Cours Apache :
INSTALLATION ET ADMINISTRATION

ATRID

Cours Apache : INSTALLATION ET ADMINISTRATION
par ATRID

Historique des version
Version 1.0 du 3/09/1998
Version initiale
Version 1.1 du 28/01/2000
Ajout API Modules

Version 1.2 du 07/11/2000
Conversion en SGML DocBook

Table des matières

Chapitre 1. Présentation

Le serveur HTTP Apache est le fruit du travail d'un groupe de volontaires, The Apache Group, qui a voulu réaliser un serveur Web du même niveau que les produits commerciaux sous forme de logiciel libre c'est à dire que son code source est disponible. L'équipe d'origine a été rejointe par des centaines d'utilisateurs qui, par leurs idées, leurs tests et leurs lignes de code, ont contribués à faire d'Apache le plus utilisé des serveurs Web du monde.

L'ancêtre d'Apache est le serveur libre développé par le National Center for Supercomputing Applications de l'université de l'Illinois. L'évolution de ce serveur s'est arrêtée lorsque le responsable a quitté le NCSA en 1994. Les utilisateurs ont continué à corriger les bugs et à créer des extensions qu'ils distribuaient sous forme de "patches" d'ou le nom "a patchee server". La version 1.0 de Apache a été disponible le 1 décembre 1995.

L'équipe de développement se coordonne par l'intermédiaire d'une liste de diffusion dans laquelle sont proposées les modifications et discutées les évolutions à apporter au logiciel. Les changements sont soumis à un vote avant d'être intégrés au projet. Tout le monde peut rejoindre l'équipe de développement, il suffit de contribuer activement au projet pour pouvoir être nommé membre du Apache Group.

Chapitre 2. Le protocole HTTP

HTTP est un protocole requête / réponse opérant au dessus de TCP. Le client ouvre une connexion TCP vers le serveur et envoie une requête. Le serveur analyse la requête et répond en fonction de sa configuration.

Par exemple, pour la requête :

```
GET / HTTP/1.0 <CR><LF><CR><LF>
```

Le serveur répond :

```
HTTP/1.1 200 OK
Date: Wed, 09 Dec 1998 17:44:56 GMT
Server: Apache/1.3.3 (Unix)
Last-Modified: Wed, 09 Dec 1998 09:45:16 GMT
ETag: "f712-d7-366e46ac"
Accept-Ranges: bytes
Content-Length: 215
Connection: close
Content-Type: text/html

<HTML>
<BODY>
<H1>Bienvenue chez LigerWine</H1>
<UL>
<LI><A href="anjou.html">Anjou</A>
<LI><A href="bourgueil.html">Bourgueil</A>
</UL>
<BR>
<BR>
LigerWine SA
3, rue RABELAIS
37512 Trouperdu
</BR>
</BODY>
</HTML>
```

2.1. Définitions

URL : Unified Resource Locator

ex : http://www.apache.org/download/apache_1_3_3_tar.gz

URI : Unified Resource Indicator

ex : /download/apache_1_3_3_tar.gz

Chapitre 3. Compilation et installation

Ce support de cours est basé sur la version 1.3.3 du logiciel Apache.

La première étape consiste à récupérer la distribution la plus récente des sources du logiciel. La source la plus à jour est le site du groupe de développement Apache : http://www.apache.org. A partir de ce site, vous pouvez être orienté vers un site miroir plus proche de chez vous.

Le fichier récupéré est une archive **tar** compressée qu'il faut installer dans un répertoire source :

```
bash$ cd /usr/src
bash$ tar xvzf apache_1_3_3_tar.gz
```

La lecture du fichier archive crée le répertoire `apache_1.3.3` et extrait les fichiers dans ce répertoire. Il est conseillé de lire les fichiers `README`, `README.configure` et `INSTALL` avant de procéder à la compilation et à l'installation du logiciel.

3.1. Configuration

La configuration du logiciel consiste à exécuter le script **configure** avec les arguments personnalisant l'installation.

Les principales options de configuration sont :

Option	**Signification**
–layout	affiche les répertoires d'installation et de fonctionnement.
–help	affiche la signification des options
–verbose	affiche plus de messages lors de la configuration
–quiet	n'affiche aucun message lors de la configuration
–prefix=répertoire	chemin d'installation du logiciel (ex : /opt/apache)
–enable-module=nom	valide le module dont le nom est donné
–disable-module=nom	dévalide le module dont le nom est donné
–add-module=fichier	ajoute le fichier dans le répertoire des modules, et dans le fichier de configuration et valide le module
–enable-suexec	valide l'utilisation de suexec

Dans la plupart des cas, on utilise :

```
bash$ ./configure –prefix=/opt/apache
```

3.2. Compilation

La compilation est exécutée simplement en tapant la commande make. Il est possible de garder une trace de la compilation en exécutant :

```
bash$ make > make.log 2>&1 &
bash$ tail -f make.log
```

3.3. Installation

La phase d'installation est aussi facile que la compilation. Elle s'effectue en tapant la commande **make install**. De la même façon, il possible de garder une trace de l'installation en tapant :

```
bash$ make install > install.log 2>&1 &
bash$ tail -f install.log
```

Après cette phase, le serveur Apache est prêt à fonctionner moyennant une petite configuration.

Dans la suite de ce document, nous allons présenter des directives de configuration du serveur Apache. Le fichier d'aide contient une description détaillée de toutes ces directives sous forme d'un manuel de référence. La syntaxe utilisée dans les descriptions suit le format suivant :

- Syntaxe : décrit le format de la directive
- Défaut : si la directive à une valeur par défaut, celle-ci est notée
- Contexte : spécifie dans quel contexte la directive peut etre utilisée (configuration du serveur, hote virtuel, directory et .htaccess)
- Origine : informe de l'endroit ou est implémenté la directive (coeur, base, extension, expérimental)
- module : le nom du module implémentant la directive.

Chapitre 4. Le premier site

4.1. Introduction

Nous allons parcourir les options de configuration d'Apache en nous mettant à la place du responsable du site de la société (imaginaire) LigerWine négociant en vins de Loire. Les dirigeants de cette société veulent à tout prix utiliser les moyens les plus modernes de promotion et de vente et pensent qu'Internet est l'avenir du commerce.

Le premier site à réaliser est un site "vitrine" présentant la société au monde entier. Nous partons du principe que nous disposons d'une machine raccordée à Internet de façon permanente et que le nom de domaine ligerwine.com a été déposé.

Pour pouvoir faire fonctionner le serveur, nous allons créer un utilisateur http appartenant au groupe http. Dans le répertoire `/opt/apache`, nous créons deux petits scripts pour le lancement et l'arrêt du serveur.

```
bash$ cd /opt/apache
bash$ cat > go
#!/bin/sh
sbin/httpd -f $1
^D
bash$chmod +x go
bash$ cat > stop
#!/bin/sh
sbin/apachectl stop
^D
bash$chmod +x stop
```

Le paramètre du script de lancement est le chemin d'accès au fichier de configuration du serveur **httpd**.

La documentation d'Apache propose d'utiliser trois fichiers de configuration en fonction des directives utilisées :

- le fichier `httpd.conf` contient les directives de contrôle du démon
- le fichier `srm.conf` contient les directives de spécification des documents fournis par le serveur
- le fichier `access.conf` contient les directives de contrôle d'accès aux documents.

Nous allons utiliser un seul fichier `httpd.conf` pour la présentation de nos exemples. Nous signifions au serveur de ne pas chercher les autres fichiers en spécifiant les directives :

```
AccessConfig /dev/null
ResourceConfig /dev/null
```

Les fichiers de ce premier exemple se trouvent dans le répertoire `/opt/formation/site1`. Le fichier de configuration minimal `conf/httpd.conf` contient :

```
User http
Group http
AccessConfig /dev/null
ResourceConfig /dev/null
```

```
ServerName localhost
DocumentRoot /opt/formations/site1/html
```

Le tableau ci-dessous présente la signification des directives utilisées.

Directive	**Signification**
User	Identificateur de l'utilisateur pour l'exécution du serveur
Group	Identificateur de groupe pour l'exécution du serveur
ServerName	Nom du serveur pour la redirection des URL
DocumentRoot	Répertoire racine des fichiers publiés

Le chemin spécifié dans la directive `DocumentRoot` est ajouté aux noms des fichiers demandés dans un URL. Par exemple, la demande http://www.ligerwine.com/prix.html va chercher le fichier `/opt/formations/site1/html/prix.html`.

Lorsque l'on se connecte sur le site, le serveur envoie la liste des fichiers contenus dans le répertoire donné par la directive `DocumentRoot`.

Le fichier `var/log/error_log` contient les enregistrements d'erreurs détectées par le serveur. Apache permet d'envoyer les enregistrements d'erreurs à d'autres endroits en utilisant la directive `ErrorLog`. La directive `LogLevel` permet de spécifier le niveau d'enregistrement :

Niveau	**Signification**
debug	les messages de mise au point
info	informations de fonctionnement
notice	conditions normales mais significatives
warn	petits problèmes non graves
error	conditions d'erreur
crit	niveau critique perturbant le fonctionnement
alert	il faut réagir immédiatement
emerg	système inutilisable

Lorsqu'un niveau donné est demandé, tous les messages des niveaux inférieurs sont affichés.

4.2. Un site plus évolué

Le premier site présenté n'étant pas satisfaisant en terme de présentation, il faut créer quelques fichiers contenant du code HTML pour permettre la création de liens entre fichiers et offrir plus de possibilités de navigation. Le premier fichier du site s'appelle `index.html` car c'est le nom du fichier affiché par défaut lors de la connexion sur le site. Il est possible de fixer d'autres noms avec la directive `DirectoryIndex`. Les fichiers d'exemple se trouvent dans le répertoire `/opt/formations/site2`.

Il est possible de spécifier des directives supplémentaires pour ajuster le fonctionnement du serveur.

Directive	Signification
StartServers	Permet de définir le nombre de serveurs lancés au démarrage.
MinSpareServers	Nombre minimum de serveurs en attente
MaxSpareServers	Nombre maximum de serveurs en attente
MaxRequestsPer-Child	Nombre maximum de requêtes traitées par un processus (0 => infini)
MaxClients	Nombre maximum de serveurs créés
ServerAdmin	Permet de spécifier une adresse mail pour les messages d'erreur.
ServerSignature	Permet d'ajouter une ligne de signature aux messages générés automatiquement (On ou EMail)
TimeOut	Temps maximum d'attente pour qu'une requte soit complétée.
HostNameLookup	Permet d'enregistrer le nom d'un client plutôt que son adresse dans les fichiers d'enregistrement

Ces directives sont utilisées dans le site exemple `/opt/formations/site3`.

Le serveur Apache gère, par défaut, la notion de connexion persistante qui permet de gérer plusieurs requêtes HTTP à travers la même connexion TCP. Cette fonction entraîne un gain de performance notable dans le chargement de pages possédant de nombreuses images. Cependant, cette fonction n'est active que pour les fichiers dont la taille est connue à l'avance. Cela ne fonctionne donc pas avec des scripts CGI.

Les directives de contrôle de cette fonction sont :

Directive	Signification
KeepAlive	Spécifie le nombre de requètes gérées par la même connexion (0 dévalide la fonction)
KeepAliveTimeout	Temps maximum d'attente entre deux requêtes

4.3. Les enregistrements

Apache permet de configurer de nombreux enregistrements de son activité. La directive `TransferLog` spécifie la destination de l'enregistrement défini par une directive `LogFormat` antérieure. Il est possible d'enregistrer des informations dans un fichier ou de les envoyer sur un tube vers un processus. Si aucune directive `LogFormat` n'a été spécifiée, `TransferLog` enregistre les messages en CLF (Common Log Format).

La directive `CustomLog` permet de générer des enregistrements liés à un nom de format ou à un format spécifié en option.

Le format CLF enregistre, pour chaque requête, une ligne contenant :

```
hôte identité autorisation date requête code taille
```

avec :

- host : adresse IP ou nom complet du client
- identité : si la directive IdentityCheck est validée et que le client répond
- autorisation : l'identificateur de l'utilisateur si le document est protégé
- date : la date et l'heure de la requête
- requête : la ligne de requête du client
- code : le code à trois chiffres envoyé au client
- taille : la taille en octets du document envoyé sans prendre en compte les entêtes

Si un champ n'a pas de valeur, il est remplacé par un tiret (-).

Un exemple est donné dans le site `/opt/formations/site4`.

4.4. La gestion des accès

Le site `/opt/formations/site5` met en évidence un problème de sécurité. Si le client demande l'URL http://localhost/Anjou/, le serveur lui retourne la liste des fichiers du répertoire.

La directive `Options` permet, entre autres, d'éliminer ce problème en utilisant le paramètre `-Indexes`. Plus généralement, elle permet de spécifier ce qui est autorisé dans un répertoire donné :

Paramètre	**Description**
All	toutes les options sauf Multiviews
ExecCGI	l'exécution de scripts est autorisée
FollowSymLinks	le serveur suivra les liens symboliques rencontrés dans le répertoire
Includes	permet l'utilisation de SSI
IncludesNOEXEC	permet l'utilisation de SSI sauf les directives #exec et #include
Indexes	permet l'affichage des répertoires
MultiViews	permet la négociation de contenu
SymLinksIfOwner-Match	le serveur suivra les liens symboliques rencontrés dans le répertoire si le fichier pointé appartient à l'uid d'exécution

4.5. Les serveurs virtuels

Pour éviter d'avoir à lancer autant de serveurs Apache que de sites à gérer, il faut utiliser les blocs `VirtualHost`. Ceux ci permettent de spécifier des directives applicables à un serveur virtuel.

La syntaxe est :

```
<VirtualHost adresseIP>
...
</VirtualHost>
```

Le paramètre `adresseIP` peut être donnée sous forme d'adresse ou de nom complet.

Il existe deux méthodes de déclaration de serveurs virtuels :

- le système doit fournir une adresse IP différente pour chaque serveur virtuel. La directive `Listen` permet de spécifier que le serveur attend des connexions sur une adresse IP et un port particulier.
- le système ne fournit qu'une adresse IP et la discrimination s'effectue par le nom. La directive `NameVirtualHost` définit l'adresse IP sur laquelle seront accrochés les serveurs virtuels. Ce mode pose un problème avec les navigateurs générant des requêtes HTTP/1.0, car le nom n'est pas émis dans la requête. La directive `ServerPath` permet, en partie, de palier le problème.

Une utilisation importante des serveurs virtuels avec de nombreux fichiers d'enregistrement peut générer des problèmes de disponibilité d'identificateurs de fichiers.

Un exemple est donné dans `/opt/formations/site6`.

4.6. Les blocs de spécifications

4.6.1. Bloc Directory

Un bloc `Directory` permet de spécifier un ensemble d'options pour un ou plusieurs répertoires. La syntaxe est :

```
<Directory dir>
...
</Directory>
```

L'argument `dir` peut être une expression régulière permettant de spécifier un ensemble de répertoires.

Ce bloc peut contenir des directives de contrôle d'accès :

Directive	Signification
allow from	donne la liste des clients ayant accès à ce bloc
deny from	donne la liste des clients dont l'accès est refusé
order	spécifie l'ordre dans lequel sont évaluées les directives allow et deny

La directive `order` peut prendre trois arguments :

- "deny, allow" : la directive deny est évaluée avant la directive allow. Si aucune ne convient, l'accès est autorisé (valeur par défaut)
- "allow, deny" : la directive allow est évaluée avant la directive deny. Si aucune ne convient, l'accès est interdit

- "mutual-failure" : seules les machines données dans la directives allow et non données dans la directive deny sont autorisées.

La directive `DirectoryMatch` fonctionne de façon similaire avec une expression régulière comme argument.

4.6.2. Bloc Location

Le bloc `Location` permet de spécifier des directives particulières pour un URL donné.

La syntaxe est :

```
<Location URL
...
</Location>
```

La directive `LocationMatch` fonctionne de façon similaire avec une expression régulière comme argument.

4.6.3. Bloc Files

Le bloc `Files` permet de spécifier des directives particulières pour un ou plusieurs fichiers.

La syntaxe est :

```
<Files noms>
...
</Files>
```

La directive `FilesMatch` fonctionne de façon similaire avec une expression régulière comme argument.

4.7. Le fichier .htaccess

Il est possible d'ajouter un fichier `.htaccess` dans chaque répertoires à partir de `DocumentRoot`. Ce fichier contient des directives particulières pour le répertoire où il se trouve ainsi que pour tous les sous-répertoires. Le nom de ce fichier est spécifié par la directive `AccessConfigFile`.

Ce mécanisme permet de modifier des droits d'accès sans avoir à redémarrer le serveur. Par contre, cela oblige le serveur à analyser ce fichier à chaque accès, pour tous les répertoires du chemin d'accès.

La directive `AllowOverride` permet de spécifier quelles directives peuvent être modifiées par le fichier `.htaccess`. La valeur par défaut autorise toutes les modifications.

Les arguments peuvent être :

Arguments	Signification
All	autorise tout (valeur par défaut)
None	interdit tout (le fichier .htaccess n'est pas lu)

AuthConfig	permet d'utiliser les directives contrôlant les autorisation d'accès
FileInfo	permet d'utiliser les directives contrôlant les types de document
Indexes	permet d'utiliser les directives de contrôle de l'affichage des répertoires
Limit	permet de spécifier le contrôle des accès client (allow, deny, order)
Options	permet d'utiliser les directives de contrôles des actions permises dans les répertoires

4.8. L'ordre d'évaluation

L'ordre d'évaluation de ces directives est donné ci-dessous. Dans le cas des directives `Directory`, l'évaluation est effectuée du chemin le plus court au chemin le plus long. En cas de conflit, l'évaluation est effectuée dans l'ordre des déclarations.

- <Directory> (sauf avec des expressions régulières) et le fichier .htaccess sont évalués simultanément (.htaccess à la préscéance sur <Directory>)
- <DirectoryMatch> et <Directory> avec des expressions régulières
- <Files> et <FilesMatch>
- <Location> et <LocationMatch>

Les directives données dans les sections `VirtualHost` sont évaluées après celles de la configuration générale.

Chapitre 5. Les scripts CGI

Après le site vitrine, la génération de pages dynamiques permet d'obtenir une interaction beaucoup plus forte avec le visiteur du site. C'est la porte ouverte sur les applications multi-couches, le commerce électronique, etc. Les exemples de ce chapitre sont donnés dans le répertoire `/opt/formations/site7`.

C'est le module mod_cgi qui offre la gestion des scripts CGI. Ce module est inclus par défaut dans la configuration standard d'Apache.Il faut soit valider l'option `ExecCGI`, soit avoir une directive `ScriptAlias` permettant de définir le répertoire contenant les scripts. La meilleure solution consiste a utiliser `ScriptAlias` pour mettre les scripts en dehors du répertoire pointé par `DocumentRoot`.

La directive `ScriptAlias` est utilisée en associant un chemin pour les URL à un chemin dans le système de fichiers local :

```
ScriptAlias cheminURL répertoire
```

La directive `ScriptLog` permet de spécifier un fichier d'enregistrement des traces d'exécution des scripts CGI. La directive `ScriptLogLength` permet de limiter la taille de ce fichier. `ScriptLogBuffer` spécifie la taille maximum enregistrée pour une requête POST ou PUT.

La directive `ScriptLog` ne doit être utilisée que pour la mise au point des scripts.

En plus des variables standard de l'interface CGI, les variables suivantes peuvent être positionnées :

- REMOTE_HOST : si la directive HostnameLookups est validée et que la recherche DNS du client a réussi.
- REMOTE_IDENT : si la directive IDENTITY_CHECK est validée et que le client a répondu à la demande
- REMOTE_USER : si le script CGI est sujet à authentification

Le module mod_env permet de passer des variables d'environnement aux scripts CGI. Ces variables peuvent être définies par la directive `SetEnv` ou héritées de l'environnement du serveur grace à la directive `PassEnv`. La directive `UnsetEnv` permet d'enlever des variables.

le module mod_setenvif permet de positionner des variables d'environnement en fonction de conditions sur la requête du client.

La directive `BrowserMatch` permet de définir des variables en fonction du contenu du champ `User-Agent` de l'entête HTTP. La directive `BrowserMatchNoCase` fonctionne de la même façon sans tenir compte de la casse des caractères.

Les directives `SetEnvIf` et `SetEnvIfNoCase` permettent de positionner des variables en fonction de la valeur des attributs de la requête :

- soit des champs de l'entête (Host, User-Agent, Referer)
- soit d'autres informations liées à la requête (Remote_Host, Remote_addr, Remote_User, Request_Method, Request_URI)

5.1. les gestionnaires (handlers)

Un gestionnaire est une procédure interne à Apache permettant d'effectuer des actions lorsque un fichier est demandé. L'association entre un fichier et un gestionnaire est effectuée soit en fonction de sa position dans le système de fichier, soit en fonction de l'extension du fichier. Les gestionnaires standards intégrés à Apache sont :

- send-as-is: envoie le fichier tel quel
- cgi-script: gère le fichier comme un script CGI
- imap-file: gère le fichier comme un fichier image-map
- server-info: récupère les informations sur le serveur
- server-parsed: interprète pour la gestion SSI
- server-status: Récupère l'état du serveur
- type-map: interprète pour la négociation de contenu

La directive `AddHandler` permet d'associer une extension de fichier à un gestionnaire existant.

La directive `SetHandler` est utilisée dans un bloc Directory ou Location pour associer tous les fichiers de ce bloc au gestionnaire spécifié.

5.2. Sécurité et suEXEC

Le programme suExec permet d'exécuter des scripts avec un identificateur d'utilisateur différent de celui du serveur. La configuration de cet outil doit être exécutée soigneusement pour ne pas créer de trous de sécurité dans l'utilisation du serveur.

La configuration et l'installation de cette fonctionnalité se réalise en plusieurs étapes :

- configurer le fichier `suexec.h` pour l'adapter à l'environnement d'exécution
- compiler suexec avec la commande **make suexec**
- ajouter le support de suexec dans Apache
- installer l'exécutable **suexec** dans le répertoire donné avec les droits `setuid` du super utilisateur (root)

Ce programme peut être utilisé en spécifiant des directives `User` et `Group`, avec un utilisateur et un groupe différents de ceux du serveur, dans un bloc `VirtualHost`. Dans ce cas, les scripts CGI sont exécutés avec les identificateurs donnés. Il peut aussi être utilisé directement dans les scripts CGI.

Chapitre 6. Authentification

La gestion de l'authentification permet de limiter l'accès à tout ou partie du site à des utilisateurs autorisés. Pour ce faire, il faut créer un base de données des utilisateurs avec un mot de passe pour chacun d'eux. Cette base de données est stockée dans un fichier dont le format est semblable à celui du fichier `/etc/passwd` d'Unix. Ce fichier est géré avec le programme **htpasswd** livré avec Apache dans le sous-répertoire `support`.

Il faut créer le fichier des utilisateurs en dehors du répertoire pointé par `DocumentRoot`. La création s'effectue par :

```
bash$ htpasswd c nom_du_fichier utilisateur
New password:
Re-type new password:
```

Le fichier est lisible et modifiable avec un éditeur de texte mais les mots de passe sont cryptés.

La configuration du serveur consiste à définir un espace dans lequel l'authentification est nécessaire. Ceci est effectué dans un bloc Directory en spécifiant les directives suivantes :

```
AuthName nom
AuthType Basic
AuthUserFile /opt/apache/etc/passwd
require valid-user
```

La directive `AuthName` donne le nom de l'espace protégé. L'autorisation est associée à ce nom et permet de créer plusieurs zones protégées par la même identification. Cette identification est renvoyée automatiquement par le navigateur à la demande du serveur.

La directive `AuthType` spécifie le type de contrôle effectué. Seul le mot `Basic` peut être utilisé aujourd'hui. L'option `Digest` est en cours de standardisation et permettra plus de sécurité dans la procédure d'authentification.

La directive `AuthUserFile` donne le nom du fichier contenant la liste des utilisateurs autorisés. Il est possible de spécifier une directive `AuthGroupFile` donnant le nom d'un fichier de groupes d'utilisateurs. Ce fichier a une syntaxe proche de celle du fichier `/etc/group` d'Unix.

la directive `require` permet de spécifier les utilisateurs ayant accès à la ressource protégée. Elle prend en paramètre une liste d'utilisateurs, de groupes ou le mot `valid-user`.

Le bloc `Limit` permet de spécifier des directives particulières en fonction de la méthode HTTP utilisée.

Ce mécanisme n'est pas très efficace pour un grand nombre d'utilisateurs. Le module mod_auth_dbm permet d'utiliser les fichier dbm pour stocker les informations d'authentification utilisateur. Ce système utilise des paires clé et valeur avec un index sur les clés. En connaissant la clé, on retrouve très vite la valeur associée. Le module n'est pas inclut dans la construction standard d'Apache.

La gestion de la base de données est effectuée par le programme **dbmmanage** qui utilise les commandes suivantes :

Commande	Signification
adduser	pemet d'ajouter un utilisateur

delete	détruit un utilisateur
check	vérifie le mot de passe associé à un utilisateur
view	affiche le contenu de la base de données

Le principe de fonctionnement est identique en remplaçant la directive AuthUserFile par `AuthDBMUserFile`. Pour la gestion des groupes, on utilise la directive `AuthDBMGroupFile`.

Il existe d'autres modules de connexion à des bases de données telles que mSQL, PostgreSQL, etc...

La directive `IdentityCheck` permet d'effectuer une requête vers le démon **identd** de la machine cliente (RFC 1413). Cependant cette information est rarement disponible et, lorsqu'elle est disponible, son intégrité est sujette à caution. Cette directive peut générer des temps de réponse très longs.

Le module mod_anon permet, lorsqu'une politique d'accès est définie, d'autoriser un utilisateur anonyme à se connecter sur le site.

Les directives associées sont :

Directive	**Signification**
Anonymous	permet de spécifier l'identificateur d'utilisateur à utiliser
Anonymous_NoUserID	permet de laisser les champs d'identification vides
Anonymous_LogEmail	les accès sont enregistrés dans un fichier
Anonymous_VerifyEmail	le mot de passe doit contenir '@' et '.'
Anonymous_MustGiveEmail	il faut donner un mot de passe sous forme d'une adresse email
Anonymous_Authoritative	les autres possibilités d'authentification ne sont pas testées

Par exemple, les directives :

```
Anonymous invite anonyme
Anonymous_NoUserID off
Anonymous_LogEmail on
Anonymous_VerifyEmail on
Anonymous_MustGiveEmail on
Anonymous_Authoritative off
```

autorisent les utilisateurs à s'enregistrer avec les noms invite et anonyme en fournissant, comme mot de passe, leur adresse email. Tous les accès sont enregistrés dans le fichier `httpd_log`. Les utilisateurs référencés utilisent leur identificateur habituel.

Chapitre 7. Affichage des répertoires

Dans le cas ou le fichier `index.html` n'existe pas dans un répertoire et que l'option `Indexes` est validée, Apache retourne une liste des fichiers du répertoire demandé. L'affichage de cette liste peut être ajusté grâce au module mod_autoindex.

Un exemple de site est donné dans `/opt/formations/site9`.

La directive `FancyIndexing` permet de valider la gestion de l'affichage des répertoires par le serveur. Celui-ci gère l'affichage d'une ligne d'entête de colonne permettant le tri en fonction du critère affiché (nom, date de modification, taille et description).

Le contrôle de l'affichage est effectué grâce aux directives suivantes :

Directive	Signification
AddIcon	permet de spécifier un fichier image associé à un type de fichier donné
AddIconByEncoding	permet de spécifier un fichier image associé à un type d'encodage MIME
AddIconByType	permet de spécifier un fichier image associé à un type MIME
DefaultIcon	permet de spécifier l'icône par défaut lorsque aucun type ne correspond
HeaderName	nom du fichier à inclure en entête de l'affichage
IndexIgnore	spécifie les fichiers qui ne doivent pas être affichés
ReadmeName	nom du fichier à inclure à la fin de l'affichage
AddDescription	permet de spécifier le texte de description d'un fichier
AddAlt	permet de spécifier un texte associé à un type de fichier donné
AddAltByEncoding	permet de spécifier un texte associé à un type d'encodage MIME
AddAltByType	permet de spécifier un texte associé à un type MIME

La directive `IndexOptions` permet de contrôler l'affichage :

Option	Signification
IconHeight=pixels	donne la hauteur des icônes en pixels
IconWidth=pixels	donne la largeur des icônes en pixels
IconsAreLinks	permet d'utiliser les icônes comme des liens
NameWidth	permet de forcer la largeur de la colonne nom
ScanHTMLTitle	Affiche le titre des documents HTML

SuppressColumnSorting	n'affiche pas l'entête de tri
SuppressDescription	n'affiche pas la description
SuppressLastModified	n'affiche pas la date de modification
SuppressSize	n'affiche pas la taille

Chapitre 8. La négociation de contenu

Apache a la possibilité de servir les documents dans le format demandé par le programme client. Cela concerne les types MIME retournés ainsi que le langage et le jeux de caractères utilisés. Cette possibilité est offerte par le module mod_negociation qui est intégré par défaut lors de la compilation.

Apache supporte les spécifications HTTP/1.1 pour la négociation de contenu. Il reconnaît les entêtes `Accept`, `Accept-Language`, `Accept-Charset` et `Accept-Encoding`.

Il existe deux possibilités de gestion du contenu :

- par la directive Option `Multiviews` qui doit être positionnée pour les répertoires dans lesquels la négociation est possible.
- un fichier type-map qui contient la liste exacte des variantes.

8.1. L'options Multiviews

8.1.1. Les images

Le navigateur expose ses préférences dans l'entête de la requête par une ligne du type :

```
Accept : image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
```

Dans les pages HTML générées, il ne faut pas donner les extensions des fichiers images. Lorsque le navigateur demandera une image, Apache renverra le format disponible le plus adapté à la demande.

8.1.2. Le langage

La directive `AddLanguage` permet de spécifier une langue dans laquelle peuvent être transmis les documents. La directive associe une extension sur les noms des fichiers à une demande de langage sous forme de type MIME. Les documents doivent être réalisés dans les langues supportées.

La directive `LanguagePriority` permet au serveur de fixer la priorité des langages dans le cas où le client n'émet pas de préférences.

8.2. Le fichier type-map

Le fichier `type-map` doit posséder une entrée pour chaque variante du document dans un format compatible avec la RFC 822, les définitions étant séparées par une ligne vide. Il est possible de spécifier un critère de qualité de la variante avec le mot qs. La valeur spécifiée peut varier de 0.000 à 1.000, cette dernière étant la valeur par défaut.

Par exemple :

```
URI: sandra
```

```
URI: sandra.jpeg
  Content-type: image/jpeg; qs=0.8

  URI: sandra.gif
  Content-type: image/gif; qs=0.5

  URI: sandra.txt
  Content-type: text/plain; qs=0.01
```

Les descriptions utilisent les termes suivants :

- URI : donne le fichier contenant la variante, la localisation est relative au fichier de description
- Content-type: type du fichier en prenant en compte le jeu de caractères et le niveau de qualité (type MIME)
- Content-language: le code du langage
- Content-encoding: le codage du document (x-compress ou x-gzip)
- Content-length: la taille du fichier

Chapitre 9. Le mode mandataire

Le module mod_proxy permet de faire fonctionner Apache en mandataire pour les protocoles HTTP et FTP.

La directive `ProxyRequests` permet de faire fonctionner Apache en mode mandataire. Les directives associées sont :

Directive	**Signification**
ProxyRemote	permet de spécifier un autre mandataire pour un URL ou un protocole donné
NoProxy	permet de spécifier des adresses qui ne passent pas par le proxy défini par ProxyRemote.
ProxyPass	permet de fonctionner apparemment en site miroir
ProxyPassReverse	permet de rediriger les réponses d'une machine pseudo miroir
AllowCONNECT	donne la liste des ports pour CONNECT (https)
ProxyBlock	permet d'interdire l'accès aux sites spécifiés par les options
ProxyDomain	spécifie le domaine DNS pour un intranet

Apache permet de gérer un cache lorsqu'il fonctionne en mode mandataire. Les directives associées sont :

Directive	**Signification**
CacheRoot	donne le répertoire dans lequel seront stockés les fichiers.
CacheSize	spécifie la taille du cache en kilo-octets
CacheGcInterval	donne l'intervalle de temps entre deux tests du cache pour vérifier la taille
CacheMaxExpire	durée de validité d'un document
CacheLastModifiedFactor	facteur de correction pour le calcul de la date de validité (fonction de la dernière modification)
CacheDirsLevels	nombre maximum de niveaux de répertoires dans le cache
CacheDirLength	longueur d'un chemin dans le cache
CacheDefaultExpire	donne une durée de validité par défaut pour les protocoles n'en fournissant pas
NoCache	permet de spécifier des critères pour les documents ne devant pas être dans le cache

Chapitre 10. Les scripts inclus

Le module mod_include permet de gérer des scripts intégrés aux documents fournis par le serveur (Server Side Includes). Le traitement du script est effectué si l'option `Includes` est validée pour le fichier. Il est possible d'utiliser la directive `XbitHack` pour vérifier l'autorisation d'exécution au niveau du fichier. La déclaration du gestionnaire de scripts doit être donnée dans le fichier de configuration :

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

La syntaxe générale d'une commande est :

```
<!-#element attribut=valeur attribut=valeur ... ->
```

avec les éléments suivants :

- config : permet de contrôler l'interprétation :
  - errmsg : donne le message en cas d'erreur d'interprétation
  - sizefmt : donne le format d'affichage de la taille (bytes ou abbrev)
  - timefmt : donne le format d'affichage des temps (voir strftime())
- echo : permet d'afficher une variable
- exec : permet d'exécuter un script en fonction de l'attribut :
  - cgi : pour les scripts CGI
  - cmd : pour un script shell
- fsize : affiche la taille du fichier en fonction de l'attribut
  - file : relatif au répertoire du script
  - virtual : URL relatif s'il ne commence pas par "/"
- flastmod affiche la date de modification du fichier (idem fsize)
- include : insère le texte d'un autre document dans le fichier courant ; l'option IncludeNOEXEC empêche l'exécution des scripts (les attributs sont les mêmes que pour fsize)
- printenv : affiche les variables d'environnement
- set : initialise une variable :

  ```
  <!-#set var="nom" value="sandra" ->
  ```

- if, elif, else et endif : exécution conditionnelle

  ```
  <!-#if expr="test_condition" ->
  <!-#elif expr="test_condition" ->
  <!-#else ->
  ```

```
<!-#endif ->
```

Un exemple est donné dans `/opt/formations/site11`.

Chapitre 11. le contrôle du fonctionnement

Le site donné dans `/opt/formations/site10` permet d'avoir accès aux informations de configuration et d'état du serveur.

Le module mod_status permet de définir un URI pour l'accès à l'état du serveur. Les informations affichées sont :

- la date courante
- la date du dernier démarrage
- la durée de fonctionnement
- le nombre total d'accès et le trafic généré
- l'état du processeur de machine hôte
- le nombre de requêtes par secondes
- le nombre d'octets par secondes
- le nombre d'octets par requête
- le nombre de requête en cours
- le nombre de serveurs en attente
- l'état des serveurs

La directive `ExtendedStatus` permet d'obtenir en plus le détail de l'état pour chaque requête.

Il est possible d'obtenir un affichage rafraîchi à une périodicité donnée en spécifiant `?refresh=T`, ou T est la période en secondes, à la suite de l'URI.

Le module mod_info permet d'afficher la configuration courante du serveur.

Chapitre 12. La redirection

La redirection des URL permet d'accéder à des documents situés ailleurs que sous le répertoire donné par `DocumentRoot`.

Les directives sont :

Directive	**Signification**
Alias	définit un chemin à substituer dans les URL pour les fichiers standards
ScriptAlias	comme pour Alias mais sur les scripts
AliasMatch	permet d'utiliser un expression régulière pour la concordance
Redirect	permet de renvoyer un URL modifié vers le client associé à un mot d'état
RedirectMatch	comme Redirect avec un expression régulière
RedirectTemp	la redirection est temporaire
RedirectPermanent	la redirection est permanente

Le module mod_rewrite permet de transformer les URL reçus à l'aide de règles basées sur des expressions régulières. Il n'y a pas de limites sur le nombre de règles pouvant être appliquées ; les définitions pouvant être effectuées au niveau du serveur, du serveur virtuel, du bloc `Directory` ou du répertoire.

Chapitre 13. Les modules

Apache offre la possibilité de rajouter des modules pour effectuer une tâche non prise en compte par les modules actuels. Pour ajouter un nouveau module dans Apache, il faut :

- créer un répertoire pour contenir les sources du module dans src/modules
- copier les fichiers du module example
- modifier les fichiers
- ajouter la ligne suivante dans le fichier `src/Configuration.tmpl`

```
Module nom_module        modules/nom/mod_nom.o
```

- compiler et installer Apache

Le reste de ce chapitre présente le module example fournit en exemple avec les sources d'Apache. Ce module affiche une trace des appels de ces fonctions.

Les fonctions de l'API Apache sont déclarées dans les fichiers du répertoire `include`. Elles sont toutes préfixées par la macro API_EXPORT.

13.1. Les réservoirs

Apache utilise la notion de réservoir (pool) pour la gestion des ressources internes. La destruction d'un réservoir entraîne la destruction des ressources qui lui sont associées. Il faut éviter d'utiliser les fonctions standards de gestion de mémoire.

Cela permet aux modules de ne pas avoir à gérer la mémoire notamment sur les conditions d'erreur.

Les fonctions de manipulation des réservoirs sont déclarées dans `include/alloc.h`.

13.2. La structure d'un module

La structure module est l'interface entre le serveur et le module. Elle contient des pointeurs sur les fonctions et les données du module.

```
module example_module =
{
    STANDARD_MODULE_STUFF,
    example_init,               /* module initializer */
    example_create_dir_config,  /* per-directory config creator */
    example_merge_dir_config,   /* dir config merger */
    example_create_server_config, /* server config creator */
    example_merge_server_config,  /* server config merger */
    example_cmds,               /* command table */
    example_handlers,           /* [7] list of handlers */
    example_translate_handler,  /* [2] filename-to-URI translation */
    example_check_user_id,      /* [5] check/validate user_id */
```

```
    example_auth_checker, /* [6] check user_id is valid *here* */
    example_access_checker,     /* [4] check access by host address */
    example_type_checker,       /* [7] MIME type checker/setter */
    example_fixer_upper,        /* [8] fixups */
    example_logger,             /* [10] logger */
#if MODULE_MAGIC_NUMBER >= 19970103
    example_header_parser,      /* [3] header parser */
#endif
#if MODULE_MAGIC_NUMBER >= 19970719
    example_child_init,         /* process initializer */
#endif
#if MODULE_MAGIC_NUMBER >= 19970728
    example_child_exit,         /* process exit/cleanup */
#endif
#if MODULE_MAGIC_NUMBER >= 19970902
    example_post_read_request   /* [1] post read_request handling */
#endif
{;
```

MODULE_MAGIC_NUMER est défini dans le fichier `include/ap_mmn.h` et sert à tracer les modification de l'API d'Apache.

STANDARD_MODUL_STUFF permet d'initialiser l'entête de la structure avec les valeurs par défaut. La plupart des champs seront initialisés à l'exécution.

Les gestionnaires du module qui retournent un entier peuvent générer les valeurs suivantes :

- OK : le gestionnaire a traité la requête
- DECLINED : le gestionnaire ne fait rien
- HTTP_xxxx : un des codes du protocole HTTP (voir httpd.h)

13.2.1. example_init

Cette fonction est appelée à l'initialisation du serveur avant qu'il accepte les requêtes. Elle est exécutée de nouveau à chaque configuration du serveur.

La syntaxe est :

```
static void example_init(server_rec *s, pool *p)
```

13.2.2. example_create_dir_config

Cette fonction est appelée une fois avec le paramètre `dirspec` égal à NULL à l'initialisation du serveur principal et pour chaque bloc `Location`, `Directory`, `File` ou fichier `.htaccess` dans lequel apparaît une directive du module.

La syntaxe est :

```
static void *example_create_dir_config(pool *p, char *dirspec)
```

La fonction retourne le pointeur alloué.

13.2.3. example_merge_dir_config

Cette fonction est appelée pour fusionner deux structures liées à un répertoire dans le cas d'un héritage.

La syntaxe est :

```
static void *example_merge_dir_config(pool *p, void *parent_conf, void *newloc_conf)
```

La fonction retourne le nouveau pointeur alloué pour le répertoire.

13.2.4. example_create_server_config

Cette fonction crée la structure liée au serveur pour le module. Elle est appelée une fois pour le serveur principal et ensuite pour chaque serveur virtuel.

La syntaxe est :

```
static void *example_create_server_config(pool *p, server_rec *s)
```

La fonction retourne le pointeur alloué.

13.2.5. example_merge_server_config

Cette fonction est appelée pour chaque serveur virtuel avec la structure allouée pour le serveur principal. Cela donne la possibilité de gérer des héritages éventuels.

La syntaxe est :

```
static void *example_merge_server_config(pool *p, void *server1_conf, void *server2_conf)
```

La fonction retourne le nouveau pointeur alloué pour le serveur virtuel.

13.2.6. example_cmds

C'est un tableau de structures du type command_rec de description des directives.

```
static const command_rec example_cmds[] =
{
    {
        "Example",          /* directive name */
        cmd_example,        /* config action routine */
        NULL,               /* argument to include in call */
        OR_OPTIONS,         /* where available */
```

```
        NO_ARGS,            /* arguments */
        "Example directive - no arguments"
                            /* directive description */
    {,
    {NULL{
{;
```

La description des arguments s'effectue avec `cmd_how` :

```
enum cmd_how {
    RAW_ARGS,       /* cmd_func parses command line itself */
    TAKE1,          /* one argument only */
    TAKE2,          /* two arguments only */
    ITERATE,        /* one argument, occuring multiple times
 * (e.g., IndexIgnore) */
    ITERATE2,       /* two arguments, 2nd occurs multiple times * (e.g., AddIcon) */
    FLAG,           /* One of 'On' or 'Off' */
    NO_ARGS,        /* No args at all, e.g. </Directory> */
    TAKE12,         /* one or two arguments */
    TAKE3,          /* three arguments only */
    TAKE23,         /* two or three arguments */
    TAKE123,        /* one, two or three arguments */
    TAKE13          /* one or three arguments */
{;
```

La fonction appelée pour gérer la directive est :

```
static const char *cmd_example(cmd_parms *cmd, void *mconfig)
{
    excfg *cfg = (excfg *) mconfig;

    /*
     * "Example Wuz Here"
     */
    cfg->local = 1;
    trace_add(cmd->server, NULL, cfg, "cmd_example()");
    return NULL;
{
```

Le paramètre `cmd_parms` est initialisé par le serveur :

```
typedef struct {
    void *info;      /* Argument to command from cmd_table */
    int override;     /* Which allow-override bits are set */
    int limited;      /* Which methods are <Limit>ed */
    configfile_t *config_file;  /* Config file structure from
                                 * pcfg_openfile() */
    ap_pool *pool;      /* Pool to allocate new storage in */
    struct pool *temp_pool; /* Pool for scratch memory
                             * persists during
```

```
                            * configuration, but wipe
                            * before the first
                            * request is served */
    server_rec *server; /* Server_rec being configured for */
    char *path;         /* If configuring for a directory,
                         * pathname of that directory.
                         * NOPE!  That's what it mean
                         * previous to the
                         * existance of <Files>, <Location>
                         * and regex
                         * matching.  Now the only usefulness
                         * that can
                         * be derived from this field is
                         * whether a command
                         * is being called in a server
                         * context (path == NULL)
                         * or being called in a dir context
                         * (path != NULL).*/
    const command_rec *cmd; /* configuration command */
    const char *end_token;  /* end token required to end a
                             * nested section */
{ cmd_parms;
```

13.2.7. example_handlers

Il s'agit d'un tableau de structures effectuant l'association entre le nom servant au référencement et le gestionnaire.

```
static const handler_rec example_handlers[] =
{
    {"example-handler", example_handler{,
    {NULL{
{;
```

La fonction du gestionnaire est appelée pour tous les documents qu'elle est sensée gérer.

La syntaxe est :

```
static int example_handler(request_rec *r)
```

13.2.8. example_translate_handler

Cette fonction permet au module de traduire l'URL en un nom de fichier. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

La syntaxe est :

```
static int example_translate_handler(request_rec *r)
```

13.2.9. example_check_user_id

Cette fonction permet au module de vérifier les informations d'authentification. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

La syntaxe est :

```
static int example_check_user_id(request_rec *r
```

13.2.10. example_auth_checker

Cette fonction permet au module de vérifier si la ressource demandée requiert une autorisation. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

La syntaxe est :

```
static int example_auth_check(request_rec *r)
```

13.2.11. example_access_checker

Cette fonction permet au module de vérifier les conditions d'accès à la ressource. Le premier module qui ne retourne pas DECLINED est supposé avoir effectué le travail.

La syntaxe est :

```
static int example_access_checker(request_rec *r)
```

13.2.12. example_type_checker

Cette fonction permet au module de fixer le type du document. S'il retourne OK, aucun autre module n'est appelé.

La syntaxe est :

```
static int example_type_checker(request_rec *r)
```

13.2.13. example_fixer_upper

Cette fonction est appelée pour effectuer les dernières modifications sur les entêtes.

La syntaxe est :

```
static int example_fixer_upper(request_rec *r)
```

13.2.14. example_logger

Cette fonction permet au module d'effectuer les enregistrements de traces qu'il souhaite.

La syntaxe est :

```
static int example_logger(request_rec *r)
```

13.2.15. example_header_parser

Cette fonction permet au module d'avoir accès à l'entête de la requête au début du processus.

La syntaxe est :

```
static int example_header_parser(request_rec *r)
```

13.2.16. example_child_init

Cette fonction est appelée à l'initialisation du processus serveur avant qu'il accepte les requêtes. Cela permet d'exécuter des actions qui ne doivent être exécutées qu'une fois par processus.

La syntaxe est :

```
static void example_child_init(server_rec *s, pool *p)
```

13.2.17. example_child_exit

Cette fonction est appelée à l'arrêt du processus serveur.

La syntaxe est :

```
static void example_child_exit(server_rec *s, pool *p)
```

13.2.18. example_post_read_request

Cette fonction est appelée après la lecture de la requête mais avant les autres phases. Cela permet, par exemple, de positionner des variables d'environnement.

La syntaxe est :

```
static int example_post_read_request(request_rec *r)
```