulent obtenir des privilèges élevés et accéder aux informations sensibles résidant sur les serveurs Linux. Une façon de procéder consiste à écrire un script qui partage le nom d'une commande système, puis à modifier la variable PATH pour inclure le répertoire personnel de l'administrateur. Lorsque l'administrateur tape la commande, il exécute en fait le script malveillant !

3.3 Fichiers d'initialisation

Lorsqu'un utilisateur ouvre un nouveau shell, soit lors de la connexion, soit lorsqu'il exécute un terminal qui démarre un shell, le shell est personnalisé par des fichiers appelés fichiers d'initialisation (ou de configuration). Ces fichiers d'initialisation définissent la valeur des variables, créent des alias et des fonctions et exécutent d'autres commandes utiles au démarrage du shell.

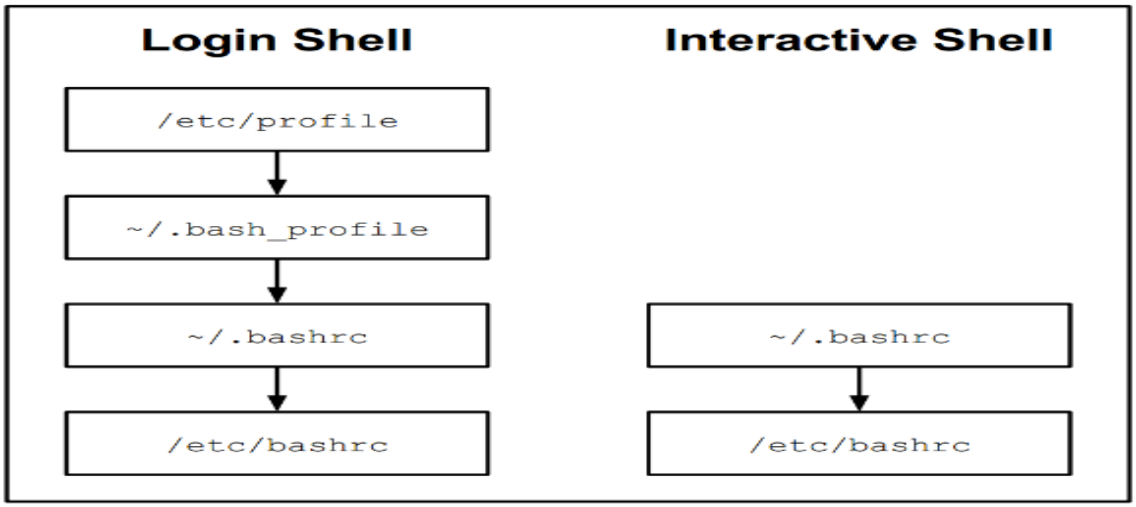
Il existe deux types de fichiers d'initialisation : les fichiers d'initialisation globaux qui affectent tous les utilisateurs du système et les fichiers d'initialisation locaux spécifiques à un utilisateur individuel.

Les fichiers de configuration globale se trouvent dans le répertoire /etc. Les fichiers de configuration locaux sont stockés dans le répertoire personnel de l'utilisateur.

Fichiers d'initialisation BASH

Chaque shell utilise des fichiers d'initialisation différents. De plus, la plupart des shells exécutent des fichiers d'initialisation différents lorsque le shell est démarré via le processus de connexion (appelé shell de connexion) et lorsqu'un shell est démarré par un terminal (appelé shell sans connexion ou shell interactif).

Le diagramme suivant illustre les différents fichiers démarrés avec un shell de connexion classique par rapport à un shell interactif :



Note

Le caractère **tilde ~** représente le répertoire personnel de l'utilisateur.

Note

Noms de fichiers précédés d'un point . Le caractère indique les fichiers cachés. Vous pouvez afficher ces fichiers dans votre répertoire personnel en utilisant la commande **ls** avec l'option **all -a.**

```
sysadmin@localhost:~$ ls -a
.               .bashrc   .selected_editor  Downloads  Public
..               .cache Desktop         Music   Templates
.bash_logout  .profile  Documents       Pictures   Videos
```

Lorsque Bash est démarré en tant que shell de connexion, le fichier /etc/profile est exécuté en premier. Ce fichier exécute généralement tous les fichiers se terminant par .sh qui se trouvent dans le répertoire /etc/profile.d.

Le fichier suivant exécuté est généralement ~/.bash_profile (mais certains utilisateurs peuvent utiliser le fichier ~/.bash_login ou ~/.profile). Le fichier ~/.bash_profile exécute généralement également le fichier ~/.bashrc qui à son tour exécute le fichier /etc/bashrc.

Considère ceci

Étant donné que le fichier ~/.bash_profile est sous le contrôle de l'utilisateur, l'exécution des fichiers ~/.bashrc et /etc/bashrc est également contrôlable par l'utilisateur.

Lorsque Bash est démarré en tant que shell interactif, il exécute le fichier ~/.bashrc, qui peut également exécuter le fichier /etc/bashrc, s'il existe. Encore une fois, puisque le fichier ~/.bashrc appartient à l'utilisateur qui se connecte, l'utilisateur peut empêcher l'exécution du fichier /etc/bashrc.

Avec autant de fichiers d'initialisation, une question courante à ce stade est « quel fichier suis-je censé utiliser ? » Le tableau suivant illustre l'objectif de chacun de ces fichiers, en fournissant des exemples des commandes que vous pouvez placer dans chaque fichier :

File	Purpose
`/etc/profile`	This file can only be modified by the administrator and will be executed by every user who logs in. Administrators use this file to create key environment variables, display messages to users as they log in, and set key system values.

`~/.bash_profile`
`~/.bash_login`
`~/.profile`
	Each user has their own `.bash_profile` file in their home directory. The purpose of this file is the same as the `/etc/profile` file, but having this file allows a user to customize the shell to their own tastes. This file is typically used to create customized environment variables.

`~/.bashrc`	Each user has their own `.bashrc` file in their home directory. The purpose of this file is to generate items that need to be created for each shell, such as local variables and aliases.
`/etc/bashrc`	This file may affect every user on the system. Only the administrator can modify this file. Like the `.bashrc` file, the purpose of this file is to generate items that need to be created for each shell, such as local variables and aliases.

3.3.1 Modification des fichiers d'initialisation

La façon dont fonctionne le shell d'un utilisateur peut être modifiée en modifiant les fichiers d'initialisation de cet utilisateur. La modification de la configuration globale nécessite un accès administratif au système car les fichiers du répertoire /etc ne peuvent être modifiés que par un administrateur. Un utilisateur ne peut modifier que les fichiers d'initialisation dans son répertoire personnel.

La meilleure pratique consiste à créer une sauvegarde avant de modifier les fichiers de configuration pour vous assurer contre les erreurs ; une sauvegarde peut toujours être restaurée en cas de problème. L'exemple suivant fait référence à CentOS, une distribution Linux prise en charge par la communauté et basée sur le code source de Red Hat Enterprise Linux.

Dans certaines distributions, le fichier ~/.bash_profile par défaut contient les deux lignes suivantes qui personnalisent la variable d'environnement PATH :

```
PATH=$PATH:$HOME/bin
export PATH
```

La première ligne définit la variable PATH sur la valeur existante de la variable PATH avec l'ajout du sous-répertoire bin du répertoire personnel de l'utilisateur. La variable $HOME fait référence au répertoire personnel de l'utilisateur. Par exemple, si l'utilisateur qui se connecte est Joe, alors $HOME/bin est /home/joe/bin.

La deuxième ligne convertit la variable PATH locale en variable d'environnement.

Le fichier ~/.bashrc par défaut exécute /etc/bashrc en utilisant une instruction telle que :

```
./etc/bashrc
```

Le caractère **période .** est utilisé pour sourcer un fichier afin de l'exécuter. Le caractère point a également une commande synonyme, la commande **source**, mais l'utilisation du caractère point est plus courante que l'utilisation de la commande **source**.

Le sourcing peut être un moyen efficace de tester les modifications apportées aux fichiers d'initialisation, vous permettant de corriger les erreurs sans avoir à vous déconnecter et à vous reconnecter. Si vous mettez à jour le fichier ~/.bash_profile pour modifier la variable PATH, vous pouvez vérifier que vos modifications sont correctes en exécutant ce qui suit :

```
. ~/.bash_profile
echo $PATH
```

3.3.2 Scripts de sortie BASH

Tout comme Bash exécute un ou plusieurs fichiers au démarrage, il peut également exécuter un ou plusieurs fichiers à la sortie. À la sortie de Bash, il exécutera les fichiers ~/.bash_logout et /etc/bash_logout, s'ils existent. Généralement, ces fichiers sont utilisés pour des tactiques de « nettoyage » lorsque l'utilisateur quitte le shell. Par exemple, le fichier ~/.bash_logout par défaut exécute la commande **clear** pour supprimer tout texte présent sur l'écran du terminal.

Considère ceci

Lorsqu'un nouvel utilisateur est créé, les fichiers du répertoire /etc/skel sont automatiquement copiés dans le répertoire personnel du nouvel utilisateur. En tant qu'administrateur, vous pouvez modifier les fichiers du répertoire/etc/skel pour fournir des fonctionnalités personnalisées aux nouveaux utilisateurs.

3.4 La command History

Dans un sens, le fichier ~/.bash_history pourrait également être considéré comme un fichier d'initialisation, puisque Bash lit également ce fichier au démarrage. Par défaut, ce fichier contient un historique des commandes qu'un utilisateur a exécutées dans le shell Bash. Lorsqu'un utilisateur quitte le shell Bash, il écrit l'historique récent dans ce fichier.

Cet historique de commandes présente plusieurs avantages pour l'utilisateur :

Vous pouvez utiliser les touches fléchées **Haut ↑** et **Bas ↓** pour consulter votre historique et sélectionner une commande précédente à exécuter à nouveau.

Vous pouvez sélectionner une commande précédente et la modifier avant de l'exécuter.

Vous pouvez effectuer une recherche inversée dans l'historique pour trouver une commande précédente afin de la sélectionner, de la modifier et de l'exécuter. Pour lancer la recherche, appuyez sur **Ctrl+R**, puis commencez à taper une partie d'une commande précédente.

Vous exécutez à nouveau une commande, en fonction d'un numéro associé à la commande.

3.4.1 Exécution des commandes précédentes

Le shell Bash permettra à un utilisateur d'utiliser la touche fléchée vers le **haut ↑** pour afficher les commandes précédentes. À chaque pression sur la flèche vers le haut, le shell affiche une commande supplémentaire dans la liste de l'historique.

Si l'utilisateur remonte trop loin, la touche Flèche **Bas ↓** peut être utilisée pour revenir à une commande plus récente. Une fois la commande correcte affichée, vous pouvez appuyer sur la touche Entrée pour l'exécuter.

Les touches fléchées **Gauche ←** et **Droite →** peuvent également être utilisées pour positionner le curseur dans la commande. Les touches **Retour arrière** (Backspace) et **Suppr** (delete) sont utilisées pour supprimer du texte, et des caractères supplémentaires peuvent être saisis dans la ligne de commande pour le modifier avant d'appuyer sur la touche Entrée pour l'exécuter.

D'autres touches peuvent être utilisées pour modifier une commande sur la ligne de commande. Le tableau suivant résume quelques clés d'édition utiles :

Action	Key	Alternate Key Combination
Previous history item	↑	**Ctrl+P**
Next history item	↓	**Ctrl+N**
Reverse history search		**Ctrl+R**
Beginning of line	**Home**	**Ctrl+A**
End of line	**End**	**Ctrl+E**
Delete current character	**Delete**	**Ctrl+D**
Delete to the left of the cursor	**Backspace**	**Ctrl+X**
Move cursor left	←	**Ctrl+B**
Move cursor right	→	**Ctrl+F**

3.4.2 Modification des clés d'édition

Les touches disponibles pour modifier une commande sont déterminées par les paramètres d'une bibliothèque appelée **Readline**. Les touches sont généralement définies par défaut pour correspondre aux affectations de touches trouvées dans l'éditeur de texte **emacs** (un éditeur Linux populaire).

Pour lier les touches afin qu'elles correspondent aux affectations de touches trouvées avec un autre éditeur de texte populaire, l'éditeur vi, le shell peut être configuré avec la commande **set -o vi**. Pour redéfinir les raccourcis clavier sur l'éditeur de texte **emacs**, utilisez la commande **set -o emacs**.

Note

Si vous ne savez pas encore comment utiliser l'éditeur **vi**, vous souhaiterez peut-être vous en tenir aux touches **emacs** car l'éditeur **vi** a une courbe d'apprentissage plus longue.

L'éditeur vi sera abordé plus en détail plus tard dans le cours.

Pour configurer automatiquement les options de l'historique des modifications lors de la connexion, modifiez le fichier ~/.inputrc. Si ce fichier n'existe pas, le fichier /etc/inputrc est utilisé à la place. Les raccourcis clavier sont définis différemment dans les fichiers de configuration et sur la ligne de commande ; par exemple, pour activer le mode de liaison de clé vi pour un individu, ajoutez les lignes suivantes au fichier ~/.inputrc :

```
set editing-mode vi
set keymap vi
```

Si la situation est inversée, peut-être en raison de l'ajout des deux lignes ci-dessus au fichier /etc/inputrc, un utilisateur peut activer le mode **emacs** en ajoutant les lignes suivantes au fichier ~/.inputrc :

```
set editing-mode emacs
set keymap emacs
```

3.4.3 Utilisation de la commande history

La commande **history** peut être utilisée pour réexécuter des commandes précédemment exécutées. Lorsqu'elle est utilisée sans argument, la commande **history** fournit une liste des commandes précédemment exécutées :

```
sysadmin@localhost:~$ history
        1  ls
        2  cd test
        3  cat alpha.txt
        4  ls -l
        5  cd ..
        6  ls
        7  history
```

Notez que chaque commande se voit attribuer un numéro qu'un utilisateur peut utiliser pour réexécuter la commande.

La commande history propose de nombreuses options ; les options les plus courantes sont répertoriées ci-dessous :

Option	Purpose
`-c`	Clear the list
`-r`	Read the history file and replace the current history
`-w`	Write the current history list to the history file

Comme la liste **history** contient généralement cinq cents commandes ou plus, il est souvent utile de filtrer la liste. La commande **history** accepte un nombre comme argument pour indiquer le nombre de commandes à répertorier. Par exemple, l'exécution de la commande suivante n'affichera que les trois dernières commandes de votre historique.

Mr William

```
sysadmin@localhost:~$ history 3
        5  cd ..
        6  ls
        7  history
```

Note

La commande **grep** est très utile pour filtrer la sortie des commandes qui produisent une sortie abondante. Par exemple, pour afficher toutes les commandes de votre historique qui contiennent la commande ls, recherchez le modèle ls à l'aide de la commande suivante :

```
sysadmin@localhost:~$ history | grep "ls"
    1  ls
    4  ls -l
    6  ls
    9  history | grep "ls"
```

3.4.4 Configuration de la commande history

Lorsque vous fermez le programme shell, il prend les commandes dans la liste d'historique et les stocke dans le fichier ~/.bash_history, également appelé fichier historique. Par défaut, cinq cents commandes seront stockées dans le fichier historique. La variable HISTFILESIZE déterminera le nombre de commandes à écrire dans ce fichier.

Si un utilisateur souhaite stocker les commandes d'historique dans un fichier différent de ~/.bash_history, alors l'utilisateur peut spécifier un chemin absolu vers l'autre fichier comme valeur de la variable locale HISTFILE :

```
HISTFILE=/path/to/file
```

La variable HISTSIZE déterminera le nombre de commandes à conserver en mémoire pour chaque shell Bash. Si la taille de HISTSIZE est supérieure à la taille de HISTFILESIZE, alors seul le nombre de commandes le plus récent spécifié par HISTFILESIZE sera écrit dans le fichier historique à la fermeture du shell Bash.

Bien qu'elle ne soit normalement définie sur rien par défaut, vous souhaiterez peut-être profiter de la définition d'une valeur pour la variable HISTCONTROL dans un fichier d'initialisation tel que le fichier ~/.bash_profile. La variable HISTCONTROL peut être définie sur l'une des fonctionnalités suivantes :

```
HISTCONTROL=ignoredups
```

Empêche les commandes en double exécutées consécutivement.

```
HISTCONTROL=ignorespace
```

Toute commande commençant par un espace ne sera pas stockée. Cela fournit à l'utilisateur un moyen simple d'exécuter une commande qui n'entrera pas dans la liste de l'historique.

```
HISTCONTROL=ignoreboth
```

Les doublons consécutifs et les commandes commençant par un espace ne seront pas stockés.

```
HISTCONTROL=erasedups
```

Les commandes identiques à une autre commande de votre historique ne seront pas stockées. (L'entrée précédente de la commande sera supprimée de la liste de l'historique.)

```
HISTCONTROL=ignorespace:erasedups
```

Inclut l'avantage des suppressions effacées avec l'avantage d'ignorer l'espace.

Une autre variable qui affectera ce qui est stocké dans l'historique des commandes est la variable HISTIGNORE. La plupart des utilisateurs ne souhaitent pas que la liste d'historique soit encombrée de commandes de base telles que les commandes **ls, cd, exit** et **history**. La variable HISTIGNORE peut être utilisée pour indiquer à Bash de ne pas stocker certaines commandes dans la liste historique.

Pour que les commandes ne soient pas incluses dans la liste de l'historique, incluez une affectation comme celle-ci dans le fichier ~/.bash_profile :

```
HISTIGNORE='ls*:cd*:history*:exit'
```

Note

Le caractère ***** est utilisé pour indiquer autre chose après la commande. Ainsi, ls* correspondrait à n'importe quelle commande ls, telle que ls -l ou ls etc.

Le globaling sera abordé plus en détail plus tard dans le cours.

3.4.5 Exécution des commandes précédentes

Le ***point d'exclamation !*** est un caractère spécial du shell Bash pour indiquer l'exécution d'une commande dans la liste d'historique. Il existe de nombreuses façons d'utiliser le caractère

d'exclamation **!** pour réexécuter les commandes ; par exemple, l'exécution de deux caractères d'exclamation répétera la commande précédente :

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
sysadmin@localhost:~$ !!
ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

Le tableau suivant fournit quelques exemples d'utilisation du caractère exclamation ! :

History Command	Meaning
`!!`	Repeat the last command
`!-4`	Execute the command that was run four commands ago
`!555`	Execute command number `555`
`!ec`	Execute the last command that started with `ec`
`!?joe`	Execute the last command that contained `joe`

Une autre façon d'utiliser la liste historique consiste à tirer parti du fait que le dernier argument de chaque commande est stocké. Souvent, un utilisateur tape une commande pour faire référence à un fichier, peut-être pour afficher des informations sur ce fichier. Par la suite, l'utilisateur souhaitera peut-être taper une autre commande pour faire autre chose avec le même fichier. Au lieu d'avoir à saisir à nouveau le chemin, un utilisateur peut utiliser quelques raccourcis clavier pour rappeler ce chemin de fichier à partir de la ligne de commande précédente.

En appuyant soit sur **Esc + .** (Échap+Période) ou **Alt + .** (Alt+Période), le shell ramènera le dernier argument de la commande précédente. En appuyant plusieurs fois sur l'une de ces combinaisons de touches, vous rappellerez, dans l'ordre inverse, le dernier argument de chaque commande précédente.