1 Introduction

Le Shell Bash est désormais omniprésent, car il a été intégré à Windows 10/11.

Ainsi, dorénavant, les 3 grands systèmes d'exploitation pour ordinateur (Windows, MacOS

et Linux) partagent cet outil, 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.

1.1 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.

2 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

```
     Avril 2024
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
```

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 ».

En vous aidant de la commande "**man**", donnez la ligne de commande à taper afin d'afficher le calendrier du mois de mars de l'année 2000.

3 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.

dbernard@b107PCD:~$

Dans l'exemple ci-dessus, l'utilisateur sait instantanément qu'il travaille avec l'identifiant **dbernard** et que le Shell de ce terminal s'exécute sur la machine **b107PCD**.

Le répertoire courant indique dans quel répertoire sera exécutée la commande. 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 '**#**'.

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/#

Ouvrez un terminal et listez les informations suivantes en vous basant sur le prompt affiché:
+ Login
+ Nom de la machine
+ Répertoire courant
+ Privilège (utilisateur "classique" ou super-utilisateur)

4 Commandes

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 . 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
     echo est une primitive du Shell
type date
     date est /usr/bin/date
type cal
     cal est haché (/usr/bin/cal)
```

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

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

La commande **cal** 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.

Donnez le type de commande pour "ls", "ip" et "pwd".

4.1 Algorithme du Shell

Une commande est un appel à un programme.

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.

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.

4.2 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 :

```
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.

```
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)
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.

Affichez la date et la date au format universel.
Que constatez-vous ?
En vous aidant de la commande "**man**", afficher la date sous la forme : jour en toutes lettres jour du mois nom du mois en toutes lettres année (Exemple : mardi 2 septembre 2024).

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) :

```
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 :

```
date --date="2004-02-29"
     dimanche 29 février 2004, 00:00:00 (UTC+0100)
```

5 Activités

À l'aide de la commande **pwd**, donnez le chemin complet menant à votre répertoire personnel.
Créer un répertoire nommé "public_html" à la racine de votre répertoire personnel, à l'aide de la commande **mkdir**.

Les permissions relatives à un fichier ou répertoire sont divisées en 3 niveaux (utilisateur, groupe, autres) qui ont chacun 3 types de permissions (lecture, ecriture, exécution).

En vous aidant de la commande **ls -l**, donnez les informations suivantes pour le répertoire "public_html" que vous venez de créer:
+ propriétaire
+ groupe
+ permission du propriétaire
+ permission du groupe
+ permission des autres

Les permissions se fixe à l'aide de la commande chmod ayant pour syntaxe:

chmod permissions nom du fichier ou du répertoire.

Les permissions sont représentées par des lettres (r pour lecture, w pour écriture, x pour exécution) ou des chiffres (4 pour lecture, 2 pour écriture, 1 pour exécution).

Type	Utilisateur	Groupe	Autres
Lecture (r)	4	4	4
Ecriture (w)	2	2	2
Exécution (x)	1	1	1

Par exemple:

```
chmod 754 bidon.txt
```

Cette commande signifie que pour le fichier bidon.txt, l'utilisateur à des droits en lecture, ecriture et execution (4+2+1), que le groupe à des droits en lecture et exécution (4+1) et que les autres ont des droits en lecture (4).

Il est possible d'obtenir le même résultat avec la syntaxe suivante:

```
chmod u=rwx,g=rx,o=r bidon.txt
```

Modifier les permissions du répertoire "public_html" afin que le propriétaire ait tout les droits, le groupe aucun droit et les autres des droits en lecture et exécution.

Il est possible de changer le propriétaire d'un fichier ou d'un répertoire à l'aide de la commande chown.

En vous aidant de la documentation de la commande **chown** via la commande "**man**", essayez d'attribuer le répertoire "public_html" à l'utilisateur "toto".
Quel message est affiché ? Pourquoi ?

Il est important de savoir si le système a suffisamment d'espace disque pour travailler correctement.

À l'aide d'une ligne de commande, donnez le taux d'utilisation des disques utilisables sur votre machine.
Donnez la capacité encore utilisable et celle utilisée dans un format humainement compréhensible (aidez-vous de la documentation de la commande pour savoir quelle option utiliser avec cette dernière).

La commande **du** (disk usage) affiche l'espace disque utilisé par chaque sous-répertoire du répertoire courant.

À l'aide de cette commande, donner la taille qu'occupe votre répertoire personnel sur la disque (dans un format humainement lisible).
Même question pour le répertoire /home/USERS/ELEVES/CIEL2024.

6 Interragissez avec le Bash

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.

Le fonctionnement du Shell suit ces étapes :

1. La saisie de ligne de commande devient possible lorsque l'invite de commande apparaît.
2. La saisie est aidée par les fonctions d'édition.
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.
4. Puis, l'interprétation de ces constructions est effectuée afin de les substituer par leur résultat.
5. Ensuite, l'entrée et la sortie de la commande sont définies.
6. Finalement, la commande avec ses arguments substitués est exécutée par l'appel du programme correspondant.

6.1 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.

6.1.1 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

Alt+f avance d'un mot

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

```
history 3
 1964  nslookup 172.18.58.51
 1965  nslookup 172.18.58.189
 1966  ipcalc 172.18.58.0/23
```

Les 3 dernières commandes sont nslookup (2 fois) et ipcalc.

Il y avait 1963 commandes déjà enregistrées avant celles-ci.

Donnez les 5 dernières commandes de l'historique des commandes sur votre session.
Combien de commandes ont été enregistrées ?

6.1.3 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.

Quelle est la dernière commande ayant été saisie dans votre terminal ?

6.1.4 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 réexécuter 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
```

6.2 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'autocomplétion. 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[tab][tab]
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.

6.3 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

6.4 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.

6.4.1 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 existe 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
```

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$
```

Afficher les chemins connus par votre Shell.

6.5 Votre environnement de travail

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

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
```

6.5.1 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.

Créez une variable nommée age et qui prendra comme valeur, votre age.
Créez une variable nommée prenom et qui prendra comme valeur, votre prénom.
Créez une variable nommée temps et qui prendra comme valeur, "il fait beau".
Afficher les valeurs de ses 3 variables.

La commande **`unset`** permet 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:~$
```

Supprimez les 3 variables précédentes.
Vérifier qu'elles n'existent plus.

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 et les variables locales.