Les notions importantes pour bien débuter la programmation SHELL

1. Les différents type de Shell

Le Shell est un programme, un composant du système, et il existe plusieurs shells pour les systèmes UNIX. Chacun met à disposition de l'utilisateur des commandes internes et des fichiers d'initialisation (qui permettent de paramétrer l'environnement de travail). La plupart des shells présentent des fonctionnalités communes car sont issus d'un ancêtre commun : le Bourne shell.

Le shell est un interpréteur de commandes, chargé de traduire et d'exécuter ce que vous saisissez dans une interface (le « terminal » ou la « console »). C'est en quelque sorte la « coquille » qui entoure le noyau du système. (En anglais shell signifie coquille).

Développé à la fin des années 70 par Steve Bourne, le Bourne Shell (abrégé en « sh ») permet à l'époque de lancer des commandes complexes (grâce à son système de redirection et de « tubes »), mais il est bien plus qu'un simple interpréteur car il a été conçu comme un véritable langage de programmation. On le trouve aujourd'hui sur la plupart des systèmes Unix.

Parallèlement, Bill Joy crée le C shell (« csh »), peu utilisé actuellement. Celui-ci est incompatible avec le shell Bourne, il inclut toutefois des fonctionnalités supplémentaires particulièrement intéressantes : un historique de commandes, la création d'alias de commandes ou encore le contrôle des tâches.

Ces fonctionnalités seront plus tard reprises par David Korn, qui crée ainsi un nouvel interpréteur, un Bourne shell amélioré baptisé Korn Shell (« ksh »). Nous sommes alors en 1983. Le Korn Shell a servi de base à la normalisation du shell.

La même année, Ken Greer propose le TC shell (« tcsh ») basé sur le C shell et compatible avec ce dernier : le « T » désigne le TENEX, un système d'exploitation dont s'est inspiré Greer. Le tcsh apporte des fonctionnalités essentielles, comme la gestion d'un historique de commandes, l'auto-complétion des noms de fichiers.

Actuellement, l'interpréteur que l'on rencontre le plus souvent sur les systèmes GNU/Linux et qui combine tous les atouts de ces prédécesseurs est le shell Bash (Bourne Again Shell), publié pour la première fois en 1989 par Brian Fox pour le compte de la Free Software Foundation. Le shell que nous allons utilisé sera celui-ci.

La Free Software Foundation est une organisation américaine à but non lucratif fondée par Richard Stallman en octobre 1985, dont le but est de promouvoir le logiciel libre et défendre ses utilisateurs. La FSF aide également au financement du projet GNU depuis sont origine.

A noter néanmoins que beaucoup délaissent le Bash au profit du Z shell (« zsh ») dont la première version a été écrite en 1990 par Paul Falstad, un étudiant de l'université de Princeton.

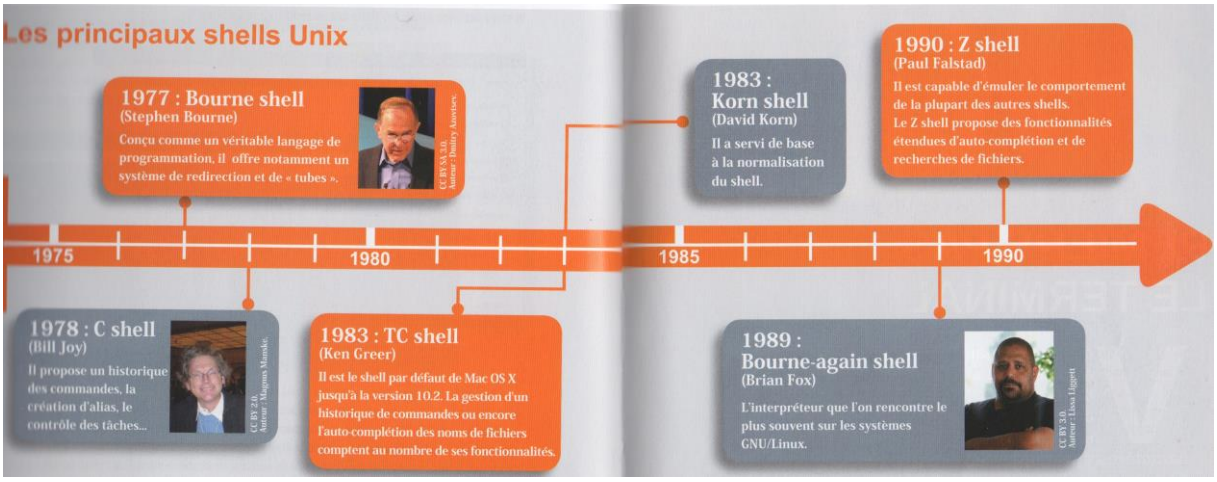

Figure 1: Les principaux shells

Quels sont les types de shell dont je dispose ?

```
$ more /etc/shells
```

Quel est le shell que j'utilise ?

```
$ echo $SHELL
```

Passer d'un shell à un autre

```
$ zsh
$ exit
```

2. Le système de fichier sous linux

/	Répertoire « racine », point d'entrée du système de fichiers
/bin	Répertoire contenant les exécutables de base, comme par exemple cp, mv, ls, etc.
/boot	Répertoire contenant les fichiers nécessaires au démarrage (Grub, Lilo) et le noyau linux
/dev	Répertoire contenant des fichiers spéciaux nommés devices qui permettent le lien avec les périphériques de la machine
/,etc	Répertoire contenant les fichiers de configuration du système
/home	Répertoire contenant les fichiers personnels des utilisateurs (un sous-répertoire par utilisateur)
/lib	Répertoire contenant les bibliothèques et les modules du noyau (/lib/modules)
/mnt	Contient les points de montage temporaires des systèmes de fichiers
/opt	Répertoire permettant l'installation d'applications extérieures
/root	Répertoire personnel de l'administrateur
/sbin	Répertoire contenant les exécutables destinés à l'administration du système
/tmp	Répertoire contenant des fichiers temporaires utilisés par certains programmes
/usr	Répertoire contenant les exécutables des programmes (/usr/bin et /usr/sbin), la documentation (/usr/doc), et les programmes pour le serveur graphique (/usr/X11R6)
/var	Répertoire contenant les fichiers qui servent à la maintenance du système (les fichiers de journaux notamment dans /var/log)
/media	Répertoire contenant les « points de montage » des médias usuels : CD, DVD, disquette, clef USB...
/dev/null	On peut envoyer une infinité de données à ce périphérique qui les ignorera...
/dev/zero	On peut lire une infinité de zéros depuis ce périphérique
/dev/random	On peut lire des nombres aléatoires depuis ce périphérique

3. Parler le Bash

Pour commencer, chaque shell est doté d'un fichier de configuration dans lequel sont définis son comportement, son apparence et bien d'autres options. Chaque utilisateur peut personnaliser son shell suivant ses préférences.

Nous aborderons la configuration des fichiers de démarrage du shell Bash dans un autre TP mais voici un aperçu :

Les fichiers /etc/profile et ~/.bash_profile sont lus quand le shell est appelé en tant que shell interactif de connexion.

Le fichier ~/.bashrc est lu quand le shell est invoqué comme shell interactif sans fonction de connexion.

Pour s 'en convaincre et vérifier l'ordre d'appel, insérez des messages dans chacun de ces fichiers :

```
echo « Lecture du fichier /etc/profile »
```

4. Le prompt

Le prompt est l'invite de commande qui vous signale que le système attend la saisie au clavier d'une commande. Par défaut, sur de nombreuses distributions, le prompt est constitué de votre nom d'utilisateur, suivi du nom de la machine. Suit le caractère « ~ » qui indique que vous vous trouvez dans votre répertoire personnel. Généralement, le caractère « $ » indique que vous agissez en tant qu'utilisateur normal ; si vous êtes connecté en tant que root (l'administrateur du système), ce caractère est remplacé par un #.

Vous pouvez très bien personnaliser votre prompt. En effet la valeur du prompt est stockée dans une variable « PS1 », que vous pouvez afficher via la commande :

```
echo $PS1
```

Caractère spécial	Signification
\u	Nom d'utilisateur
\h	Nom de la machine
\w	Répertoire courant depuis le répertoire /home
\d	La date
\n	Saut de ligne
\t	L'heure au format HH :MM :SS

Pour modifier la valeur du prompt il suffit de saisir PS1='nouveau motif'.

Néanmoins ceci ne durera que le temps de la session : dès que vous fermerez votre terminal, vous perdrez la modification. Pour modifier durablement l'apparence de votre prompt, il faudra modifier la variable PS1 dans le fichier .bashrc.

Les différents prompts :

Prompt	Signification
PS1	Le prompt principal
PS2	Le prompt secondaire (lorsqu'on saute une ligne dans le terminal)
PS3	Le prompt de la commande select
PS4	Le prompt affiché en mode debug

Ajout de la couleur au prompt :

```
$ export PS1="\e[0;31m[\u@\h \W]\$ \e[m "
```

Liste de couleur:

Black: 0;30

Blue: 0;34

Green: 0;32

Cyan : 0;36

Red : 0;31

Purple : 0;35

Brown ; 0;34

5. Rappel sur les commandes et les fichiers

Les commandes

Une commande s'applique généralement à un ou plusieurs fichier(s) ou répertoire(s). Pour lancer une commande on respectera la syntaxe suivante :

```
Commande [options] argument(s)
```

Un doute sur l'utilisation d'une commande, un oubli concernant une option ? Dans ce cas, consultez le manuel des commandes ! La commande « man », suivie d'un nom de commande permet d'accéder à la page du manuel Unix s'y rapportant. Pour sortir du manuel, on tape simplement sur la commande « q ».

La bible: man

```
man [-s<section>] <nom de la commande>
```

Les sections:

1. Commandes utilisateur
2. Appels système
3. Fonctions de bibliothèque
4. Fichiers spéciaux
5. Formats de fichier
6. Jeux
7. Divers
8. Administration système
9. Interface du noyau Linux

Ecrire une page man:

```
.TH CORRUPT 1
.SH NAME
corrupt \- modify files by randomly changing bits
.SH SYNOPSIS
.B corrupt
[\fB\-n\fR \fIBITS\fR]
[\fB\-\-bits\fR \fIBITS\fR]
.IR file ...
.SH DESCRIPTION
.B corrupt
modifies files by toggling a randomly chosen bit.
.SH OPTIONS
.TP
.BR \-n ", " \-\-bits =\fIBITS\fR
Set the number of bits to modify.
Default is one bit.
```

```
man –l corrupt.1
```

Copier dans /usr/share/man/man1

Les fichiers

Chaque fichier et répertoire est caractérisé par un ensemble de propriétés. Ces dernières peuvent être visualisées via la commande :

```
ls -l
```

Vous obtenez ainsi diverses informations, dans cet ordre :

- Le type de fichier et les permissions qui sont octroyées aux différents utilisateurs du système,
- Le nombre de liens pointant vers le fichier (ou répertoire)
- Le propriétaire du fichier
- Le groupe auquel il appartient,
- La taille du fichier (en octets)
- La date et l'heure de la dernière modification
- Le nom du fichier ou répertoire

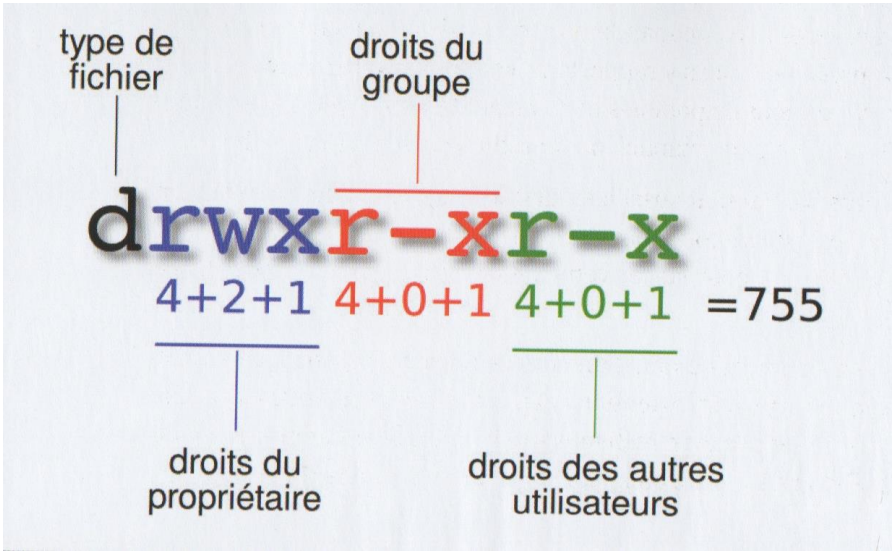

Figure 2: Les permissions.

6. Les fonctionnalités qui simplifient la vie

La complétion automatique

Le shell bash propose de compléter automatiquement les commandes que vous saisissez. Vous entrez les premières lettres d'une commande et vous appuyez sur la touche [TAB]. Le shell complète alors automatiquement la saisie ou affiche la liste des possibilités s'il y en a plusieurs.

La fonctionnalité d'autocomplétion de votre shell est possible grâce à cette instruction contenue dans le fichier de configuration de Bash, à savoir le fichier /home/user/.bashrc.

```
if [ -f /etc/bash_completion ] ; then
      . /etc/bash_completion
fi
```

Le fichier /etc/bash_completion est fourni par le paquet bash-completion.

Pour installer ce paquet sur CentOS:

```
yum install bash-completion
```

(Sur Ubuntu ce paquet doit être installé par défaut sinon :

apt-get install bash-completion)

L'historique des commandes

Le shell a très bonne mémoire ! Il conserve toutes les commandes que vous avez saisies dans un fichier : /home/user/.bash_history

La commande history permet de lister tout l'historique ; Les commandes sont identifiées par un numéro.

Vous pouvez appeler une commande en saisissant « ! » suivi du numéro de la dernière commande. « !! » appelle la dernière commande utilisée.

Vous pouvez personnaliser ce comportement en modifiant le fichier .bashrc :

Ne pas conserver les doublons :

```
export HISTCONTROL=ignoredups
```

Ne pas conserver les commandes qui commencent par le caractère espace

```
export HISTCONTROL=ignorespace
```

Ajouter la date et l'heure devant chaque commande :

```
export HISTTIMEFORMAT= « %D %H :%M »
```

Pour définir le nombre maximal de commande sauvegardées

```
export HISTFILESIZE=1000
```

Pour conserver les commandes d'une session à une autre :

```
shopt -s histappend
```

Les alias de commandes

Les alias sont des raccourcis équivalents à des commandes. En général, on crée des alias pour les commandes que l'on utilise fréquemment.

Pour créer un nouvel alias, on lance la commande en respectant la syntaxe suivante :

```
alias <nom du raccourci>='<commande à remplacer>'
```

Exemple :

```
alias la='ls -la'
```

La commande alias utilisée toute seule vous donne la liste des alias.

Les raccourcis pour naviguer dans le shell

RACCOURCI	ACTION
CTRL+P ou ↑	Rappelle la commande précédente
CTRL+N ou ↑	Rappelle la commande suivante
CTRL+A ou Home	Va en début de ligne
CTRL+E ou End	Va en fin de ligne
CTRL+D ou Suppr	Supprime le caractère qui se trouve sous le curseur
CTRL+K	Supprime tout ce qui se trouve à droite du curseur
CTRL+U	Supprime tout ce qui se trouve à gauche du curseur
CTRL+L	Efface l'écran
CTRL+R	Lance une recherche dans l'historique : on saisit un motif et la plus récente commande qui le contient apparait (une succession de CTRL+R permet de remonter dans l'historique.
CTRL+D	Ferme la session Bash.

7. Le contexte : quelques concepts de base

Commandes internes et externes

On distingue 2 types de commandes acceptées par le shell :

Les commandes dites internes sont les commandes qui sont exécutées par le shell lui-même (cd, pwd, type,…)

Les commandes dites externes sont des commandes dont le code est situé dans un fichier exécutable, stocké dans le système de fichiers (ls, cp, mv, …). Ainsi lorsque vous saisissez la commande ls, dans le terminal, le shell demande au noyau de charger le fichier /usr/bin/ls ; un nouveau processus est donc créé. Naturellement le shell doit connaître l'emplacement du code d'une commande externe pour pouvoir l'exécuter. Pour cela, il utilise la variable PATH qui contient la liste des répertoires où sont localisés les exécutables.

La commande « type » vous indique si une commande est interne ou externe.

Les variables d'environnement

Dès votre connexion au système, le shell met à disposition des variables spécifiques appelées « variable d'environnement ». Elles contiennent des informations de base dont le shell a besoin pour fonctionner.

Pour afficher la liste des variables définies au niveau du shell, on utilise la commande « set ».

Par défaut, les variables définies au niveau du shell, ne sont pas transmises aux commandes lancées à partir de celui-ci : le shell doit d'abord les exporter. Toutefois, certaines variables sont exportées par défaut : on utlise la commande « env » pour en visualiser la liste.

La commande locale permet de visualiser les variables concernant la langue et l'encodage.

Un certain nombre de variables prédéfinies jouent un rôle essentiel pour l'utilisateur. Leur valeur peut être définie par défaut dès l'installation du système ou être initialisée à tout moment par l'utilisateur :

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

Pour modifier une variable d'environnement et l'exporter, on utilisera la commande « export ». Une variable qu'il est indispensable de modifier est la variable « PATH ».

En effet la variable PATH contient une liste de répertoires qui sont explorés par le shell lorsqu'une commande est lancée.

```
~$ echo $PATH
/usr/bin:/bin:/usr/local/bin
```

Les répertoires sont explorés dans l'ordre dans lequel ils sont affichés.

Pour qu'un nouveau chemin apparaisse au début :

```
export PATH=/nouveau/chemin :$PATH
```

A la fin :

```
export PATH=$PATH:/nouveau/chemin
```

La gestion des processus

Nous allons ici faire un rappel sur la gestion des processus :

Pour récupérer la main après avoir lancé une commande on utilisera le caractère « & ».

Le système Unix attribue un numéro à chaque processus, appelé identifiant du processus ou « PID ».

Toutes les informations relative à un processus se trouve dans le système de fichier « /proc »

Pour ramener un job (processus) au premier plan, on utilise la commande « fg » (pour foreground).

Si plusieurs processus s'exécutent en arrière plan on utilisera « fg %n » ou n est le numéro du processus lancé en arrière plan.

Inversement la commande « bg » pour background).

Pour afficher les processus en cours d'exécution : « jobs –l »

Pour lister l'ensemble des processus en cours d'exécution sur le système : « ps –ef ».

Pour envoyer un signal à un processus : kill .

Les principaux signaux sont les suivants :

1	SIGHUP	Les fichiers de configuration du processus sont rechargés
9	SIGKILL	Le processus est brutalement tué par le système
15	SIGTERM	Equivalent au CTRL C. Le processus se termine proprement en sauvegardant des fichiers si nécessaire.
19	SIGSTOP	Equivalent au CTRL Z. On suspend l'exécution des processus.

8. Des caractères spéciaux

La signalisation

La redirection

Par défaut le résultat d'une commande est dirigé vers la sortie standard du système à savoir votre écran. S'il y a une sortie standard, il y a également une entrée standard ; il s'agit de votre clavier.

On utilise les redirections lorsque l'on ne souhaite pas que le retour d'une commande aille directement vers la sortie standard. On peut par exemple envoyer le résultat d'une commande dans un fichier, vers une imprimante ou tout autre périphérique.

>	`ls > fichier.txt`	Permet de rediriger le résultat de la commande dans un fichier
>>	`date >> fichier.txt`	Permet de rediriger le résultat de la commande vers la fin du fichier
2>	`make 2> erreur.txt`	Perler de ne rediriger que les messages d'erreur
<	`wc < fichier.txt`	La commande prend en entrée le contenu du fichier
|	`ps –ef | grep gedit`	Le pipe, permet d'utiliser le résultat de la première commande comme entrée standard de la commande qui suit

L'enchainement de commandes

;	mkdir images ; mv *.png images ; cd images	Permet d'exécuter plusieurs commandes les unes à la suite des autres
&	skype & gedit & firefox	Permet d'exécuter les commandes simultanément

L'enchainement conditionnel

||	`ps –ef | grep trucdeouf || echo « n'importe quoi »`	la 2ème commande ne s'exécutera que si la 1ère commande ne retourne rien ou une erreur
&&	`make && make install`	La 2ème commande ne s'exécute que si la 1ère s'est achevée sans erreur

La substitution de commande

La substitution de commande permet de positionner toute une commande – dont on ne connaît pas le résultat- comme paramètre d'entrée d'une autre.

$(commande)

'commande'

```
~$ kill 'pidof gedit'
```

Les expressions régulières

Expressions régulières	Signification
Linux$	La chaîne correspondante (de longueur quelconque) se termine par « Linux ».
^$	La chaîne correspondante est constituée d'un début de ligne, immédiatement suivi d'une fin de ligne
^[A-Z][3-6].$	La chaîne correspondante est composée de trois caractères : le premier est une lettre majuscule, le deuxième un chiffre compris entre 3 et 6 et le troisième est un caractère quelconque
[0-9][^A-Z?]$	La chaîne correspondante (de longueur quelconque) comporte un chiffre en tant qu'avant-dernier caractère et le dernier caractère n'est ni une lettre majuscule, ni un point d'interrogation.
8 [357]y*	La chaîne correspondante (de longueur quelconque) comporte le chiffre 8, suivi d'un espace, puis d'un chiffre parmi 3,5 et 7, puis de la lettre 'y' répétée 0 à n fois.