1 Introduction

Le shell Bash est désormais omniprésent car il a été intégré à Windows 10 depuis une mise à jour récente. Ainsi, dorénavant, les 3 grands systèmes d'exploitation pour ordinateur (Windows, MacOS et Linux) partagent cet outil puissant et incontournable pour tout informaticien, que ce soit à l'échelle du poste de travail, d'un serveur et même d'un smartphone ou d'une tablette. L'essor des terminaux Android permet en effet également d'intégrer la dimension mobile au spectre d'utilisation du shell. On peut aussi noter que l'engouement des nano-ordinateurs comme les Raspberry PI et plus généralement de l'instrumentation connectée (ou Internet des objets) fonctionne presque exclusivement sous une distribution Linux. L'utilisation du shell est donc largement transversale et sa connaissance est un passage obligé pour tous les informaticiens.

2 Pourquoi connaître le shell ?

La science informatique revêt plusieurs aspects, un des plus fondamentaux porte sur le langage pour écrire des programmes. Dans la multitude des langages de programmation, le langage de programmation shell occupe une place à part. Et pour bien marquer cette différence, on parle de script et non de programme. En effet, en plus d'être un langage de programmation, le shell est aussi un interpréteur de commandes accessible depuis la console (aussi appelé terminal). Le shell offre alors une interface homme-machine pour décrire, sous la forme de lignes de commande, les actions à effectuer au sein du système d'exploitation. Cette interface textuelle nécessite de la pratique pour être maîtrisée et elle est souvent ressentie comme complexe à apprendre. Par la flexibilité et la richesse qu'elle propose, elle est indispensable pour celui qui souhaite administrer et configurer un ordinateur personnel ou un serveur sous Linux.

La programmation de script shell répond à la philosophie Unix classique consistant à diviser les tâches complexes en sous-tâches plus simples, puis à chaîner des composants et des utilitaires. Ces utilitaires sous forme de commandes sont comme une boite à outils disponible dans les systèmes de type Unix. Beaucoup considèrent que c'est la meilleure approche, ou tout au moins plus agréable pour la résolution d'un problème que d'avoir recours à écrire un programme complet avec un langage puissant et généraliste. Cela impose cependant une modification du mode de réflexion, qui doit être adapté aux utilitaires disponibles.

3 La ligne de commande

Lorsque l'invite de commande s'affiche, cela signifie que l'utilisateur peut saisir une nouvelle ligne de commande. La ligne de commande regroupe une ou plusieurs commandes et se termine par un retour à la ligne. Une commande est composée d'un nom qui décrit de façon mnémonique une tâche ou un programme, parfois suivie d'arguments qui permettent de spécifier les paramètres de la tâche à effectuer.

```
$ cal -m apr
```

L’exemple ci-dessus est une ligne de commande qui ne contient qu’une seule commande. Le nom de la commande est cal (diminutif de calendrier) suivi de deux arguments -m (diminutif de mois) et apr (diminutif de April). Avec seulement quelques caractères à saisir, cette commande permet d’exprimer la tâche « afficher-le-calendrier-du-mois-d’avril ». Vous en conviendrez, c’est plutôt efficace.

Après avoir introduit l’invite de commande, nous détaillerons les notions de commande et ligne de commande. Nous y verrons également l’algorithme du shell, c’est-à-dire la suite d’actions génériques qui suit l’appel à une commande. Ces notions seront illustrées à travers des commandes classiques telles que echo, date et cal.

4 Invite de commande

Lorsque vous démarrez un terminal sur lequel le shell Bash est présent, vous êtes accueillis avec l’invite de commande appelée prompt en anglais. Elle vous indique que le terminal est prêt à accepter votre ligne de commande. Cette invite de commande apparaît également chaque fois que la précédente commande a fini son exécution. Son format et sa signification peuvent varier en fonction des configurations mais la plupart du temps elle est similaire au format suivant :

```
"login"@"nom d’hôte":"répertoire courant"[$|#]
```

L’identifiant de l’utilisateur correspond à l’identifiant du compte shell associé (on parle de login). Les commandes vont être exécutées pour le compte de l’utilisateur et avec les droits de cet utilisateur. Le nom d’hôte (ou nom de machine) indique la machine sur laquelle les commandes sont exécutées.

```
alice@pc-alice.boulot.fr:~$
```

Dans l’exemple ci-dessus, l’utilisateur sait instantanément qu’il travaille avec l’identifiant **alice** et que le shell de ce terminal s’exécute sur la machine distante **pc-alice.boulot.fr**. Cela peut être particulièrement utile lorsque l’utilisateur est connecté à une machine distante (par ssh). Cela lui évite d’insérer des commandes dans le mauvais terminal.

Le répertoire courant indique dans quel répertoire sera exécuté la commande. Par exemple si vous demandez à lister le contenu d’un répertoire, il est préférable de savoir si on se trouve dans le bon répertoire. Cependant, le nom de répertoire est souvent abrégé pour éviter qu’il occupe trop d’espace sur la ligne de commande. Ainsi, le répertoire personnel de l’utilisateur (en anglais home directory) est habituellement abrégé par le caractère tilde noté ’~’.

L’invite de commande peut se terminer par l’indication du type d’utilisateur. S’il s’agit d’un utilisateur normal l’invite se terminera par le caractère dollar $. S’il s’agit du super-utilisateur (son identifiant est root), la commande se terminera par le caractère ’#’. Cela permet de savoir en un coup d’œil dans quel contexte sera exécutée la commande. Cette désambiguïsation réduit les chances de se tromper.

Par exemple, l’invite de commande ci-dessous nous indique que la commande sera exécutée en tant qu’utilisateur tonton, sur la machine ayant pour nom d’hôte portable1, dans le répertoire personnel

de tonton. La présence du suffixe $ nous rappelle que la commande sera exécutée avec les droits de l'utilisateur tonton.

```
tonton@portable1:~$
```

L'invite de commande ci-dessous nous indique que la commande sera exécutée en tant que super-utilisateur, sur la machine ayant pour nom d'hôte portable1, dans le répertoire /tmp/abc/def/ . La présence du suffixe # nous rappelle que la commande sera exécutée avec les droits du super-utilisateur.

```
root@portable1:/tmp/abc/def/#
```

5 Commande

Comme nous l'avons vu, le shell est une application qui sert d'interface entre le noyau et l'utilisateur. Il sert à exécuter des commandes qui proviennent d'un terminal (mode interactif) ou d'un fichier (mode script). Ces commandes peuvent être internes au shell ou externes au shell 1 . Les commandes externes font appel à des programmes séparés du shell tandis que les commandes internes sont exécutées par le shell lui-même.

Il est possible de savoir de quel type est une commande. La commande interne type suivie du nom d'une commande sert à indiquer le type de la commande.

```
$ type echo cal date
echo is a shell builtin
cal is /usr/bin/cal
date is hashed (/bin/date)
```

Ici, type nous apprend que la commande echo est de type built-in, c'est-à-dire une commande interne.

La commande cal est une commande externe dont le programme est dans le répertoire /usr/bin/cal .

La commande date est également une commande externe mais type nous indique que cette dernière est « hashée », c'est-à-dire que la localisation du programme a été mise en mémoire afin d'accélérer le lancement de la commande.

5.1 Algorithme du shell

Une commande est un appel à un programme généralement. Après avoir saisi la ligne de commande, il suffit d'appuyer sur la touche de retour à la ligne pour lancer son exécution. Avant l'exécution, le shell effectue l'analyse de la ligne de commande pour y retrouver les éléments constitutifs. Nous reviendrons en détail sur cette étape dans la séquence 2. Pendant l'exécution, l'utilisateur peut être sollicité pour entrer des données supplémentaires. Une fois l'exécution de la commande terminée, le résultat s'affiche à l'écran et l'invite de commande s'affiche à nouveau, ce qui indique que le shell est prêt à recevoir une nouvelle ligne de commande.

5.2 Structure d’une commande

Une commande (interne ou externe) est constituée par des mots séparés par des espaces. Le format général d’une commande est le nom de la commande suivie d’arguments qui seront interprétés avec cette dernière. Les arguments sont passés avec l’appel du programme associé à la commande.

$ commande arg1 arg2 ... argn

Le nombre d’arguments dépend de la commande et de la tâche à effectuer par la commande. Par exemple la commande date peut être appelée sans aucun argument. Elle affichera alors la date :

```
$ date
lundi 7 septembre 2020, 15:38:13 (UTC+0200)
```

On peut spécifier à la commande date qu’on veut une date affichée comme le nombre de secondes écoulées depuis le 1er janvier 1970 :

```
psimier@b107PSR:~$ date +%s
1599485933
```

Parmi les commandes les plus simples, on retrouve également echo dont le rôle est simplement d’afficher les caractères qui lui sont passés en paramètres.

psimier@b107PSR:~$ echo "Bonjour le monde"

Bonjour le monde

La syntaxe attendue pour chaque argument est spécifique à chaque commande. Un argument peut être une option, il sera alors de la forme -x avec x la lettre identifiant l’option. Une lettre étant peu explicite, il est souvent possible d’identifier une option via un ou plusieurs mots. Elle sera alors préfixée de deux -- . Par exemple date -u et date --utc font références à la même option, --utc ayant l’avantage d’être plus explicite que -u .

```
$ date -u
lundi 7 septembre 2020, 13:41:27 (UTC+0000)
psimier@b107PSR:~$ date --utc
lundi 7 septembre 2020, 13:42:18 (UTC+0000)
```

Pour qu'elle ait un sens, une option doit parfois être suivie d'une valeur, appelée argument d'option. Cette valeur est spécifiée dans l'argument qui suit l'option. Par exemple on peut demander au programme cal (CALendar) d'afficher le calendrier du mois d'avril (april en anglais) via la commande suivante (option -m avec l'argument d'option apr) :

```
psimier@b107PSR:~$ cal -m apr
     Avril 2020
di lu ma me je ve sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

La valeur associée à l'option peut être spécifiée dans le même argument mais séparée de l'identifiant d'option via un caractère délimiteur. Par exemple, pour demander au programme date de modifier le format d'une date passée en argument, le caractère délimiteur entre l'option et sa valeur sera le '=' comme le montre la commande suivante :

```
psimier@b107PSR:~$ date --date="2004-02-29"
dimanche 29 février 2004, 00:00:00 (UTC+0100)
```

6 Conclusion

Dans cette activité vous avez découvert la ligne de commande. Vous connaissez maintenant le mode interactif du Bash. Lorsqu'une invite de commande s'affiche, cela signifie que vous pouvez entrer une ligne de commande. Une fois validée, elle est interprétée par le shell et vous devez attendre que la commande termine son exécution pour que l'invite de commande s'affiche à nouveau.

Nous avons également vu la structure d'une ligne de commande. Elle est composée d'un nom suivi d'options et d'arguments. Notez cependant que la commande a une syntaxe qui lui est propre. Il est donc indispensable d'obtenir de l'aide pour savoir l'utiliser.