# CHAPITRE 2

## MANIPULER LE SHELL LINUX


**Ce que vous allez apprendre dans ce chapitre :**

- Assurer la gestion de base du système de fichiers
- Gérer les droits d'accès aux utilisateurs
- Assurer la gestion de processus et redirection des flux
- Maîtriser la Programmation Shell

**30 heures**

# CHAPITRE 2

# MANIPULER LE SHELL LINUX

1. **Assurer la gestion de base du système de fichiers**
2. Gérer les droits d’accès aux utilisateurs
3. Assurer la gestion de processus et redirection des flux
4. Maîtriser la Programmation Shell

## Assurer la gestion de base du système de fichiers

Afin de permettre à l'utilisateur la gestion du système de fichier, Linux propose un ensemble de commandes de base permettant la manipulation des fichiers décrits dans la Table7, 8 et 9.

| Fonctionnalité | Syntaxe |
|---|---|
| Affichage de répertoire courant | pwd |
| Copie d'un fichier | cp fich-orig fich-dest Ou<br><br>cp fichier repertoire |
| Copie des répertoires entiers | cp –r rep-origine rep-destination ou<br><br>rsync –a rep-origine/ rep-destination/ |
| Renommer fichier1 en fichier2 (idem pour répertoire) | mv fichier1 fichier2 |

*Table 7 : Commande de base (1)*

## Assurer la gestion de base du système de fichiers

On vous invite à tester ces commandes dans votre terminal sous linux.

| Fonctionnalité | Syntaxe |
|---|---|
| Déplacer fichier1 dans le répertoire /tmp | mv fichier1 /tmp |
| Déplace le répertoire TEST dans le répertoire /tmp | mv TEST /tmp |
| Supprimer des fichiers | rm fichier1 fichier2 |
| Supprimer un répertoire vide | rmdir rep |

*Table 8 : Commande de base (2)*

## Assurer la gestion de base du système de fichiers

On a pas fini. Il existe encore plusieurs commandes.

| Fonctionnalité | Syntaxe |
|---|---|
| Supprimer un répertoire non vide | rm –rf rep |
| Lister des fichiers du dossier courant | ls |
| Afficher une liste détaillée | ls -l |
| Lister tous les fichiers (cachés) | ls -a |
| Trier les fichiers par date (plus récents sont les premiers) | ls -t |
| Trier les fichiers par taille | ls -S |
| Changer rep | cd |
| Créer un rep | mkdir rep |
| Crée une arborscence rep_parent: dans rep_parent , crée rep_enfant | mkdir –p rep1/rep2 |

*Table 9 : Commande de base (3)*

## Assurer la gestion de base du système de fichiers

**Administrateur Linux :**

Sous Redhat, l'administrateur Linux est l'utilisateur root.

- Il dispose de toutes les permissions.
- Il se caractérise par l'ID 0.

Certaines distributions Linux désactivent le compte root. Mais, il est toujours possible de se connecter en tant que root.
Linux permet donc aux utilisateurs de changer l'utilisateur.

**Changer d'utilisateur :**

La commande su (switch user) permet d'ouvrir un shell en tant qu'utilisateur quelconque :

- Lancer un shell en tant qu'utilisateur **user1** :

```
su user1
```

- Ouvrir un shell en tant qu'utilisateur **user1** et charger son profil :

```
su – user1
```

- Ouvrir un shell en tant qu'utilisateur **root** et charger son profil utilisateur :

```
su –
```

## Assurer la gestion de base du système de fichiers

**Faire des opérations en tant que root :**

La commande sudo (substitute user do) permet d'exécuter des commandes en tant qu'administrateur (root).

Sous Red hat Linux, les utilisateurs doivent appartenir au groupe **wheel** pour pouvoir utiliser la commande sudo.

```
usermod –G wheel user1
sudo ma-commande
```

Sous Debian Linux, les utilisateurs doivent appartenir au groupe **sudo** pour pouvoir utiliser la commande sudo.

```
usermod –G sudo user1
sudo ma-commande
```

La configuration de sudo s'effectue dans le fichier /etc/sudoers.

Ce fichier est modifiable par le root avec la commande visudo.

## Assurer la gestion de base du système de fichiers

Un utilisateur = un compte.

Un utilisateur est identifié par un UID et un GID.

- UID = User Identifiant
- GID = Groupe Identifiant

Les utilisateurs sont stockés dans le fichier /etc/passwd.

Les groupes sont stockés dans le fichier /etc/group.

Les commandes useradd et groupadd permettent de créer des utilisateurs et des groupes.
Fichier contenant les utilisateurs : /etc/passwd.
Ce fichier contient plusieurs champs séparés par **:** Nom utilisateur - UID = User Identifiant - GID = Groupe Identifiant – Commentaire - Le répertoire personnel -Le shell de connexion.

## Assurer la gestion de base du système de fichiers

Commande d'ajout d'un utilisateur : useradd.

Modification des options par défaut lors de la création des utilisateurs (Voir la Table 10) :

| Option | Interprétation |
|---|---|
| -b, --base-dir REP_BASE | Répertoire personnel du compte du nouvel utilisateur. |
| -c, --comment COMMENTAIRE | Définir le champ commentaire. |
| -g, --gid GROUPE | Forcer l'utilisation de GROUPE pour le compte du nouvel utilisateur. |
| -u, --uid UID | Forcer l'utilisation de l'identifiant. « UID » pour le compte du nouvel utilisateur. |
| -s, --shell INTERPRÉTEUR | Interpréteur de commandes initial pour le compte du nouvel utilisateur. |

*Table 10: Commande useradd et ses options*

## Assurer la gestion de base du système de fichiers

Commande de modification d'un utilisateur : usermod

Modification des options par défaut lors de la création des utilisateurs (voir la table 11) :

| Option | Interprétation |
|---|---|
| -c, --comment COMMENT | définir une nouvelle valeur pour le champ commentaire. |
| -d, --home REP_PERS | définir un nouveau répertoire personnel pour le compte de l'utilisateur. |
| -g, --gid GROUPE | forcer l'utilisation de GROUPE comme nouveau groupe primaire. |
| -G, --groups GROUPES | définir une nouvelle liste de groupes supplémentaires. |
| -s, --shell INTERPRÉTEUR | nouvel interpréteur de commandes initial pour le compte de l'utilisateur. |

*Table 11 : Commande usermod et ses options*

## Assurer la gestion de base du système de fichiers

Suppression d'un utilisateur :

```
userdel user1
```

Suppression d'un utilisateur et suppression du répertoire personnel :

```
userdel -r user1
```

Profil de l'utilisateur :
Le fichier /etc/default/useradd : contient les options de création par défaut.
Le fichier /etc/login.defs : contient des paramètres de création du compte utilisateur.
Le répertoire /etc/skel : modèle de création des répertoires personnels.

## Assurer la gestion de base du système de fichiers

**Les groupes :**

- Un groupe est identifié par un GID.
- GID = Groupe Identifiant.
- Les groupes sont stockés dans le fichier /etc/group ;

Les commandes groupadd, groupdel, groups permettent de manipuler les groupes.

Les groupes sont stockés dans le fichier /etc/group.

La commande groups permet de lister les utilisateurs d'un groupe :

Le fichier est formé par plusieurs champs séparés par deux points :

- Nom du groupe ;
- GID ;
- Utilisateurs du groupe.

# CHAPITRE 2

## MANIPULER LE SHELL LINUX

1. Assurer la gestion de base du système de fichiers
2. **Gérer les Droits d’accès aux utilisateurs**
3. Assurer la gestion de processus et redirection des flux
4. Maîtriser la Programmation Shell

## Gérer les Droits d 'accès et utilisateurs

Linux permet la gestion des droits d'accès aux utilisateurs. Voici une présentation sur les informations concernant le mot de passe de l'utilisateur.

```
1user03:2$6$CSsX...output omitted...:317933:40:599999:67:72:818113:9
```

1. Nom d'utilisateur du compte.
2. Le mot de passe crypté de l'utilisateur.
3. Date de la dernière modification (en nombre de jours depuis le 1er janvier 1970). (chage –d)
4. Le nombre minimum de jours qui doivent s'écouler depuis le dernier changement de mot de passe avant que l'utilisateur ne puisse le modifier à nouveau. (chage –m)
5. Le nombre maximum de jours qui peuvent s'écouler sans modification du mot de passe avant l'expiration du mot de passe. (Chage –M)
6. Nombre de jours durant lesquels l'utilisateur est prévenu de l'expiration de son mot de passe. (chage –W)
7. Période d'inactivité. Une fois le mot de passe expiré, il sera toujours accepté pour la connexion, et ce pour plusieurs jours. Une fois cette période écoulée, le compte sera verrouillé. (Chage –I)
8. Le jour où le mot de passe expire. Ceci est défini en nombre de jours depuis le 1970-01-01 . Un champ vide signifie qu'il n'expire pas à une date particulière. (Chage –E)
9. Le dernier champ est généralement vide et est réservé pour une utilisation future.

Linux permet aussi de gérer les droits d'accès entre fichiers et répertoires.

**Les droits d'accès pour un fichier :**

- r : Lecture (afficher) ;
- w : Ecriture (modification) ;
- x : Exécution (exécution d'un script).

**Les droits d'accès pour un répertoire :**

- r : Lire le contenu, lister les fichiers (avec ls par exemple) ;
- w : Modifier le contenu, créer et supprimer des fichiers (avec les commandes « cp », « mv », « rm ») ;
- x : Permet d'accéder aux fichiers du répertoire. Mais aussi de naviguer dans les sous-répertoires (avec « cd ») ;
- En général, le droit w est souvent associé au droit x.

La commande « chmod » offerte par Linux permet de modifier les droits :

Il existe deux 2 syntaxes différentes :

**Mode symbolique :**

- Basé sur des symboles (ugoa) et des opérateurs (+,-,=) ;
- u (user), g (group), o (others), a (all users) ;
- + (Ajouter le droit), - (Retirer le droit), = (Ajouter le droit et retirer tous les autres).

Exemple (Ajoute le droit d'exécution au propriétaire) :

**Mode octal :**

- Basé sur des nombres de 0 à 7.
- A chaque bit de la traduction binaire correspond un droit.

Exemple (rw- rw- r--)

# CHAPITRE 2

## MANIPULER LE SHELL LINUX

1. Assurer la gestion de base du système de fichiers
2. Gérer les Droits d’accès aux utilisateurs
3. **Assurer la gestion de processus et redirection des flux**
4. Maîtriser la Programmation Shell

Linux permet aux utilisateur de gérer les processus ainsi que la redirection des flux. On commence par la gestion des processus. Pour ce faire, on vous rappelle la définition d'un processus.

**Définition d'un processus :** c'est l'entité qui correspond à l'exécution d'un programme. Un processus est défini par :

- un espace mémoire contenant le code, les données et la pile d'instruction ;
- un compteur ordinal (zone mémoire qui pointe sur l'instruction en cours) ;
-un ensemble de registres ( zones mémoire à accès rapide situées au niveau du processeur).

**Informations liés au processus :**

- UID : nom de propriétaire qui a lancé le processus (user, root, ...) ;
- PID : numéro du processus ;
- PPID : numéro du processus père (créateur).

**Commandes de gestion de processus :** ps, top, …

La Table 12 présente la commande ps et ses options.

| Option | Fonction |
|---|---|
| Sans options | processus en exécution. |
| -u user | processus en exécution pour l'utilisateur user. |
| -ef | informations complètes concernant les processus en cours d'exécution. |
| -x | processus actifs de l'utilisateur courant. |
| -ax | processus de tous les utilisateurs. |
| -p | informations sur le processus PID. |
| -l | afficher des informations assez complètes. |
| -c | afficher les commandes exécutées. |

*Table 12: Commande ps et ses options*

La Table 13 présente la commande Top et ses options.

| Option | Rôle |
|---|---|
| Sans options | Table des processus qui se met à jour d'une manière continuelle. |
| -d | Configuration de délai de rafraichissement. |
| -n | Affichage des processus en arrière plan. |

*Table 13 : Commande Top et ses options*

Linux nous permet la définition de la priorité à chaque processus, et permet sa modification à l'aide de la commande **nice** et **renice**.

Modification de la priorité : commande **nice**

- Valeur de la priorité :
  - Si simple utilisateur : entre 0 et 19
  - Si Super utilisateur : entre -20 et 19
  - Valeur par défaut = 0
  - Valeur de la priorité la plus basse = 19
  - Valeur de la priorité la plus haute = -20
- Modification de la valeur de nice à l'aide de la commande **renice.**

**Priorité du processus**

La Table 14 présente la commande nice et renice.

| Commande | Exemple | Explication |
|---|---|---|
| nice –n priorité commande | nice –n 19 find | Commande find avec la plus basse priorité. |
| renice priorité PID | renice −20 2535 | - Attribuer la priorité la plus haute au processus 2535.<br>- Seul le root peut attribuer cette priorité. |

*Table 14 : Commande nice et renice*

Linux nous permet également la gestion des signaux, qui sont des moyens utilisés pour communiquer avec les processus (Voir la Table 15).

Il y'a 64 signaux avec des identifiants différents :

- 0 : le seul signal qui n'a pas de nom ;
- 1 à 31 : signaux classiques ;
- 32 à 63 : signaux temps réels.

| Commande | Fonctionnement |
|---|---|
| Ctrl+Z | Un signal numéro 19 (SIGSTOP) est envoyé au processus en cours d'exécution. Ce qui stoppe son traitement. |
| Déconnexion | Envoi d'un SIGHUP (signal 1) à tous les processus. |
| Ctrl+C | Envoi d'un SIGINT) (signal 2) au processus courant. |

*Table 15 : Commande de gestion des signaux et ses options*

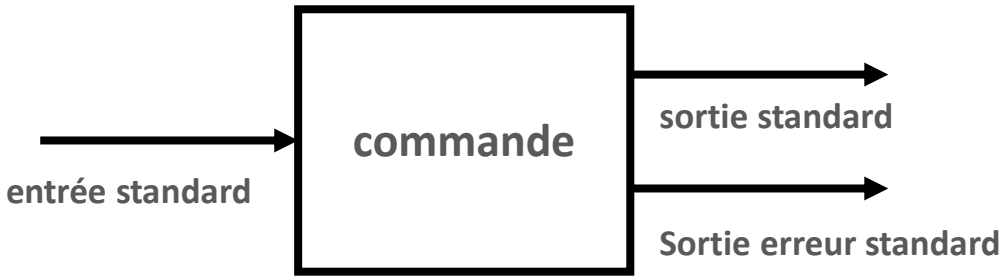
**Une commande :**

Entrée : arguments en entrée standard.

Sortie : une réponse en sortie standard

+

une réponse en sortie erreur standard




**La redirection :**

- Il s'agit de renvoyer le résultat d'une commande vers une sortie différente de la sortie normale.
- Elle correspond au détournement des 3 descripteurs de fichiers standards :

-l'entrée standard (notée 0) : le clavier ;

- la sortie standard (notée 1) : la console courante ;
- la sortie des erreurs (notée 2) : la console courante.

**Redirection de la sortie :**

Il s'agit de renvoyer le résultat d'une commande vers une sortie différente de la sortie normale .

- > et >> : renvoient le résultat vers un fichier.
  - **>** : rediriger dans un nouveau fichier.

```
ls > file1
```

Le contenu de fichier sera supprimé automatiquement si le fichier existe déjà.

  - **>>** : rediriger à la fin d'un fichier.

```
ls >> file1
```

Si le fichier n'existe pas, il sera créé. Sinon, les données seront ajoutées à la fin.

**< et << : lire depuis un fichier ou le clavier.**

< : permet d'indiquer d'où vient l'entrée d'où on envoie à la commande.

Exemple : Le fichier est lu par la commande cat.

```
#cat <  fichier1
```

<< : lire depuis le clavier progressivement.

Exemple : la commande cat lit du clavier jusqu'à ce qu'elle rencontre la chaine END et redirige le résultat dans le fichier fich.

```
#cat > fich<<END
```

### Redirection des erreurs : (2>, 2>> )

Pour chaque commande exécutée, il existe deux possibilités :

- Cas 1 : Tout va bien :

Le résultat de la commande sera affiché sur la sortie standard.

- Cas 2 : Une erreur se produit

Le résultat de la commande s'affiche dans la sortie d'erreurs.

### Syntaxe

2> : Rediriger les erreurs dans un fichier à part.

2>> : pour ajouter les erreurs à la fin du fichier.

# CHAPITRE 2

## MANIPULER LE SHELL LINUX

1. Assurer la gestion de base du système de fichiers
2. Gérer les Droits d’accès aux utilisateurs
3. Assurer la gestion de processus et redirection des flux
4. **Maîtriser la Programmation Shell**

Linux fournit un Shell et permet la programmation Shell (Voir Figure 102).

**Définition d'un shell :**

Un shell assure deux rôles principaux :

1. C'est un interpréteur de commandes ;
2. C'est un langage de programmation.

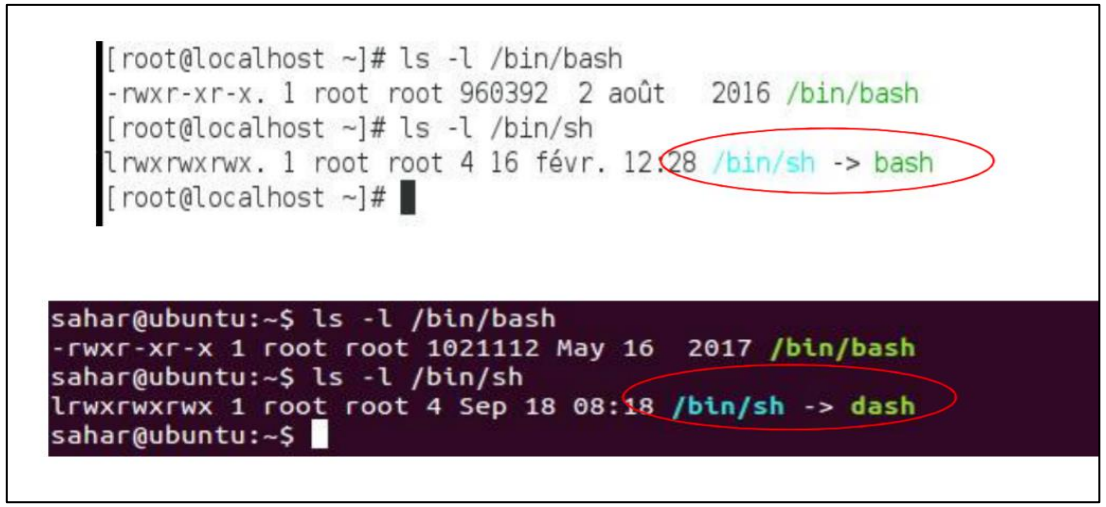



*Figure 102 : Terminal Linux Utilisation commande Shell (sh)*

PARTIE 3

**Personnaliser l'environnement BASH :**

Il y'a deux types de fichiers de configuration utilisés pour la personnalisation de l'environnement :

- Les fichiers qui sont lus au moment de la connexion (login).
- Les fichiers qui sont lus à chaque lancement d'un shell.

**Fichiers lus au moment de la connexion :**

- **/etc/profile** : commun à tous les utilisateurs (contient le umask).
- **~/.bash_profile** , **~/.bash_login** ou **~/.profile** spécifiques à chaque utilisateur.
- **~/. bash_history** : l'historique des commandes tapées.

**Fichiers lus à chaque lancement de shell :**

- **/etc/bashrc** ou **/etc/bash.bashrc** : commun à tous les utilisateurs.
- **~/.bashrc** spécifique à chaque utilisateur (on peut trouver et ajouter les alias ici).

**Variable d'environnement :**

- Une variable d'environnement est une variable accessible par tous les processus fils du shell courant.
- Pour créer une variable d'environnement, on exporte la valeur d'une variable avec la commande export export variable.
- Pour afficher toutes les variables d'environnement, il faut utiliser la commande env.
- Pour supprimer cette variable: unset ma_variable.

### Débuter avec les scripts shells

- Un script shell est un simple fichier texte exécutable (avec un droit d'exécution x).
- Il doit impérativement commencer par une ligne indiquant au système le shell qu'il faut utiliser :

**Shebang → #! /chemin/interpréteur**
**Exemple : /bin/bash ou /bin/sh**

- Le script doit être rendu exécutable : chmod +x fichier.
- Dans un script shell on peut avoir : des variables, des structures de contrôles, des structures répétitives… d'où l'appellation script shell.

**Exemple 1 :** Premier script shell

Ecrire un script qui affiche bonjour OPPT

**Exemple 1 :** Premier script shell

Ecrire un script qui affiche bonjour OPPT

**Solution :**

```
#!/bin/bash
Echo ”bonjour OPPT ”
```

**Les Entrées-Sorties:**

Ce sont les voies de communication entre le programme bash et la console.

- **echo**: affiche l'argument texte entre guillemets sur la sortie standard.

- **read**: permet l'affectation directe par lecture de la valeur, saisie sur l'entrée standard au clavier. Lorsque la commande read est utilisée sans argument, la ligne lue est enregistrée dans la variable prédéfinie du shell REPLY .

**Exemple 2 :**

Ecrire un script appelé script1 qui demande votre nom et prénom et affiche bonjour votre_nom votre_prenom.

**Solution Exemple 2 :**

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
  GNU nano 2.3.1                Fichier : fichier

#!/bin/bash
echo "Entrez votre nom : "
read nom
echo "Entrez votre prenom : "
read prenom
echo "Bonjour  $nom $prenom "
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[root@localhost ~]# nano fichier
[root@localhost ~]# bash fichier
Entrez votre nom :
tekup
Entrez votre prenom :
student
Bonjour  tekup student
[root@localhost ~]#
```

**Variables prédéfinies pour le passage de paramètres** :

Ces variables sont automatiquement affectées lors d'un appel de script suivi d'une liste de paramètres (voir le tableau ci-dessous) :

| Variable | Interprétation |
|---|---|
| $? | C'est la valeur de sortie de la dernière commande. Elle vaut 0 si la commande s'est déroulée sans problème. |
| $0 | Cette variable contient le nom du script |
| $1 à $9 | Les (éventuels) premiers arguments passés à l'appel du script |
| $# | Le nombre d'arguments passés au script |
| $* | La liste des arguments à partir de $1 |
| $$ | le n° PID du processus courant |
| $! | le n° PID du processus fils |

**Les opérateurs arithmétiques** sont décrits dans la table ci-dessous.

| Opérateur | Rôle |
|---|---|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| ** | exponentiation : $a^b$ =a**b |
| / | Division |
| % | Modulo |

Exemple :

```
[sahar@localhost ~]$ a=9
[sahar@localhost ~]$ b=10
[sahar@localhost ~]$ echo $((a+5))
14
[sahar@localhost ~]$ echo $((a-5))
4
[sahar@localhost ~]$ echo $((a**2))
81
```

```
[sahar@localhost ~]$ A=5
[sahar@localhost ~]$ B=6
[sahar@localhost ~]$ ((somme=A+B))
[sahar@localhost ~]$ echo $somme
11
[sahar@localhost ~]$ ((produit=A*B))
[sahar@localhost ~]$ echo $produit
30
```

**Les opérateurs Logiques** sont décrits dans la table ci-dessous.

| Opérateur | Rôle |
|---|---|
| **-eq** | est égal à<br>**if [ "$a" -eq "$b" ]** |
| **-ne** | n'est pas égal à<br>**if [ "$a" -ne "$b" ]** |
| **-gt** | est supérieur à<br>**if ["$a" -gt "$b" ]** |
| **-ge** | est supérieur ou égal à<br>**if [ "$a" -ge "$b" ]** |
| **-lt** | est inférieur à<br>**if [ "$a" -lt "$b" ]** |
| **-le** | est inférieur ou égal à<br>**if [ "$a" -le "$b" ]** |

❑ **La structure de contrôle : if**

**L'instruction conditionnelle : test**

Elle constitue l'indispensable complément de l'instruction if.

Elle permet :

- de reconnaître les caractéristiques des fichiers et des répertoires ;
- de comparer des chaînes de caractères ;
- de comparer algébriquement des nombres.

```
if  cmd1
then    Instructions1;
elif cmd2
then    Instructions2;
else    Instructions4;
fi
```

**La boucle for** est décrit dans la table ci-dessous.

| | Forme 1 | Forme 2 | Forme 3 |
|---|---|---|---|
| **Syntaxe** | **for** variable **in** ch1 ch2... chn **do** Commandes **done** | **for** variable **do** Commandes **done** | **for** variable **in** * **do** commandes **done** |
| **Interprétation** | les valeurs de variable sont les chaines de ch 1 à ch n | variable prend ses valeurs dans la liste des paramètres du script. | la liste des fichiers du répertoire constitue les valeurs prises par variable. |

**Itération while :** elle correspond à l'itération telle que décrit dans la table ci-dessous.

```
while suite_cmd1
do
  suite_cmd2
done
```