Interragissez avec le Bash

Il ne faut pas oublier que le shell est un **interpréteur d'un langage de commande**. Les lignes de commande sont analysées puis exécutées au fur et à mesure de leur lecture. En mode interactif,l'utilisateur saisit et édite la ligne de commande à partir du clavier. En mode script, la ligne de commande est lue à partir d'un fichier. Nous reviendrons sur ce mode dans la dernière séquence. Dans cette séquence, nous allons voir les différents traitements qui sont opérés par le mode interactif.

Le fonctionnement du shell suit ces étapes :

- 1. La saisie de ligne de commande devient possible lorsque l'invite de commande apparaît. C'est ce que vous avez vu dans la séquence 1.
- 2. La saisie est aidée par les fonctions d'édition. En effet lorsque vous utilisez un clavier, vous serez amené à faire des fautes de frappe ou à utiliser plusieurs fois la même commande, le Bash vous facilite l'interaction.
- 3. Dès que la touche de retour à la ligne est enfoncée, le shell analyse la ligne de commande. Cela consiste à identifier la commande et ses arguments. Les arguments peuvent être des constructions syntaxiques comme, par exemple, celle de **la substitution des caractères spéciaux** utilisée pour indiquer un groupe de fichiers dans un répertoire. Nous verrons cette facilité d'expression.
- 4. Puis, l'interprétation de ces constructions est effectuée afin de les substituer par leur résultat. Nous verrons comment utiliser des variables dans la ligne de commande ou le résultat d'une commande.
- 5. Ensuite, l'entrée et la sortie de la commande sont définies. nous détaillerons comment alimenter en données et récupérer le résultat d'une commande.
- 6. Finalement, la commande avec ses arguments substitués, est exécutée par l'appel du programme correspondant. Vous serez capable de contrôler l'exécution d'une commande avec la notion de processus.

1 Édition de la ligne de commande

Les raccourcis clavier sont introduits par les touches spéciales CTRL ou Alt, et sur certaines distributions Linux ou sur MacOS c'est la touche d'échappement Esc qui est utilisée à la place de Alt.

1.1 Déplacements

CTRL+a place le curseur au début de la ligne

CTRL+e place le curseur à la fin de la ligne (End)

CTRL+b recule d'un caractère (Backward)

CTRL+f avance d'un caractère (Forward)

Alt+b recule d'un mot

i.e. place le curseur sur la première lettre du mot sur lequel se trouve le curseur

Alt+f avance d'un mot

i.e. place le curseur après la dernière lettre du mot sur lequel se trouve le curseur

1.2 Historique des commandes

Lors d'une session sur un terminal vous serez amenés à utiliser souvent les mêmes commandes ou des commandes quasi identiques. Bash conserve par défaut l'historique des `1000` dernières commandes exécutées et fournit plusieurs moyens de le consulter. La commande **`history`** liste les commandes que vous avez saisies de la plus ancienne jusqu'à la plus récente. Chaque commande est précédée de son rang dans l'historique. Suivi d'un nombre

```
history n
```

n'affiche que les n dernières commandes de l'historique.

1.3 Se déplacer dans l'historique et rappeler une commande

Les touches fléchées ↑ et ↓ permettent de naviguer dans l'historique. Au fur et à mesure du déplacement, la commande correspondant à l'endroit où on se situe dans l'historique s'affiche, il n'y a qu'à frapper **enter** pour l'exécuter.

1.4 Rappeler une commande en utilisant une chaîne de caractères

Un point d'exclamation suivi d'une chaîne de caractères permet de rappeler la commande la plus récente qui commence par cette chaîne. Par exemple, pour rappeler la dernière commande utilisée pour se placer dans un répertoire, on rentrera :

```
!cd
```

Si vous ne voulez pas rexécuter une commande mais simplement la retrouver dans l'historique pourvoir si c'est bien celle que vous voulez il suffit d'ajouter `:p` (Print) en fin de ligne. Ainsi pour afficher la dernière commande utilisée pour se déplacer dans un répertoire, sans l'exécuter, on fera :

```
!cd:p
```

En encadrant une chaîne de caractères entre deux points d'interrogation on peut rechercher la commande la plus récente qui contient cette chaîne. Par exemple, on rappellera la dernière commande qui contient la chaîne **`to`** en tapant :

```
!?to?
```

Ça pourrait être `history` ou bien `ls toto`, alors si on veut vérifier avant de l'exécuter, on l'affiche :

```
!?to?:p
```

2 Auto-complétion

En plus de l'édition de la ligne de commande et de l'historique, Bash offre une troisième forme d'aide à la saisie qui est la plus utilisée : l'auto-complétion. Celle-ci s'effectue au moyen de la touche de tabulation ➔. Après avoir tapé les premiers caractères, la touche tabulation demande à compléter automatiquement le nom de commande. S'il n'y a qu'une seule complétion possible, le Bash complète le nom de la commande en entier.

Par exemple `hist➔` est remplacé par `history` sur la ligne de commande. Lorsqu'il y a une ambiguïté, Bash ne complète pas la ligne de commande après avoir tapé, il suffit alors de taper une deuxième fois la touche ➔ pour avoir une liste des complétions possibles.

Par exemple `ch➔` n'affiche rien car il existe plusieurs commandes commençant par les caractères ch. Un deuxième appui sur ➔ affiche la liste de toutes les possibilités suivi de l'invite `$` et du début de la ligne de commande que vous avez tapée.

```
psimier@b107PSR:~$ ch
chacl            chcon           chgpasswd       chroot
chage            chcpu           chgrp           chrt
chardet3         check_forensic  chktrust        chsh
chardetect3      checkgid        chmem           chvt
charmap          check_signals   chmod
chat             cheese          chown
chattr           chfn            chpasswd
psimier@b107PSR:~$ ch
```

Si après un deuxième appui sur ➔ rien n'apparaît c'est qu'aucune complétion n'est possible. C'est un indication assez forte qu'il y a une erreur de saisie.

3 Encore deux raccourcis

Ctrl + l Permet d'effacer le contenu du terminal.

Ctrl + C En cours de frappe, permet d'arrêter la saisie de la ligne de commande et de revenir à l'invite avec une ligne vierge

4 Variables et options de l'environnement de travail

Pour rappel, en programmation, on peut créer des variables qui sont l'association de noms et de valeurs. Ces associations correspondent à des emplacements dans la mémoire de l'ordinateur. Associer nom et valeur donne la possibilité de traiter des données de manière symbolique. Dans les systèmes de type Unix, l'utilisateur peut créer et utiliser des variables pour profiter de cette association symbolique.Cependant, il faut distinguer les variables créées par l'utilisateur et celles créées et gérées par le shell lui-même. Les variables créées et gérées par le shell sont les variables d'environnement. Ces variables ont des noms prédéfinis et sont utilisées pour et par le shell. Elles sont renseignées par le shell lorsque l'utilisateur se connecte au système. Ces variables peuvent éventuellement être utilisées par d'autres programmes ou par l'utilisateur.

4.1 Variables d'environnement

Les variables d'environnement sont relatives à chaque utilisateur et à chaque session. C'est-à-dire,elles ne peuvent être utilisées et exploitées que dans la session courante. Bien que ces variables soient automatiquement renseignées par le shell (ou initialisées si on utilise un langage de programmation),leur valeur est modifiable par l'utilisateur. Ces modifications influent sur le comportement du shell lors de la session courante. Il existent de nombreuses variables, nous n'allons pas ici en faire le catalogue.Cependant nous citons ci-dessous, quelques noms de variable à retenir.

`HOME : Répertoire personnel de l'utilisateur`

`PATH : Les répertoires contenant les commandes utilisables`

`LOGNAME : Nom de l'utilisateur à la connexion`

`SHELL : Shell utilisé à la connexion`

La variable **`PATH`** est une variable importante, elle donne l'accès aux commandes du système. Cette variable sert à retrouver les commandes dans l'arborescence sans qu'il ne soit nécessaire de fournir le chemin d'accès absolu. La variable `PATH` représente une règle de recherche pour le shell. C'est la liste des répertoires dans lesquels le shell doit rechercher une commande (en suivant l'ordre des répertoires listés dans la variable).

Consulter le contenu d'une variable d'environnement s'effectue par une commande d'affichage **`echo`** et une substitution de variable comme le montre l'exemple avec la variable `PATH`.

```
psimier@b107PSR:~$ echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/
usr/local/games:/snap/bin:/home/USERS/PROFS/psimier/.dotnet/tools
```

La variable HOME contient le chemin d'accès du répertoire personnel de l'utilisateur. La commande cd sans argument utilise la variable d'environnement HOME. Avec cette commande, le répertoire courant va devenir le répertoire personnel de l'utilisateur. L'exemple suivant montre l'action de cette commande dans le cas psimier.

```
psimier@b107PSR:~/2020$ pwd
/home/USERS/PROFS/psimier/2020
psimier@b107PSR:~/2020$ echo $HOME
/home/USERS/PROFS/psimier
psimier@b107PSR:~/2020$ cd
psimier@b107PSR:~$ pwd
/home/USERS/PROFS/psimier
psimier@b107PSR:~$
```

Il est possible de modifier le contenu d'une variable d'environnement pour changer le comportement d'une commande qui utiliserait cette variable.

Exemple

```
psimier@b107PSR:~$ HOME=/home/USERS/PROFS/
psimier@b107PSR:/home/USERS/PROFS/psimier$ cd
psimier@b107PSR:/home/USERS/PROFS$
```

5 Votre environnement de travail

Deux commandes sont disponibles pour manipuler et voir cet environnement : set et env.

Employée sans argument, la commande env, présente les mêmes même résultats que la commande printenv. La différence entre ces commandes est historique entre Unix et Linux. La commande env liste les variables de l'environnement courant. Utilisée avec des arguments, elle sert à modifier les

valeurs des variables de cet environnement. De nombreuses variables ont été enlevées de l'exemple suivant.

```
psimier@b107PSR:~$ env
LANG=fr_FR.UTF-8
USER=psimier
PWD=/home/USERS/PROFS/psimier
HOME=/home/USERS/PROFS/psimier
SHELL=/bin/bash
LANGUAGE=fr_FR
LOGNAME=psimier
```

5.1 Les commandes set, unset et declare

La commande `set` utilisée sans argument fournit les variables de l'environnement mais aussi les variables créées par l'utilisateur dans la session courante.

```
psimier@b107PSR:~$ var=1
psimier@b107PSR:~$ echo $var
1
psimier@b107PSR:~$ set
XDG_SESSION_TYPE=x11
XDG_VTNR=7
_=1
var=1
```

Dans l'exemple précédent, la commande `var=1` crée une variable de nom **var** dont la valeur est 1. Lors de la deuxième exécution de la commande **set**, cette variable et sa valeur sont listées et font donc partie de l'environnement.

La commande unsetpermet de supprimer une variable dont l'utilisateur n'aurait plus besoin. La syntaxe de la commande unset est la suivante

```
psimier@b107PSR:~$ unset var
psimier@b107PSR:~$ echo $var


psimier@b107PSR:~$
```

La commande declare sans argument affiche comme la commande set toutes les variables de la session courante. La seule différence est que la commande declare différencie les variables exportées(voir sous-section suivante) et les variables locales.