

Editions ENI

# **Debian GNU/Linux**

## Administration du système

dans la collection Ressources  
Informatiques

*Extrait*

## A. Utilisation de la mémoire

La quantité maximale de mémoire que Linux peut adresser sur une architecture Intel x86 est de 4 Go. Cette limite est repoussée à 64 Go pour les processeurs Pentium Pro et plus récents qui intègrent le mode PAE (*Physical Address Extension*).

Pour combler le manque de mémoire vive ou RAM, Linux supporte la mémoire virtuelle, c'est-à-dire qu'il utilise une unité de type bloc (généralement une partition sur un disque dur) pour simuler la mémoire faisant défaut. Cette mémoire virtuelle est alors nommée espace de pagination.



Le terme "swap" (échange) est aussi utilisé pour désigner la mémoire virtuelle mais il ne reflète pas exactement le mécanisme de segmentation en "page" de la mémoire sous Linux.

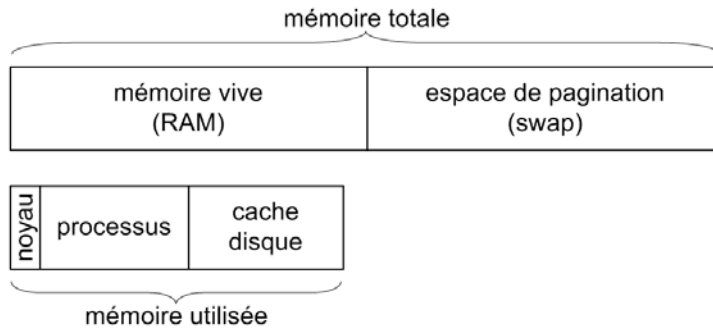
La mémoire disponible pour le système est alors la somme de la mémoire vive et de la mémoire virtuelle. Son utilisation est totalement transparente pour les processus utilisateur ; c'est le noyau qui gère le déplacement des pages mémoire en RAM ou en "swap" au besoin.

À l'inverse, pour optimiser les accès aux fichiers enregistrés sur des périphériques lents (disquette, CD-Rom, disque dur) en regard de la vitesse de la RAM, Linux utilise la mémoire vive comme cache disque (ou antémémoire). Ce mécanisme permet de garder en mémoire vive les données du disque un certain temps pour en augmenter la vitesse d'accès.

### 1. Système peu chargé

Sur un système peu chargé en comparaison avec la quantité de mémoire disponible, la mémoire vive est divisée en trois zones principales :

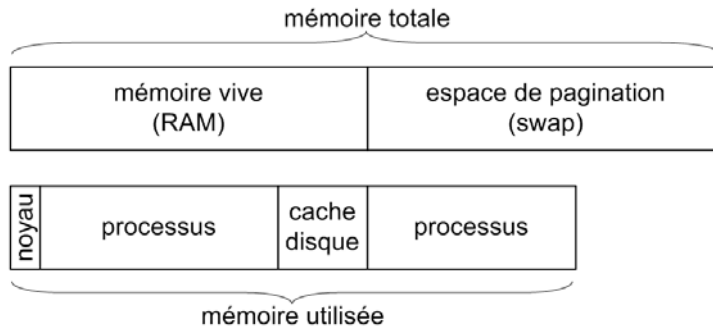
- Un espace fixe réservé pour le noyau. D'une taille approximative de 1 Mo, la réservation de cet espace pour le noyau et ses données constitue une manière simple mais très efficace de préserver l'espace mémoire du cœur du système d'exploitation.
- L'espace nécessaire à l'exécution des processus utilisateurs. Les processus emploient uniquement de la mémoire vive.
- Le cache disque utilise et remplit la mémoire vive restante au fur et à mesure des accès disque.



Lors d'une utilisation normale de Linux (en rapport avec la quantité de RAM présente sur la machine), l'espace de pagination n'est pas utilisé.

## 2. Système chargé

Sur un système chargé, Linux déplace en mémoire de pagination les processus les moins utilisés. Il garde cependant un minimum d'espace en mémoire vive pour l'antémémoire ; ce qui évite de trop détériorer les performances du système.



Il faut considérer un système qui "swappe" constamment comme un système installé sur une machine mal dimensionnée en terme de mémoire vive ; la solution la plus raisonnable consiste soit à augmenter la quantité de RAM sur la machine plutôt que la taille des espaces de pagination, soit à déplacer certains processus sur une autre machine.

## B. Pagination

Linux peut utiliser soit un fichier normal du système de fichiers, soit une partition séparée pour la pagination.

L'emploi d'une partition de swap est plus rapide car le noyau s'affranchit d'un système de fichiers pour enregistrer les pages mémoire des processus sur le disque ; cet espace de stockage est spécialement formaté pour la circonstance. Néanmoins, une partition sur un disque doit être dédiée à cet effet et il est difficile d'en modifier la taille par la suite.

Au contraire, un fichier d'échange se trouvant sur un système de fichiers, il est facile d'en modifier la taille.

Linux peut gérer jusqu'à trente-deux espaces de pagination allant de 40 ko à 2 Go pour les derniers noyaux sur une architecture matérielle Intel 386.

## 1. Partition de swap

Une partition de swap peut être enregistrée sur une partition de disque dur, un volume RAID ou un volume logique LVM. Le type des partitions Linux pour l'espace de pagination est 0x82 ("Échange Linux" sous **fdisk**).

Pour préparer la partition à accueillir l'espace de pagination, il faut la formater à l'aide de :

```
mkswap [-c] <partition ou volume RAID/LVM>
```

➤ Comme pour la commande **mke2fs**, l'option **-c** permet de vérifier les erreurs sur le disque en même temps que le formatage.

Une fois cette partition initialisée, il suffit de l'activer avec **swapon** ; cette commande permet en outre de consulter la charge des espaces de swap avec l'option **-s** :

```
[root]# swapon -s
Filename                Type          Size    Used    Priority
/dev/sdb2               partition    506036  0       -1
[root]# fdisk -l /dev/sdc

Disk /dev/sdc: 9100 MB, 9100032000 bytes
255 heads, 63 sectors/track, 1106 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1            1          122     979933+   83  Linux
/dev/sdc2           123          185     506047+   82  Linux swap / Solaris
[root]# mkswap /dev/sdc2
Setting up swapspace version 1, size = 518184 kB
no label, UUID=0b443f8c-e4d0-438e-a76c-c5d911cfff3b
[root]# swapon /dev/sdc2
[root]# swapon -s
Filename                Type          Size    Used    Priority
/dev/sdb2               partition    506036  0       -1
/dev/sdc2               partition    506036  0       -2
```

Il est possible de spécifier la priorité d'utilisation des différents espaces de swap. Les espaces de plus grande priorité sont employés en premier. Il est intéressant de fixer la même valeur pour des espaces se trouvant physiquement sur des disques différents car Linux les utilisera alors en parallèle. L'option **-p** de **swapon** permet de régler cette priorité.

D'un autre côté, on désactive un espace de pagination avec **swapoff**. Cette opération ne nécessite pas le redémarrage de la machine, mais il faut veiller à ne pas exécuter cette commande si la mémoire fait défaut.

Par exemple :

```
[root]# swapoff /dev/sdc2
[root]# swapon -s
Filename                                Type              Size      Used      Priority
/dev/sdb2                               partition        506036    0         -1
```

Pour que cette partition de swap soit utilisée de façon permanente et en particulier, qu'elle soit activée dès le démarrage de Linux, il faut ajouter la ligne suivante au fichier */etc/fstab* :

```
/dev/sdc2    swap          swap          defaults    0 0
```

## 2. Fichier de swap

L'utilisation d'un fichier à la place d'une partition est tout aussi simple à mettre en œuvre. La seule différence réside à la création du fichier ; on utilisera pour cela la commande **dd** qui permet entre autres de créer un fichier de la taille désirée :

```
dd if=<input> of=<output> bs=<blocks size> count=<nb>
```

La commande **dd** copie le fichier passé en entrée (**if=**) dans le fichier de sortie (**of=**). On peut limiter le nombre d'octets copiés en spécifiant le nombre (**count=**) et la taille (**bs=**) des blocs de données à copier.

- En utilisant */dev/zero* comme fichier d'entrée, il est possible de créer un fichier contenant autant d'octets "null" que l'on veut et donc d'une certaine taille. En effet, le fichier spécial */dev/zero* retourne des caractères "null" (premier caractère de la table ASCII) à l'infini quand on le lit.

Par exemple, pour créer un fichier de swap de 128 Mo :

```
[root]# dd if=/dev/zero of=/swapfile bs=1024k count=128
128+0 records in
128+0 records out
134217728 bytes transferred in 2.265933 seconds (59232875 bytes/sec)
[root]# ls -l /swapfile
-rw-r--r-- 1 root root 134217728 Apr  5 19:12 /swapfile
[root]# mkswap /swapfile
```

.../...

```

.../...
Setting up swapspace version 1, size = 134213 kB
no label, UUID=1087cf5f-123d-4957-9b3e-3402f5570bde
[root]# swapon -p2 /swapfile
[root]# swapon -s

```

Filename	Type	Size	Used	Priority
/dev/sdb2	partition	506036	0	-1
/swapfile	file	131064	0	2

Comme pour la partition de swap, pour activer de façon permanente ce fichier de pagination, il faut ajouter la ligne suivante au fichier `/etc/fstab` :

```

/swapfile      swap          swap          defaults,pri=2 0 0

```

## C. Gestion des processus

Un processus est un programme ou une commande en cours d'exécution sur un système. Linux peut exécuter plusieurs processus en même temps et il est possible de lancer plusieurs occurrences d'un même programme simultanément.

- Le terme "tâche" est équivalent à "processus". C'est pourquoi on qualifie Linux de système multitâche.

Tout processus est identifié par un numéro unique, le PID (*Process IDentifier*) ; le noyau utilise une table des processus pour la gestion des tâches.

Deux primitives du noyau Linux permettent de créer des processus : **fork** et **exec** (ce sont des primitives programmables qui peuvent être invoquées dans un langage comme C). La première permet à un processus de créer un clone de lui-même, la seconde servira à ce clone pour exécuter le code d'un autre programme à sa place. Il en résulte une affiliation entre processus ; on parle alors de processus fils et de processus pères.

Tout processus a donc obligatoirement un père, sauf le premier processus du système : `init`. Celui-ci est donc l'ancêtre de tous les processus du système et son PID est 1.

### 1. Démons

Un processus démon est un processus qui s'exécute en permanence sur le système ; son code boucle à l'infini afin d'attendre et d'être prêt lorsqu'un utilisateur sollicite le service auquel il est attaché. Certains de ces démons sont reconnaissables par leur nom se terminant par un `d`, comme `syslogd` (journalisation des messages) ou bien `inetd` (le démon fournisseur de plusieurs services).

### 2. Priorité des processus et ordonnancement

Linux exécute plusieurs processus sur la même machine bien que celle-ci ne possède pas un nombre de processeurs égal au nombre de processus. La méthode consiste alors à affecter une petite plage de temps (appelée quantum) successivement à chaque processus, pendant laquelle le processus peut utiliser le CPU. La partie du module chargée de l'affectation de ces quantum est appelée ordonnanceur.

Cet ordonnanceur doit donc gérer l'accès équitable de chaque processus au CPU. Une fois que le processus a utilisé son quantum de temps processeur, l'ordonnanceur le préempte et réévalue la priorité de tous les processus au moyen d'une file de priorité. Le processus avec la plus haute priorité dans l'état "prêt à s'exécuter et chargé en mémoire" est alors chargé dans le processeur. La priorité d'un processus est d'autant plus basse qu'il a récemment utilisé le processeur.

Il existe plusieurs algorithmes d'ordonnancement des processus qui ne seront pas décrits ici. Cependant, l'utilisateur a le moyen d'influencer l'ordre d'attribution du processeur en jouant sur la priorité des processus. Pour cela, il dispose de la commande **nice** pour changer le niveau de priorité par défaut et de **renice** pour changer celle d'un processus déjà lancé.

---

➤ "nice" signifie "gentil", on parle donc aussi de valeurs de gentillesse comme niveau de priorité.

---

#### a. nice

Le niveau de priorité par défaut d'un processus utilisateur est 0. Celui-ci peut varier de -20 (le plus prioritaire) à 19 (le moins prioritaire). Un moyen simple de retenir ce sens de priorité est d'utiliser le terme "gentillesse" ; ainsi, plus un processus sera "gentil" (19), plus il laissera le processeur aux autres. Inversement, moins il sera "gentil", plus il s'accapamera du temps processeur.

On utilise donc **nice** pour lancer un programme ou une commande en modifiant la valeur de gentillesse par défaut ; sans option, celle-ci sera incrémentée de 10, sinon l'ajustement spécifié avec l'option **-n** sera appliqué :

```
nice [-n <ajust.>] <commande>
```

Un utilisateur ordinaire ne peut qu'incrémenter la gentillesse des processus alors que l'administrateur peut spécifier des ajustements négatifs (pour des moins gentils nécessitant davantage de ressources CPU).

#### b. renice

Cette commande permet toujours de modifier les valeurs de gentillesse mais cette fois, sur des processus en cours d'exécution.

Editions ENI

# **APACHE (version 2)**

## Installation, administration et sécurisation

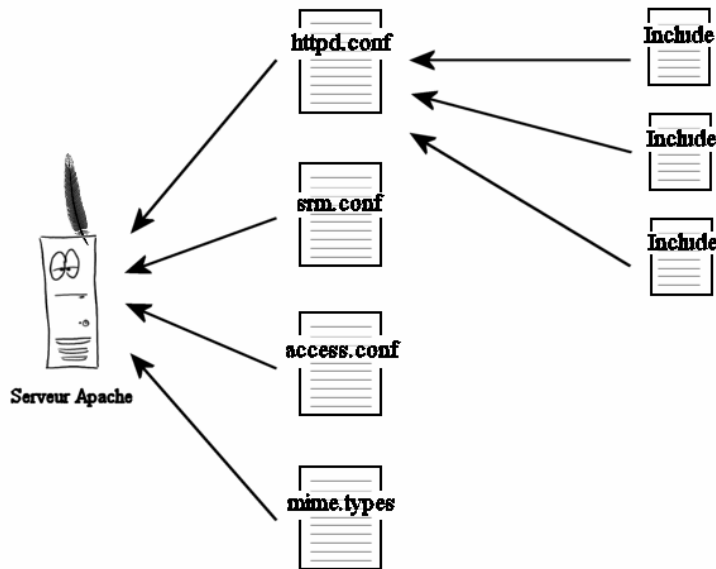
dans la collection Ressources  
Informatiques

*Extrait*

Voilà notre serveur web Apache installé et fonctionnel ! Voyons maintenant plus en détail sa configuration.

Nous allons tout d'abord voir quels sont ses fichiers de configuration, leur format et où ils se trouvent suivant le système d'exploitation. Ensuite, nous verrons plus en détail les directives de base les plus usitées faisant partie des modules **core** et **mpm** (pour la version 2 d'Apache). Enfin, nous découvrirons des outils graphiques permettant de générer cette configuration. Il s'agit de **Webmin**, **Comanche** ou **TkApache**. Une petite remarque tout de même : jamais aucun outil graphique ne remplacera l'efficacité d'un éditeur de texte tel que **Vim** permettant de modifier directement ces fichiers de configuration !

### A. Les fichiers de configuration



Apache est d'une extrême simplicité : sa configuration consiste à placer des directives dans des fichiers texte particuliers. Le principal se nomme **httpd.conf**. Pour des raisons historiques, nous pourrions trouver d'autres fichiers de configuration nommés **srm.conf** et **access.conf** qui ne sont plus utilisés, la configuration s'effectuant totalement dans le fichier **httpd.conf**. De plus, l'arborescence d'un site web pourra contenir un ou plusieurs fichiers **.htaccess** définissant une configuration propre aux répertoires où ils se trouvent.

Voyons un peu la syntaxe d'un fichier de configuration. Il contient une seule directive par ligne. Si cette dernière doit s'étendre sur plusieurs lignes, le caractère `"\"` est utilisé pour indiquer que la ligne de directive n'est pas terminée et se continue sur la ligne suivante.

Les directives ne sont pas sensibles à la casse, mais leur arguments le sont parfois !!!

Les lignes commençant par un caractère "#" sont des lignes de commentaires et sont donc ignorées. De même, les tabulations et espaces en début de ligne sont ignorés. Donc n'hésitez surtout pas à commenter et à indenter vos fichiers ! Cela permettra une meilleure compréhension de ces derniers et donc moins d'erreurs et plus de sécurité.

Voyons plus en détail tous ces fichiers de configuration.

## 1. Le fichier de configuration principal : httpd.conf

**httpd.conf** se trouve dans le répertoire de configuration défini lors de la compilation du binaire. Il s'agit du répertoire **/usr/local/apache[2]?/etc** par défaut. Sous Debian, il se trouve dans **/etc/apache**, tandis que sous Windows, nous le trouverons dans le répertoire **conf/** du répertoire d'installation d'Apache. Par défaut, il s'agit de **C:/Program Files/Apache Group/Apache[2]?/conf**.

➤ Remarquez que sous Windows, les chemins n'utilisent pas le caractère "\" comme c'est normalement l'habitude sous ce système. Le caractère "/" est par contre utilisé !

Mais nous pouvons indiquer un autre emplacement lors de l'exécution du binaire à l'aide de l'option `-f`.

D'autres fichiers de configuration peuvent être inclus à partir de **httpd.conf** à l'aide de la directive `Include` :

Syntaxe : `Include chemin`

Cette directive permet d'inclure d'autres fichiers de configuration dans celui où elle se trouve.

*chemin* définit l'emplacement du fichier à inclure. Cet emplacement peut être absolu, c'est-à-dire commençant par un / sous Linux ou par une lettre d'un lecteur (par exemple C:/) sous Windows, ou relatif au répertoire désigné comme étant le **ServerRoot**, c'est-à-dire le répertoire **/usr/local/apache[2]?** par défaut, **/etc/apache** sous Debian, ou **C:/Program Files/Apache Group/Apache[2]?/** sous Windows.

Depuis la version 1.3.13, Apache permet de définir un répertoire comme *chemin*. Tout fichier contenu dans ce répertoire, ainsi que tous ses sous-répertoires, seront considérés comme fichiers de configuration à inclure. Cela permet une organisation de la configuration d'Apache comme par exemple un fichier incluant les différents modules, un autre avec les options générales, un fichier de configuration par hôte virtuel (**VirtualHost**), etc. Enfin, depuis la version 2.0.41, *chemin* peut contenir des caractères de remplacement tels que le caractère \* permettant ainsi de définir plusieurs fichiers à l'aide d'une seule directive `Include`.

➤ Le serveur Apache, par défaut, ne peut abriter qu'un seul site web (ou hôte). À l'aide de directives `<VirtualHost>`, Apache peut abriter plusieurs sites web (ou hôtes virtuels).

Quelques exemples placés dans le fichier **httpd.conf** :

```
# mon fichier de configuration incluant les modules
Include /etc/apache/modules.conf
# mon répertoire contenant un fichier de configuration
# par virtualhost. J'utilise *.conf pour être sûr de
# ne pas inclure d'autres fichiers qui empêcheraient
# Apache de démarrer correctement
Include /etc/apache/virtualHosts/*.conf
```

La commande `apachectl configtest` permet de savoir quels fichiers de configuration sont lus et pris en compte :

```
# apachectl configtest
Processing config file: /etc/apache/modules.conf
Processing config file: /etc/apache/virtualHosts/domaine1.conf
Processing config file: /etc/apache/virtualHosts/domaine2.conf
Processing config file: /etc/apache/virtualHosts/mon_asso.conf
Syntax OK
```

## 2. Les fichiers **.htaccess**

Apache permet une gestion décentralisée de la configuration par l'utilisation de fichiers spéciaux à l'intérieur de l'arborescence web. Ces fichiers spéciaux sont généralement appelés **.htaccess**, mais n'importe quel nom peut être défini par la directive `AccessFileName`. Les directives placées dans les fichiers **.htaccess** s'appliquent au répertoire où se trouve le fichier ainsi qu'aux sous-répertoires. Les fichiers **.htaccess** respectent la même syntaxe que les fichiers principaux de configuration. Comme les fichiers **.htaccess** sont lus à chaque requête, les modifications effectuées dans ces fichiers prennent effet immédiatement.

---

➤ Toutes les directives ne peuvent pas être placées dans les fichiers **.htaccess**. Il faut tout d'abord vérifier leur contexte.

---

L'administrateur peut contrôler quelles directives peuvent être placées dans les fichiers **.htaccess** en modifiant la directive `AllowOverride` du fichier principal de configuration.

Les fichiers **.htaccess** sont très souvent utilisés pour contrôler l'accès à un répertoire dans l'arborescence web du serveur. Par exemple, l'accès à une ressource web pourra être permise à l'aide d'une procédure d'authentification mise en place à l'aide de ce fichier. Pour plus d'informations à ce sujet, voyez le chapitre 6 traitant de la sécurité.

### a. `AccessFileName`

Syntaxe : `AccessFileName nomFichier [nomFichier] ...`

Cette directive définit le ou les noms que porteront les fichiers de configuration dans un contexte de répertoire. Par défaut, il s'agit du nom **.htaccess**.

Lorsqu'il retourne un document au client, le serveur cherche le premier fichier de contrôle d'accès existant dans cette liste dans chacun des répertoires du chemin d'accès menant à ce document. Ceci pour déterminer si l'accès est autorisé dans chacun de ces répertoires. Par exemple, si nous avons la ligne suivante dans notre configuration principale :

```
AccessFileName .acl
```

Avant d'envoyer le document `/usr/local/web/index.html`, le serveur lira les fichiers `/.acl`, `/usr/.acl`, `/usr/local/.acl` et `/usr/local/web/.acl` à la recherche de directives, sauf si celles-ci ont été désactivées par la présence des lignes suivantes :

```
<Directory />
  AllowOverride None
</Directory>
```

### b. AllowOverride

Syntaxe :

```
AllowOverride All/None/type de directive [type de directive] ...
```

Lorsque le serveur trouve un fichier `.htaccess` (comme spécifié par `AccessFileName`) il doit savoir quelles directives déclarées dans ce fichier peuvent outrepasser les droits fixés par des directives précédentes.

Si la directive est définie à `None`, les fichiers `.htaccess` sont ignorés. Dans ce cas, le serveur n'essaie même pas de lire les fichiers `.htaccess`.

Si la directive est définie à `All` toutes les directives possibles dans un contexte de répertoire sont autorisées dans les fichiers `.htaccess`.

Il existe plusieurs groupes de directives :

#### AuthConfig

Autorise l'usage de la directive `Authorization` (`AuthDBMGroupFile`, `AuthDBMUserFile`, `AuthGroupFile`, `AuthName`, `AuthType`, `AuthUserFile`, `Require`, etc.).

#### FileInfo

Autorise l'usage de directives contrôlant l'accès aux types de document (`AddEncoding`, `AddLanguage`, `AddType`, `DefaultType`, `ErrorDocument`, `LanguagePriority`, etc.).

#### Indexes

Autorise l'usage de directives contrôlant l'indexation des répertoires (`AddDescription`, `AddIcon`, `AddIconByEncoding`, `AddIconByType`, `DefaultIcon`, `DirectoryIndex`, `FancyIndexing`, `HeaderName`, `IndexIgnore`, `IndexOptions`, `ReadmeName`, etc.).

#### Limit

Autorise l'usage de directives contrôlant les accès de certains hôtes (`allow`, `_deny` et `order`).

### Options

Autorise l'usage de directives contrôlant certaines fonctionnalités spécifiques des répertoires (`Options` et `XBitHack`).

### 3. Le fichier MIME

Un dernier fichier de configuration est lu par Apache. Il s'agit du fichier contenant les types MIME.

MIME (*Multipurpose Internet Mail Extensions*) est une extension du protocole de messagerie SMTP. Il permet l'échange de différents types de données comme du son, des images, de la vidéo et bien d'autres encore, en plus du texte inclus dans le protocole original (SMTP : *Simple Mail Transfert Protocol*). C'est en 1991 que Nathan Borenstein de Bellcore propose à l'IETF (*Internet Engineering Task Force*, organisation contrôlant et définissant les protocoles standards de l'Internet) d'étendre le protocole SMTP pour que les serveurs et clients présents sur l'Internet puissent reconnaître et utiliser toutes sortes de données autres que du simple texte.

Les serveurs web insèrent des en-têtes MIME permettant aux clients web, c'est-à-dire les navigateurs comme Internet Explorer, Netscape, etc, de lancer une application pour lire un type particulier de données. La plupart de ces applications font partie du navigateur lui-même ou sont ajoutées au navigateur en tant que plug-ins.

Les nouveaux types MIME sont enregistrés auprès de l'IANA (*Internet Assigned Numbers Authority*).

Nous trouverons les types MIME dans le fichier `/etc/mime.types` sous Debian, ou dans le répertoire de configuration d'Apache. Ce fichier est défini à l'aide de la directive `TypesConfig`, faisant partie du module `mod_mime`, dans `httpd.conf`. Par défaut, il s'agit du fichier `mime.types` fourni avec les sources d'Apache. Il peut être aussi récupéré à l'adresse suivante, chargé de maintenir cette liste :

<http://www.isi.edu/in-notes/iana/assignments/media-types/media-types>

Ce fichier ne doit pas être édité ! Si l'ajout ou le remplacement d'un type MIME est nécessaire, il faut alors utiliser la directive `AddType`, provenant elle aussi du module `mod_mime`, dans le fichier de configuration `httpd.conf`.

#### a. TypesConfig

Syntaxe : `TypesConfig chemin`

Cette directive définit le fichier contenant les types MIME.

`chemin` indique l'emplacement de ce fichier. Par défaut, sa valeur est `conf/mime.types`.

Ce fichier est un simple fichier texte avec un type MIME et ses extensions par ligne. Ces lignes sont de la forme suivante :

```
TypeMIME extension extension ...
```

## b. AddType

Syntaxe :

```
AddType typeMIME extension [extension] ...
```

Cette directive ajoute la définition d'un nouveau type MIME et les extensions qui lui sont associées.

*typeMIME* est le code MIME du type de média à prendre en compte pour les documents se terminant par une des extensions listées, une fois les extensions sémantiquement associées aux métainformations d'encodage et de langues supprimées (voir les directives `AddEncoding` et `AddLanguage`).

*extension* est la liste des extensions associées au type MIME, insensible à la casse. De plus, le point `.` au début de l'extension est facultatif.

Voici un exemple définissant les fichiers contenant des images GIF :

```
AddType image/gif .gif
```



Il est recommandé d'ajouter des nouveaux types MIME via la directive `AddType` plutôt qu'en modifiant le fichier **TypesConfig**.

## B. Les directives de type bloc du module core

Les directives, d'une manière générale, sont globales au serveur. Certaines d'entre elles permettent d'appliquer une configuration particulière seulement à une partie du serveur. Il s'agit des directives de type bloc. Il en existe plusieurs, s'appliquant à un répertoire, un fichier... Elles définissent un bloc commençant par `<nom_de_la_directive_paramètres>` et finissant par `</nom_de_la_directive>`. Suivant les directives, il peut y avoir des paramètres indiquant par exemple un répertoire, un fichier, une url...

### 1. Le système de fichiers et l'espace web

Les directives de type bloc s'appliquent à un espace particulier. Celui-ci peut être une partie du système de fichiers, ou un espace de type web.

Le système de fichiers est l'arborescence des répertoires et fichiers telle qu'elle est vue par le système d'exploitation. Des chemins tels que `/etc/apache` ou `C:\Program Files\Apache Group\Apache\conf` désignent bien les répertoires `/etc/apache` ou `C:\Program Files\Apache Group\Apache\conf` présents sur le disque dur.

L'espace web désigne l'arborescence telle qu'elle est vue par les clients web comme les navigateurs Internet. Prenons un exemple : soit un serveur web Apache dont le site web est placé dans le répertoire `/var/www` sous Linux, ou `C:\Program Files\Apache Group\Apache\htdocs` sous Windows. Alors, l'espace web `/dir1/` désigne en fait le répertoire `/var/www/dir1/` sous Linux, ou `C:\Program Files\Apache Group\Apache\htdocs\dir1\` sous Windows, sur le système de fichiers. Ce n'est qu'un exemple !

Editions ENI

# **MySQL 5**

Installation, mise en oeuvre,  
administration et  
programmation

dans la collection Ressources  
Informatiques

*Extrait*

# A. La gestion des privilèges

## 1. Introduction

La sécurité de MySQL est basée sur un système de privilèges simple et performant. Chaque utilisateur qui veut se connecter doit s'identifier auprès du serveur MySQL. Cette identification est une combinaison de trois critères : l'hôte (la machine) depuis lequel la connexion est initiée, un nom d'utilisateur et un mot de passe.

À cette identification est associé un ensemble d'attributs qui vont permettre de définir des privilèges sur certaines bases de données, tables, colonnes ou opérations particulières.

---

➤ Attention, il n'y a aucun lien entre les utilisateurs MySQL et les utilisateurs Linux ou Windows. Les noms d'utilisateurs et les mots de passe sont différents (sauf si vous attribuez la même valeur) et la modification du mot de passe d'un compte MySQL n'entraîne aucune modification du compte Linux ou Windows et inversement.

---

## 2. Privilèges par défaut et connexion en tant qu'administrateur

Après avoir installé MySQL, nous disposons de deux utilisateurs pouvant se connecter en local (depuis le serveur lui-même).

L'un se nomme **root**, c'est le super-utilisateur, il dispose de tous les privilèges sur le serveur et sur toutes les bases de données. Cet utilisateur n'a pas de mot de passe.

L'autre est l'utilisateur anonyme, c'est-à-dire tout utilisateur non connu du système, il a accès à la base de données **test** et à toutes les bases de données commençant par **test**.

- L'une des premières opérations à effectuer est d'affecter un mot de passe à l'utilisateur **root**. Nous verrons cette notion dans la section A - 4 - c - Modification d'un mot de passe d'un utilisateur.
- 

Utilisons le client MySQL pour nous connecter au serveur. Ceci requiert un certain nombre de paramètres : l'hôte (le nom du serveur), le nom de l'utilisateur et le mot de passe. Tous ces arguments sont optionnels :

```
shell>mysql -h hostname -u username -pmypassword
```

ou

```
shell>mysql -host=hostname -user=username -password=mypassword
```

Évitons de taper le mot de passe dans la ligne de commande. Laissons plutôt le mot de passe vide.

Nous préférons donc plutôt la syntaxe suivante :

```
shell>mysql -h hostname -u username -p
Enter password:*****
```

Lorsque nous validons la commande, MySQL demande le mot de passe.

---

- Si nous tapons directement le mot de passe dans la ligne de commande, la simple commande `ps auxww` (qui liste les processus en cours sous Linux) permet de révéler le mot de passe.
- 

Si nous ne spécifions pas les paramètres optionnels, MySQL considère que :  
--host = -h = localhost (la machine depuis laquelle la commande est exécutée)

--user = -u = nom de login de l'utilisateur Linux ou Windows

--password = -p = pas de mot de passe

→) Donc, si nous sommes connecté en tant qu'utilisateur **root** sans mot de passe sur un système Linux, les commandes suivantes sont équivalentes (attention, car un utilisateur devrait toujours avoir un mot de passe et particulièrement l'utilisateur **root** qui est l'administrateur du système) :

```
mysql -h localhost -u root -p
mysql -u root -p
mysql -u root
mysql
```

```
cthibaud@srv-jupiter:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11 to server version: 5.0.15-Debian_1-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> _
```

Le client MySQL, appelé aussi moniteur, permet de converser avec le serveur en lui envoyant des requêtes (aussi appelées commandes ou instructions). Chacune de ces requêtes doit se terminer par le symbole point-virgule (;) pour être interprétée, sauf pour quelques commandes telles que `use` et `exit` pour lesquelles le symbole de terminaison est optionnel. Il est possible d'enchaîner un bloc de requêtes en ne spécifiant qu'un point virgule à la fin du bloc. Les requêtes permettent de travailler sur les données (sélection, insertion, suppression...) et sur les privilèges (création d'utilisateurs, gestion des privilèges...). Elles permettent aussi d'effectuer des opérations de maintenance du système telles que les sauvegardes.

Après la connexion, le prompt (symbole de début de ligne : `mysql>`) change et attend nos requêtes.

→) La première requête que nous tapons permet de se déconnecter du serveur. Une fois exécutée, le prompt redevient celui du système.

```
mysql>exit
```

```
cthibaud@srv-jupiter:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11 to server version: 5.0.15-Debian_1-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> exit
Bye
cthibaud@srv-jupiter:~$ _
```

### 3. Principe de fonctionnement du système de privilèges

MySQL conserve les privilèges dans la base de données nommée **mysql** présente dès l'installation du serveur. MySQL permet d'octroyer des privilèges aux utilisateurs mais ne permet pas d'en refuser. Il utilise les tables **user**, **db**, **host**, **tables\_priv** et **columns\_priv** pour gérer les privilèges des utilisateurs. Vérifions la présence de cette base de données et de ces tables. Par défaut, aucune base de données n'est sélectionnée, la première opération à effectuer est donc d'en sélectionner une.

→) Nous utilisons la requête suivante pour sélectionner la base de données **mysql** :

```
mysql>use mysql;
```

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```

→) et la requête suivante pour lister les tables :

```
mysql>show tables;
```

```
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv    |
| db              |
| func            |
| help_category  |
| help_keyword    |
| help_relation   |
| help_topic      |
| host            |
| proc            |
| procs_priv      |
| tables_priv     |
| time_zone       |
| time_zone_leap_second |
| time_zone_name  |
| time_zone_transition |
| time_zone_transition_type |
| user            |
+-----+
17 rows in set (0.01 sec)

mysql> _
```

MySQL contrôle l'accès en deux temps :

- Il vérifie l'identité de l'utilisateur à sa connexion (via l'hôte de connexion, le nom de l'utilisateur et le mot de passe).
- Il vérifie chacune des requêtes envoyées pour s'assurer que l'utilisateur a les privilèges nécessaires.

## 4. Étape 1 : la connexion

### a. Fonctionnement

MySQL utilise la table **user** pour vérifier qu'un utilisateur est autorisé à se connecter.

→ Visualisons la structure de cette table avec la requête suivante :

```
mysql> describe user;
```

```
mysql> describe user;
```

Field	Type	Null	Key	Default	Extra
Host	char(60)	NO	PRI		
User	char(16)	NO	PRI		
Password	varchar(41)	NO			
Select_priv	enum('N','Y')	NO		N	
Insert_priv	enum('N','Y')	NO		N	
Update_priv	enum('N','Y')	NO		N	
Delete_priv	enum('N','Y')	NO		N	
Create_priv	enum('N','Y')	NO		N	
Drop_priv	enum('N','Y')	NO		N	
Reload_priv	enum('N','Y')	NO		N	
Shutdown_priv	enum('N','Y')	NO		N	
Process_priv	enum('N','Y')	NO		N	
File_priv	enum('N','Y')	NO		N	
Grant_priv	enum('N','Y')	NO		N	
References_priv	enum('N','Y')	NO		N	
Index_priv	enum('N','Y')	NO		N	
Alter_priv	enum('N','Y')	NO		N	
Show_db_priv	enum('N','Y')	NO		N	
Super_priv	enum('N','Y')	NO		N	
Create_tmp_table_priv	enum('N','Y')	NO		N	
Lock_tables_priv	enum('N','Y')	NO		N	
Execute_priv	enum('N','Y')	NO		N	
Repl_slave_priv	enum('N','Y')	NO		N	
Repl_client_priv	enum('N','Y')	NO		N	
Create_view_priv	enum('N','Y')	NO		N	
Show_view_priv	enum('N','Y')	NO		N	
Create_routine_priv	enum('N','Y')	NO		N	
Alter_routine_priv	enum('N','Y')	NO		N	
Create_user_priv	enum('N','Y')	NO		N	
ssl_type	enum('', 'ANY', 'x509', 'SPECIFIED')	NO			
ssl_cipher	blob	NO			
x509_issuer	blob	NO			
x509_subject	blob	NO			
max_questions	int(11) unsigned	NO		0	
max_updates	int(11) unsigned	NO		0	
max_connections	int(11) unsigned	NO		0	
max_user_connections	int(11) unsigned	NO		0	

```
37 rows in set (0.02 sec)

mysql>
mysql>
mysql>
mysql>
```

Nous remarquons que le nom (**User**) est limité à 16 caractères, le mot de passe (**Password**) est limité à 41 caractères et le nom d'hôte (**Host**) à 60 caractères.

MySQL utilise les champs **Host**, **User** et **Password** pour valider la connexion. Il trie les utilisateurs du plus restrictif au moins restrictif et utilise la première ligne répondant aux critères de la connexion pour autoriser l'accès au serveur. Si aucune ligne n'est trouvée, la connexion est refusée.

# **PHP 5.2**

Développer un site Web  
dynamique et interactif

dans la collection Ressources  
Informatiques

*Extrait*

## A. Vue d'ensemble

### 1. Petit rappel sur les formulaires

Le formulaire est un outil de base indispensable pour les sites Web dynamiques puisqu'il permet à l'utilisateur de saisir des informations et donc d'interagir avec le site.

Un formulaire HTML est défini entre les balises <FORM> et </FORM>.

#### Syntaxe

```
<FORM
  [ ACTION="url_de_traitement" ]
  [ METHOD="GET"|"POST" ]
  [ NAME="nom_formulaire" ]
>
...
</FORM>
```

Les options de la balise <FORM> sont les suivantes :

ACTION	URL relative ou absolue ( <i>Uniform Resource Locator</i> ), qui va traiter le formulaire, en ce qui nous concerne, un script PHP. Si rien n'est spécifié, la même URL que la page est utilisée.
METHOD	Mode de transmission vers le serveur des informations saisies dans le formulaire. GET (valeur par défaut) : les données du formulaire sont transmises dans l'URL. POST : les données du formulaire sont transmises dans le corps de la requête.
NAME	Nom du formulaire. Si la page HTML contient plusieurs formulaires, le nom permet de les différencier. En ce qui nous concerne, ce nom ne présente pas d'intérêt car il n'est pas récupéré dans le script de traitement du formulaire. Par contre, il peut être utilisé côté client, en JavaScript par exemple.

Entre les balises <FORM> et </FORM>, il est possible de placer des balises <INPUT>, <SELECT> ou <TEXTAREA> pour définir des zones de saisies.

#### Exemple (formulaire HTML complet)

```
<FORM>
Nom :
<INPUT TYPE="text" NAME="nom" VALUE=""
  SIZE="20" MAXLENGTH="20">
Mot de passe :
<INPUT TYPE="password" NAME="mot_de_passe" VALUE=""
  SIZE="20" MAXLENGTH="20">
<BR>Sexe :
<INPUT TYPE="radio" NAME="sexe" VALUE="M"> Masculin
<INPUT TYPE="radio" NAME="sexe" VALUE="F"> Féminin
<INPUT TYPE="radio" NAME="sexe" VALUE="?"
  CHECKED> Ne sait pas
<BR>Photo :
<INPUT TYPE="file" NAME="photo" VALUE="" SIZE="50">
<BR>Couleurs préférées :
<INPUT TYPE="checkbox" NAME="bleu"> Bleu
<INPUT TYPE="checkbox" NAME="blanc"> Blanc
<INPUT TYPE="checkbox" NAME="rouge"> Rouge
```

```

<INPUT TYPE="checkbox" NAME="nesaitpas" CHECKED> Ne sait pas
<BR>Langue :
<SELECT NAME="langue">
  <OPTION VALUE="E">Espagnol
  <OPTION SELECTED VALUE="F">Français
  <OPTION VALUE="I">Italien
</SELECT>
<BR>Fruits préférés :<BR>
<SELECT NAME="fruits" MULTIPLE SIZE="8">
  <OPTION VALUE="A">Abricots
  <OPTION VALUE="C">Cerises
  <OPTION VALUE="F">Fraises
  <OPTION VALUE="P">Pêches
  <OPTION SELECTED VALUE="?">Ne sait pas
</SELECT>
<BR>Commentaire :<BR>
<TEXTAREA NAME="commentaire" ROWS="4" COLS="50"></TEXTAREA>
<BR>
<INPUT TYPE="hidden" NAME="invisible" VALUE="123"><BR>
<INPUT TYPE="submit" NAME="OK" VALUE="OK">
<INPUT TYPE="image" NAME="valider"
  SRC="valider.gif" WIDTH="23" HEIGHT="34">
<INPUT TYPE="reset" NAME="effacer" VALUE="Effacer">
<INPUT TYPE="button" NAME="action" VALUE="Ne fait rien">
</FORM>

```

Nom :  Mot de passe :   
 Sexe :  Masculin  Féminin  Ne sait pas  
 Photo :    
 Couleurs préférées :  Bleu  Blanc  Rouge  Ne sait pas  
 Langue :    
 Fruits préférés :  
  
  
  
  
  
  
 Commentaire :

Résultat

### 2. Interaction entre un formulaire et un script PHP

PHP peut intervenir à deux endroits par rapport au formulaire :

- pour la construction du formulaire, si ce dernier doit contenir des informations dynamiques ;
- pour le traitement du formulaire (c'est-à-dire des données saisies par l'utilisateur dans le formulaire).

Trois grandes méthodes sont utilisables pour faire interagir un formulaire et un script PHP :

- placer le formulaire dans un document HTML "pur" (.htm ou .html), le formulaire ne contient alors aucun élément dynamique, et indiquer le nom du script PHP qui doit traiter le formulaire dans l'option ACTION de la balise <FORM> ;
- placer le formulaire dans un script PHP (par exemple, pour construire une partie du formulaire dynamiquement) et faire traiter le formulaire par un autre script PHP (mentionné dans l'option ACTION de la balise <FORM> ;
- placer le formulaire dans un script PHP (par exemple, pour construire une partie du formulaire dynamiquement), et le faire traiter par le même script PHP (mentionné dans l'option ACTION de la balise <FORM> ou appelé par défaut si cette option n'est pas présente).

#### Exemple (première méthode)

##### Document HTML saisie.htm

```
<HTML>
<HEAD><TITLE>Saisie</TITLE></HEAD>
<BODY>
<FORM ACTION="traitement.php" METHOD="POST">
Nom : <INPUT TYPE="text" NAME="nom" VALUE=""><BR>
<INPUT TYPE="submit" NAME="OK" VALUE="OK">
</FORM>
</BODY>
</HTML>
```

##### Script PHP traitement.php

```
<?php
/* A faire ...
- récupérer les informations saisies
- faire le traitement
- afficher une nouvelle page
*/
?>
```

Par ailleurs, sur une autre page, un lien ([Saisie](#) par exemple) peut être inséré pour appeler le formulaire de saisie :

```
<A HREF="saisie.htm">Saisie</A>
```

Résultat

- Affichage initial du formulaire :

- Saisie d'une information :

- Le résultat du clic sur le bouton **OK** est une page vide car, pour l'instant, le script de traitement ne fait rien.

Exemple (deuxième méthode)Document PHP saisie.php

Un peu de code PHP (en gras) est utilisé pour générer la partie dynamique du formulaire.

```
<?php
// inclure un fichier qui contient des définitions de
// constantes, dont le titre de la page (TITRE_PAGE_SAISIE)
require("constantes.inc");
// initialisation d'une variable qui contient la valeur
// initiale de la zone de saisie (dans la pratique, cette
// valeur vient sans doute d'ailleurs et n'est pas codée
// en dur)
$nom = "X";
// dans le code HTML qui suit, inclusion de deux petits
// bouts de code PHP pour afficher respectivement le titre
// de la page et la valeur initiale de la zone de saisie
?>
<HTML>
<HEAD><TITLE><?php echo TITRE_PAGE_SAISIE ?></TITLE></HEAD>
<BODY>
<FORM ACTION="traitement.php" METHOD="POST">
Nom : <INPUT TYPE="text" NAME="nom"
      VALUE="<?php echo $nom ?>"><BR>
<INPUT TYPE="submit" NAME="OK" VALUE="OK">
</FORM>
</BODY>
</HTML>
```

Script PHP traitement.php


```
<?php
/* A faire ...
- récupérer les informations saisies
- faire le traitement
- afficher une nouvelle page
*/
?>
```

Par ailleurs, sur une autre page, un lien ([Saisie](#) par exemple) peut être inséré pour appeler le formulaire de saisie :

```
<A HREF="saisie.php">Saisie</A>
```

### Résultat

- Affichage initial du formulaire (une valeur initiale dynamique est proposée pour la zone de saisie) :



- Saisie d'une information :



- Le résultat du clic sur le bouton **OK** est une page vide car, pour l'instant, le script de traitement ne fait rien.

### Exemple (troisième méthode)

#### Document PHP saisie.php

C'est le même script que précédemment, dans lequel nous avons simplement changé l'option ACTION de la balise <FORM> pour indiquer que le formulaire doit être traité par le même script saisie.php (ne pas mettre d'option ACTION aurait le même effet).

```
<?php
// inclure un fichier qui contient des définitions de
// constantes, dont le titre de la page (TITRE_PAGE_SAISIE)
require("constantes.inc");
// initialisation d'une variable qui contient la valeur
// initiale de la zone de saisie (dans la pratique, cette
// valeur vient sans doute d'ailleurs et n'est pas codée en dur)
$nom = "X";
// dans le code HTML qui suit, inclusion de deux petits
// bouts de code PHP pour afficher respectivement le titre
// de la page et la valeur initiale de la zone de saisie
?>
<HTML>
<HEAD><TITLE><?php echo TITRE_PAGE_SAISIE ?></TITLE></HEAD>
<BODY>
<FORM ACTION="saisie.php" METHOD="POST">
Nom : <INPUT TYPE="text" NAME="nom"
      VALUE="<?php echo $nom ?>"><BR>
<INPUT TYPE="submit" NAME="OK" VALUE="OK">
</FORM>
</BODY>
</HTML>
```

Par ailleurs, sur une autre page, un lien ([Saisie](#) par exemple) peut être inséré pour appeler le formulaire de saisie :

```
<A HREF="saisie.php">Saisie</A>
```

### Résultat

- Affichage initial du formulaire (une valeur initiale dynamique est proposée pour la zone de saisie) :

 A screenshot of a web form. It consists of a text input field containing the character 'X' and a button labeled 'OK' positioned below the input field.

- Saisie d'une information :

 A screenshot of a web form. The text input field now contains the name 'Olivier'. The 'OK' button remains below the input field.

- Le résultat du clic sur le bouton **OK** est la même page de nouveau affichée, car, pour l'instant, le script de traitement ne fait rien de plus.

 A screenshot of a web form, identical to the first one, showing the text input field with 'X' and the 'OK' button.

### Que s'est-il passé ?

Le script `saisie.php` a été appelé de nouveau, par le clic sur le bouton **OK** ; il s'est donc exécuté comme lors de son premier appel (lien [Saisie](#)). Comme le code ne fait rien, pour l'instant, afin de traiter le formulaire, il s'affiche comme initialement. Utiliser le même script pour l'affichage initial et le traitement nécessite d'être en mesure de faire la différence entre les deux types d'appels (cf. B - 2. Les différents types de zones).

### Que choisir ?

Le choix de telle ou telle méthode dépend de la complexité du site et des préférences de chacun.

Quelques considérations générales :

- Séparer la page HTML (ou le script PHP qui génère le formulaire) du script PHP présente un inconvénient au niveau de la maintenance : si des modifications sont apportées au formulaire, il y a deux fichiers à modifier (avec des risques d'erreur, d'oubli...).
- Inversement, si le formulaire n'a aucune partie dynamique, l'écrire dans un fichier HTML séparé du script PHP qui le traite, permet de bien séparer l'interface utilisateur (la couche "présentation") du traitement.
- Dans la pratique, pour faciliter la maintenance, il est souhaitable de définir certaines valeurs présentées à plusieurs reprises (nom de société par exemple) dans des constantes ou des variables et d'utiliser ces constantes/variables dans les pages : toutes les pages deviennent un peu dynamiques et la troisième méthode paraît optimale.

Dans la suite de ce chapitre, nous allons rentrer dans le détail du traitement du formulaire en PHP, en utilisant des exemples construits sur le modèle de la troisième méthode.