Web

Pour **accéder à un site web**, il vous faut **utiliser un client Web**, appelé communément **Navigateur**, par exemple : firefox, lynx, opera, konqueror, w3m… Vous devez **spécifier** en plus du **nom** ou de l'**adresse IP**, le **protocole** utilisé.

Celui qui nous intéresse est HTTP. Un document Hypertexte est un document contenant des hyperliens. Ceux-ci permettent de lier les pages les unes avec les autres. Ainsi, vous pouvez naviguer grâce à des liens sur les pages.

Il existe une version sécurisé du HTTP le HTTPS.
Afin de **différencier quel protocole** on utilise, on leur **réserve un port**, par defaut le **80 pour le mode non sécurisé** et le **443 pour le mode sécurisé**.Présentation de HTTP

- HTTP - HyperText Transfert Protocol - fondé par Tim Berners Lee, développé et utilisé par le WWW à partir de 1990.
- Protocole adapté au transfert d'information multimédia.
- Léger et rapide, à coût d'exploitation très bas.
- Introduit la notion d'hypertexte, c'est à dire que l'information de navigation est prise en compte dans le document, mais ne prend pas en charge le procédé complet de navigation.

- Le protocole HTTP sert à la communication entre le client et le serveur. Il s'agit en fait d'un dérivé du protocole FTP. Lors d'une communication, le logiciel client se connecte en TCP sur le serveur et télécharge en FTP le document désigné. Il coupe aussitôt la communication avec le serveur. S'il y a dans le document HTML plusieurs composants (comme des images, la plupart du temps), ce processus se répète autant de fois qu'il y a d'éléments constituant la page.

- L'avantage de ce processus est de limiter au maximum le temps d'occupation du serveur, de façon qu'il n'y ait pas d'engorgement de ce dernier.

Il est extrêmement simple d'étudier son fonctionnement en utilisant le programme telnet. Pour se connecter au serveur web il suffit de faire :
telnet adresse_ip_serveur_web 80
où *adresse_ip_serveur* est l'adresse ip du serveur web cible

La partie principale de la requête en ce qui concerne le client est la commande, placée sur la première ligne de la reqûete. Elle détermine l'action à effectuer. Ces commandes sont parfois accompagnées d'arguments qui précisent l'objet sur lequel porte la commande. L'action la plus courament utilisée est la récupération d'un document en utilisant la commande GET ou POST (GET et POSt sont deux méthodes HTTP différentes pour effectuer la même action). Cette commande prend comme argument la ressource à récupérer, c'est à dire le chemin vers le fichier à récupérer (ce n'est pas un chemin absolu, mais relatif à la racine du serveur). Par exemple :

```
GET /index.html HTTP/1.0
```

les clients placent une entête dans leur requête, à la suite de la commande, décrivant par exemple qui ils sont (le nom et le numéro de version pour un navigateur par exemple) et ce qu'ils peuvent accepter (des images, du texte au format HTML pour le même type de logiciel par exemple). Certaines entêtes sont optionelles et d'autres obligatoires
pour utiliser ce type d’enttête, il faut mentionner à la suite de la requête la version du protocole utilisée. (en général HTTP 1.0 ou HTTP/1.0 voire HTTP 1.1 ou HTTP/1.1)

Par exemple :

```
telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
GET /test.html HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 17 Apr 2008 07:39:21 GMT
Server: Apache/2.2.3 (Debian) mod_jk/1.2.18 mod_python/3.2.10 Python/2.4.4
PHP/5.2.0-8+etch10 mod_ssl/2.2.3 OpenSSL/0.9.8c mod_perl/2.0.2 Perl/v5.8.8
Last-Modified: Thu, 17 Apr 2008 07:16:12 GMT
ETag: "160488-9e-6883c700"
Accept-Ranges: bytes
Content-Length: 158
Connection: close
Content-Type: text/html

<html>

<head>
  <title>tester</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h1 >tester</h1>

</body>
</html>
Connection closed by foreign host.
```

La requête utilisée dans cet exemple est la plus simple que l'on puisse trouver, elle se compose d'un seule ligne qui comprend trois éléments : la méthode, l'url (elle identifie la ressource, dans la plupart des cas sur Internet il s'agit d'un simple fichier texte ou d'une image) et la version du protocole HTTP utilisé (HTTP/1.0 ou HTTP/1.1).

En plus de cette ligne on peut trouver un certain nombre de champ (1 par ligne) dont la forme est toujours la même, le nom du champ, suivi de : et d'un espace et la valeur que l'on veut lui donner (toujours suivi des caractères \r et \n). Les caractères \r et \n correspondent respectivement au retour chariot et saut de ligne. Vient ensuite une ligne vide, composée donc seulement des deux caractères \r et \n et le corps de la requête. Une requête a donc la forme suivante :

Méthode url HTTP/1.0\r\n
Champ1 : valeur 1\r\n
Champ2 : valeur 2\r\n
\r\n
Ceci est le corps de ma requête ...

Il existe cinq méthodes :

Méthode	Description
GET	Requête de la ressource située à l'URL spécifiée
HEAD	Requête de la ressource située à l'URL spécifiée (la réponse ne contient que l'entête, et pas le contenu de la ressource)
POST	Envoi de données au programme situé à l'URL spécifiée (le corps de la requête peut être utilisé)
PUT	Envoi de données à l'URL spécifiée (idem POST)
DELETE	Suppression de la ressource située à l'URL spécifiée

Les deux méthodes vraiment utilisées sont GET et POST (les amateurs d'html reconnaîtront des mots clefs familiers à la construction d'un formulaire). La méthode GET est la plus simple car le corps du message dans ce type de requête est vide. La méthode POST permet d'envoyer des informations au serveur dans le corps du message d'une requête HTTP. Lorsque des informations sont envoyées au serveur à l'aide de la méthode GET, elles sont encodées à la suite de la ressource après le symbole '?' dans l'url.

La méthode HEAD sert essentiellement pour les applications de cache. En effet la réponse HTTP renseigne sur les propriétés de la ressource demandée (date de dernière modification, ...), il est donc intéressant pour économiser du temps de traitement et de la bande passante de pouvoir ne demander que ces informations et pas le contenu de la ressource (qui se trouve quand même très souvent être un simple fichier).

Exemple:

```
telnet 127.0.0.1 80
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
HEAD / /HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 05 Jan 2005 19:15:26 GMT
Server: Apache-AdvancedExtranetServer/2.0.48 (Mandrake Linux/6mdk)
mod_perl/1.99_11 Perl/v5.8.3 mod_ssl/2.0.48 OpenSSL/0.9.7c PHP/4.3.4
Accept-Ranges: bytes
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug
Connection closed by foreign host.
```

Dans la pratique bien peu de serveurs autorisent les actions de type PUT et DELETE pour des raisons évidentes de sécurité.

Codes HTTP courants

http://www.indexa.fr/CodesHTTP.html

Lorsque vous naviguez sur le web, votre navigateur génère des requêtes

HTTP aux serveurs, qui répondent en retournant soit le document demandé, soit un code d'erreur : voici les codes d'erreur HTTP les plus courants et leur signification.

Codes : 1xx 2xx 3xx 4xx 5xx

1xx : Codes d'information / *Information codes*		
Code	Statut / *Status*	Description / *Comment*
100	**Continuer**	**Attente de la suite de la requête.** La partie initiale de la requête a bien été reçue et le client peut continuer avec la suite de cette requête.
101	**Changement de protocoles**	**Le serveur accepte la requête du client de changer de protocole.** Le client a demandé au serveur d'utiliser un autre protocole que celui actuellement utilisé, et le serveur accepte cette requête.

2xx : Codes de succès / *Success codes*		
Code	Statut / *Status*	Description / *Comment*
200	**OK**	**La requête HTTP a été traitée avec succès.** L'information retournée avec la réponse dépend de la méthode utilisée dans la requête. Par exemple la réponse à une requête GET classiquement émise par un navigateur web sera la ressource demandée (c'est-à-dire une page HTML, une image, etc).
201	**Créé**	**La requête a été correctement traitée et a résulté en la création d'une nouvelle ressource.** Cette ressource peut être référencée par l'URI retournée dans le corps de la réponse, avec l'URL la plus précise pour la ressource indiquée dans l'en-tête du champ "Location".
202	**Accepté**	**La requête a été acceptée pour être traitée, mais son traitement peut ne pas avoir abouti.** Ce code est utilisé en remplacement du 201 lorsque le traitement ne peut pas avoir lieu immédiatement, son résultat est donc indéterminé.
203	**Information non certifiée**	**L'information retournée n'a pas été générée par le serveur HTTP mais par une autre source non authentifiée.**
204	**Pas de contenu**	**Le serveur HTTP a correctement traité la requête mais il n'y a pas d'information à envoyer en retour.** Cela peut par exemple se produire lorsqu'un fichier HTML ou le résultat d'un programme CGI-BIN est vide.
205	**Contenu réinitialisé**	**Le client doit remettre à zéro le formulaire utilisé dans cette transaction.** Ce code est envoyé au logiciel de navigation quand il doit réinitialiser un formulaire généré dynamiquement par un CGI-BIN, par exemple.
206	**Contenu partiel**	**Le serveur retourne une partie seulement de la taille demandée.** Ce code est utilisé lorsqu'une requête spécifiant une taille a été transmise.

3xx : Codes de redirection / *Redirection codes*		
Code	Statut / *Status*	Description / *Comment*
300	**Choix multiples**	**L'URI demandée concerne plus d'une ressource.** Par exemple, l'URI concerne un document qui a été traduit en plusieurs langues. Le serveur doit retourner des informations indiquant comment choisir une ressource précise.
301	**Changement d'adresse définitif**	**La ressource demandée possède une nouvelle adresse (URI).** Toute référence future à cette ressource doit être faite en utilisant l'une des URIs retournées dans la réponse. Le navigateur web doit normalement charger automatiquement la ressource demandée à sa nouvelle adresse.
302	**Changement d'adresse temporaire**	**La ressource demandée réside temporairement à une adresse (URI) différente.** Cette redirection étant temporaire, le navigateur web doit continuer à utiliser l'URI originale pour les requêtes futures.
303	**Voir ailleurs**	**L'URI spécifié est disponible à un autre URI et doit être demandé par un GET.**
304	**Non modifié**	**Le navigateur web a effectué une requête GET conditionnelle et l'accès est autorisé, mais le document n'a pas été modifié.** Cette réponse classique signifie que vous avez configuré votre navigateur pour utiliser un cache HTTP (proxy) dans lequel une copie du document demandé est déjà stockée. Le proxy a donc demandé au serveur si le document original a changé depuis, et a reçu cette réponse : il pourra ainsi utiliser la copie locale.
305	**Utiliser le proxy**	**L'URI spécifié doit être accédé en passant par le proxy.**

4xx : Erreur du client / *Client Error*

Code	Statut / *Status*	Description / *Comment*
400	**Mauvaise requête**	**La requête HTTP n'a pas pu être comprise par le serveur en raison d'une syntaxe erronée.** Le problème peut provenir d'un navigateur web trop récent ou d'un serveur HTTP trop ancien.
401	**Non autorisé**	**La requête nécessite une identification de l'utilisateur.** Concrètement, cela signifie que tout ou partie du serveur contacté est protégé par un mot de passe, qu'il faut indiquer au serveur pour pouvoir accéder à son contenu.
402	**Paiement exigé**	**Ce code n'est pas encore mis en oeuvre dans le protocole HTTP.**
403	**Interdit**	**Le serveur HTTP a compris la requête, mais refuse de la traiter.** Ce code est généralement utilisé lorsqu'un serveur ne souhaite pas indiquer pourquoi la requête a été rejetée, ou lorsqu'aucune autre réponse ne correspond (par exemple le serveur est un Intranet et seules les machines du réseau local sont autorisées à se connecter au serveur).
404	**Non trouvé**	**Le serveur n'a rien trouvé qui corresponde à l'adresse (URI) demandée.** Cela signifie que l'URL que vous avez tapée ou cliquée est mauvaise ou obsolète et ne correspond à aucun document existant sur le serveur (vous pouvez essayez de supprimer progressivement les composants de l'URL en partant de la fin pour éventuellement retrouver un chemin d'accès existant).
405	**Méthode non autorisée**	**Ce code indique que la méthode utilisée par le client n'est pas supportée pour cet URI.**

Code	Statut / *Status*	Description / *Comment*
406	**Aucun disponible**	**L'adresse (URI) spécifiée existe, mais pas dans le format préféré du client.** Le serveur indique en retour le langage et les

Code	Statut / *Status*	Description / *Comment*
		types d'encodages disponibles pour cette adresse.
407	**Authentification proxy exigée**	**Le serveur proxy exige une authentification du client avant de transmettre la requête.**
408	**Requête hors-délai**	**Le client n'a pas présenté une requête complète pendant le délai maximal qui lui était imparti, et le serveur a abandonné la connexion.**
409	**Conflit**	**La requête entre en conflit avec une autre requête ou avec la configuration du serveur.** Des informations sur les raisons de ce conflit doivent être indiquée en retour.
410	**Parti**	**L'adresse (URI) demandée n'existe plus et a été définitivement supprimée du serveur.**
411	**Longueur exigée**	**Le serveur a besoin de connaître la taille de cette requête pour pouvoir y répondre.**
412	**Précondition échouée**	**Les conditions spécifiées dans la requête ne sont pas remplies.**
413	**Corps de requête trop grand**	**Le serveur ne peut traiter la requête car la taille de son contenu est trop importante.**
414	**URI trop long**	**Le serveur ne peut traiter la requête car la taille de l'objet (URI) a retourner est trop importante.**
415	**Format non supporté**	**Le serveur ne peut traiter la requête car son contenu est écrit dans un format non supporté.**
416	**Plage demandée invalide**	**Le sous-ensemble de recherche spécifié est invalide.**
417	**Comportement erroné**	**Le comportement prévu pour le serveur n'est pas supporté.**

5xx : Erreur du serveur / *Server Error*		
Code	Statut / *Status*	Description / *Comment*
500	**Erreur interne du serveur**	**Le serveur HTTP a rencontré une condition inattendue qui l'a empêché de traiter la requête.** Cette erreur peut par exemple être le résultat d'une mauvaise configuration du serveur, ou d'une ressource épuisée ou refusée au serveur sur la machine hôte.
501	**Non mis en oeuvre**	**Le serveur HTTP ne supporte pas la fonctionnalité nécessaire pour traiter la requête.** C'est la réponse émise lorsque le serveur ne reconnaît pas la méthode indiquée dans la requête et n'est capable de la mettre en oeuvre pour aucune ressource (soit le navigateur web est trop récent, soit le serveur HTTP est trop ancien).
502	**Mauvais intermédiaire**	**Le serveur intermédiaire a fourni une réponse invalide.** Le serveur HTTP a agi en tant qu'intermédiaire (passerelle ou proxy) avec un autre serveur, et a reçu de ce dernier une réponse invalide en essayant de traiter la requête.
503	**Service indisponible**	**Le serveur HTTP est actuellement incapable de traiter la requête en raison d'une surcharge temporaire ou d'une opération de maintenance.** Cela sous-entend l'existence d'une condition temporaire qui sera levée après un certain délai.
504	**Intermédiaire hors-délai**	**Cette réponse est identique au code 408 (requête hors-délai), mais ici c'est un proxy ou un autre intermédiaire qui a mis trop longtemps à répondre.**
505	**Version HTTP non supportée**	**La version du protocole HTTP utilisée dans cette requête n'est pas (ou plus) supportée par le serveur.**

Installation

Apache est composé de plusieurs paquets.

La bibliothèque, le serveur et ses outils

Nom	Rôle
libapr1	Apache's Portable Runtime Library, bibliothèque de fonctions standards portables.
apache2	Ce paquet contient le serveur.
apache2.2-common	Ce paquet contient les modules standards apache2, qui incluent le support SSL.
apache2-utils	Outils pour serveurs web.

Le MPM

Le MPM[1], indispensable, est le moteur du serveur, la manière dont il intercepte les requêtes. Il en existe plusieurs à vous de choisir en fonction de vos besoins mais le `mod-php5` ne supporte que le *prefork* :

Nom	Commentaire
apache2-mpm-prefork	Modèle traditionnel pour Apache2 version sans thread, intercepte les requêtes à la manière Apache 1.3, utile pour éviter la mise en thread pour la compatibilité avec les bibliothèques non-thread-safe. C'est le meilleur mpm pour isoler les requêtes.
apache2-mpm-worker	Modèle à processus haute vitesse pour Apache2 version avec thread, il est considérablement plus rapide que la version traditionnelle et **c'est le MPM recommandé**.

1 *Multi-Processing Module*, module multi traitements

Installation

Apache2*, les bases et le mod chroot*:

installer:apache2 apache2-doc apache2-utils apache2-mpm-prefork libapache2-mod-chroot libapache2-mod-auth-pam libapache2-mod-auth-sys-group

Au minimum:

```
sudo apt-get install apache2 apache2-doc apache2-mpm-prefork
```

Si vous voulez utiliser PHP5:

Installez les paquets **php5-common php5 php5-gd libapache2-mod-php5 + les bibliothèques dont vous vous servez dans vos pages. php5-xsl** (par exemple)

Si vous voulez utiliser MYSQL:

Installez les paquets mysql-server libapache2-mod-auth-mysql et pour son aspect pratique mysql-client

Un petit truc, pour ne "presque" rien oublier, installez phpmyadmin depuis les dépots en premier; il "oublie" juste d'installer mysql-server.

Configuration

Tous les **fichiers de configuration** de apache2 sont dans le dossier :

/etc/apache2

La structure de configuration d'Apache2 a été éclatée en plusieurs fichiers, chacun rangé dans un répertoire dédié à une tâche précise. Ceci améliore grandement l'interaction avec les scripts d'installation qui activent et configurent des modules pour vous (entre autres). La structure encourage également à utiliser une façon saine de configurer un serveur Apache.

/etc/apache2/apache2.conf	Fichier de configuration de base, ne pas toucher
/etc/apache2/ports.conf	Pour écouter sur des ports autre que 80 (ex: 443 pour HTTPS)
/etc/apache2/envvars	Environnement Apache (ex: ORACLE_HOME, TNS_ADMIN, etc.)
/etc/apache2/conf.d	Configuration globales des applications web
/etc/apache2/mods-available	Modules disponibles (activation et configuration)
/etc/apache2/mods-enabled	Modules activés (activation et configuration)
/etc/apache2/sites-available	Sites disponibles/configurés
/etc/apache2/sites-enabled	Sites activés

- apache2.conf

Le fichier de configuration principal

- conf.d/

Les fichiers de ce répertoire sont inclus par la directive suivante dans apache2.conf:

Include generic snippets of statements
Include /etc/apache2/conf.d

C'est un bon répertoire pour ajouter des configurations supplémentaires.
On y trouve le fichier "charset" qui contient une ligne:
#AddDefaultCharset UTF-8 qu'il faut laisser (ou mettre) en commentaires

- httpd.conf: Fichier vide
- magic
  donnée « magic » pour le module Apache mod_mime_magic Apache module, documenté dans htdocs/manual/mod/mod_mime_magic.html. Probablement inutile d'y toucher
- mods-available/
  Ce répertoire contient une série de fichiers en *.load et *.conf.
  Les .load contiennent les directives nécessaires pour charger les modules en question et les .conf les directives de configuration correspondantes mods-enabled/
  Pour qu'Apache puisse utiliser les modules, il est nécessaire de créer un **lien symbolique** dans le répertoire des .load (et .conf, s'il existe associé au module dans mods-available/. (voir plus loin)
  exemple:

  cgi.load -> /etc/apache2/mods-available/cgi.load

- mod-enable contient les liens vers les modules utilisés
- ports.conf
  Directives concernant les ports et ip sur lesquels on écoute.
  Listen
  Cette directive est (pour Apache2) indispensable. Elle spécifie les adresses IP des interfaces locales et les ports sur lesquels Apache doit être en écoute (par défaut les requêtes sont acceptées de toutes les interfaces IP, et donc en général seul(s) le(s) numéros de port(s) sont renseignés).
  Sur apache2, ces déclarations sont incluses dans *apache2.conf* par Include /etc/apache2/ports.conf lequel peut contenir par exemple, pour que le serveur accepte des connexions à la fois sur les ports 80 et 8080
  Listen 80
  Listen 8080
  Pour autoriser un serveur à accepter des connexions sur deux "sockets" qualifiés, écrire :
  Listen 192.170.2.1:80
  Listen 192.170.2.5:8000

- sites-available/
  Même principe que mods-available/, sauf que cela concerne les configurations pour plusieurs hotes virtuels utilisés dans Apache 2.

- sites-enabled/
  contient des liens symboliques vers des sites

Activation/désactivation des modules d'Apache

Pour voir les modules disponibles tapez la commande

```
a2enmod
```

puis pour activer le module

```
a2enmod <nom_module>
```

Pour voir les modules chargés tapez la commande

```
a2dismod
```

puis pour désactiver le module

```
a2dismod <nom_module>
```

a2ensite and **a2dissite** idem mais pour les sites

Redémarrer Apache2

```
sudo /etc/init.d/apache2 restart
```

Il est aussi possible de redémarrer apache2 sans "tuer" les requêtes en cours :

```
apache2ctl graceful
```

Cette solution est plus élégante sur un serveur en production.

apache2.conf

Voir:http://www.ac-creteil.fr/reseaux/systemes/linux/lamp/apache2-configuration.html

Configuration de base

ServerName www.monsite.be
Le nom doit correspondre à une adresse IP, donc être renseigné dans un serveur DNS (car la machine hôte est jointe par son adresse IP)
Si aucun nom n'est spécifié, alors le serveur tente de déduire un nom en procédant à un "lookup inverse" à partir de l'adresse IP.

> **Première observation** : le serveur a donc un nom pour chacune de ses implantations. Il faut donc distinguer entre un exécutable qui s'appelle **httpd**, et l'application locale du serveur auquel un nom particulier est donné.
> **Deuxième observation** : le nom du serveur est associé à celui de votre site.

Petit problème:

apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName

... waiting apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName

[OK]

vous avez oublié servername dans apache2.conf

- ServerRoot "/etc/apache2"
  Il s'agit du répertoire où le serveur trouvera son répertoire de configuration
- **PidFile** /var/run/apache2.pid
  C'est le fichier où le serveur en exécution stocke son premier numéro de processus (PID)

Réglages d'exécution

- **Timeout** 300
  Paramètre important qui fixe le temps (en ms) d'attente maximum du serveur d'une réponse à une requete envoyée à un programme extérieur (comme un gestionnaire de base de données)
- **KeepAlive**on
  MaxKeepAliverequests 100
  KeepAliveTimeout 15
  Autorise les connexions persistantes d'un client, afin de lui permettre l'envoi de plusieurs requetes sans déconnexion, avec un plafond fixé pour un client, pour servir aussi d'éventuels autres clients ! et un temps d'attente maxi de la requete suivante provenant du meme client.
- **MinSpareServers 4 MaxSpareServers** 20
  Nombres maximum et minimum de processus serveurs devant être en permanence disponibles, en attente de nouvelles connexions clientes
- **StartServers** 5

Nombre de processus serveurs démarrés à l'initialisation, en plus du processus père.
ps aux|grep apache donne le nombre de PID enfant propriété de *www-data*.

- **MaxClients** 20
  Nombre maximum de processus qu'Apache peut lancer et gérer simultanément. Ce nombre ne peut pas excéder 254
- **MaxRequestsPerChild** 500
  Nombre maximum de requetes HTTP traitées par un processus enfant avant qu'il ne soit éliminé.

Gestion et suivi des connexions

- La commande `tail -f /var/log/apache2/access.log > /dev/tty11` & affiche les dernières requêtes au serveur Apache, "en direct" sur la pseudo-console 11 (pour l'activer : alt-F11).
  On peut utiliser aussi `less /var/log/apache/access.log` puis `Shift-F` pour basculer à la fin du fichier en attente de données (Ctrl-C pour finir)

Les types MIME

- Beaucoup de types de fichiers spécifiques sont appelées par les pages WEB. Le serveur se doit de reconnaitre ces types pour envoyer au navigateur client des directives pour permettre de leur associer une application susceptible de les prendre en charge. Cela s'effectue par l'intermédiaire de la nomenclature standard appelée MIME (à l'origine destinée à reconnaitre les types des pièces jointes dans les mails.
- Le fichier mime.types Le fichier de configuration contient la directive **TypesConfig /etc/mime.types** (ligne 222) qui spécifie le fichier qui contient la liste des correspondances entre les types et les extensions de fichiers. La directive **DefaultType text/plain** indique le type par défaut.
- On peut spécifier de nouvelle association dans les divers fichiers de conf (y compris ceux se trouvant dans *mods-available/*)

Fichiers inclus:

Observer le résultat de la commande

```
grep  -in include  /etc/apache2/apache2.conf|grep -v "#"
```

```
204:Include /etc/apache2/mods-enabled/*.load
205:Include /etc/apache2/mods-enabled/*.conf
208:Include /etc/apache2/httpd.conf
211:Include /etc/apache2/ports.conf
233:Include /etc/apache2/conf.d/
236:Include /etc/apache2/sites-enabled/
```

UserDir public_html

Ce paramètre (placé ici dans */etc/apache2/mods-available/userdir.conf*) signifie que n'importe quel utilisateur peut publier ses pages WEB personnelles dans un sous-répertoire de son répertoire perso, lequel doit être obligatoirement nommé *public_html*.
Pour l'utilisateur toto, c'est dans */home/toto/public_html*.
Sa page d'accueil sera alors accessible par l'URL : *http://serveur/~toto* , où `serveur` est le nom du serveur ou son adresse IP.

Problème de php dans public_html

Visiblement, le php n'est pas reconnu par le serveur dans « **public_html** »
D'où vient le mal ?
Après beaucoup de chipotages et de prières à saint Google,
voir le site de Marco Rodrigues Merci à lui!!
il s'avère que le problème est dans un **module php5.conf**

```
1.<ifmodule mod_userdir.c>
2. <directory /home/*/public_html>
3. php_admin_value engine Off
4. </directory>
5.</ifmodule>
```

Même en mettant on au lieu de off, le résultat n'est pas garanti mais
en mettant tout en remarque (comme il est précisé !)

```
<IfModule mod_php5.c>
<FilesMatch « \.ph(p3?|tml)$ »>
SetHandler application/x-httpd-php
</FilesMatch>
<FilesMatch « \.phps$ »>
SetHandler application/x-httpd-php-source
</FilesMatch>
# To re-enable php in user directories comment the following lines
# (from <IfModule ...> to </IfModule>.) Do NOT set it to On as it
# prevents .htaccess files from disabling it.
#<IfModule mod_userdir.c>
#    <Directory /home/*/public_html>
# php_admin_value engine On
#   </Directory>
#</IfModule>
</IfModule>
```

et après un redémarrage du système (apache restart n’a pas suffit), le miracle a eu lieu!
Remarque, le module correspondant de la version 9.04 d’ubuntu contenait:
<IfModule mod_php5.c>
AddType application/x-httpd-php .php .phtml .php3
AddType application/x-httpd-php-source .phps
</IfModule>

Utilisation d'alias de répertoires

Il peut être utile de remplacer un chemin de répertoires par un nom symbolique. Ces répertoires alias peuvent être paramétrés comme les autres. Exemple significatif :

Il s'agit d'**accéder par l'alias doc aux doc HTML du serveur** Linux et de ses différentes applications et services installés, qui sont regroupées dans **/usr/share/doc**.
On réserve cette consultation aux machines du réseau local.
La stratégie consiste ici à interdire d'abord à tous (deny from all), puis on énumère les exceptions (allow from ..,..)
pour accéder à la doc directement avec l'url http://Serveur/doc
Alias /doc /usr/share/doc

Autorisations:

Deux possibilités existent :

1. utiliser des directives <directory> dans le fichier de configuration de apache,
2. ou bien à travers des fichiers « .htaccess ». Cependant, pour utiliser les fichiers « .htaccess », il faudrait que dans la directive <directory> du répertoire ou du répertoire parent il n’y ait pas cette ligne : AllowOverride None.

Multivues (option Multiviews)

MultiViews est une option qui s'applique à un répertoire, ce qui signifie qu'elle peut être activée à l'aide d'une directive Options à l'intérieur d'une section <Directory>, <Location> ou <Files> dans httpd.conf, ou (si AllowOverride est correctement positionnée) dans des fichiers .htaccess.
Notez que Options All n'active pas MultiViews; vous devez activer cette option en la nommant explicitement.

L'effet de `MultiViews` est le suivant : si le serveur reçoit une requête pour `/tel/répertoire/foo`, si `MultiViews` est activée pour `/tel/répertoire`, et si `/tel/répertoire/foo` n'existe *pas*, le serveur parcourt le répertoire à la recherche de fichiers nommés foo.*, et génère une correspondance de types (type map) qui liste tous ces fichiers, en leur associant les mêmes types de média et encodages de contenu qu'ils auraient eu si le client avait demandé l'accès à l'un d'entre eux par son nom. Il choisit ensuite ce qui correspond le mieux aux besoins du client.

Exemples:

Répertoire racine

<Directory />
Options FollowSymLinks
AllowOverride None
</Directory>
Pour des questions de sécurité laissé ça par défaut. Cela laisse les permissions de tous les répertoires par défaut et n'autorise pas d'access spéciaux même si un .htaccess existe (AllowOverride None).
De ce fait personne ne peux modifier les droits que vous avez imposé sur les répertoires où apache a accès.

<Directory /usr/share/doc>
order deny,allow
deny from all
permission à partir de localhost
allow from localhost, 127.0.0.1
permission à partir des stations du sous-domaine de l'établissement
allow from .ipeps.b
Options Indexes FollowSymLinks
</Directory>

explications:

<directory ***> début de la définition
order ordre dans lequel on teste les autorisations et les interdictions
None : Désactive toutes les options.
All : Active toutes les options SAUF Multiviews.
Indexes : Permet aux utilisateurs d'avoir des index générés par le serveur. C'est à dire si l'index du répertoire (index.htm le + souvent) est manquant, cela autorise le serveur a lister le contenu du répertoire (dangereux suivant les fichiers contenu dans ce répertoire).
FollowSymLinks : Autorise a suivre les liens symboliques.
ExecCGI : Autorise à exécuter des scripts CGI à partir de ce répertoire.
Includes : Autorise des fichiers include pour le serveur.
IncludesNOEXEC : Permet mais les includes mais empêche la commande EXEC (qui permet d'exécuter du code).
Multiviews : Autorise les vue multiples suivant un contexte. Par exemple permet d'afficher les pages dans un langage suivant la configuration du langage du client.
SymLinksIfOwnerMatch : Autorise a suivre les liens seulement si l'user ID du fichier (ou répertoire) sur lequel le lien pointe est le même que celui du lien.
AllowOverride : définit comment sont géré les fichiers .htaccess de ce répertoire:
All : Gère tout ce qui est dans .htaccess
AuthConfig : Active les directives d'autorisations AuthDBMGroupFile, AuthDBMUserFile, AuthGroupFile, AuthName, AuthType, AuthUserFile, require
FileInfo : Active les directives d'autorisations AddEncoding, AddLanguage, AddType, DefaultType, ErrorDocument, LanguagePriority

Limit : Active la directive d'autorisation Limit
None : Ne lis pas le fichier .htaccess et laisse les droits "Linux" de ce répertoire.
Options : Active la directive Option
Order : Donne l'ordre d'application des régles Allow Deny:
deny,allow : Applique les règles deny puis allow
allow,deny : Applique le srégles allow puis deny
Allow (ou deny):
Nom d'hôte : Autorise les hôtes spécifiés, les adresses IP, le nom de domaine, etc..(ou les refuse si la

règle est deny)
All : Autorise tout le monde (ou refu
</Directory> : Indique la fin des règles pour ce répertoire.

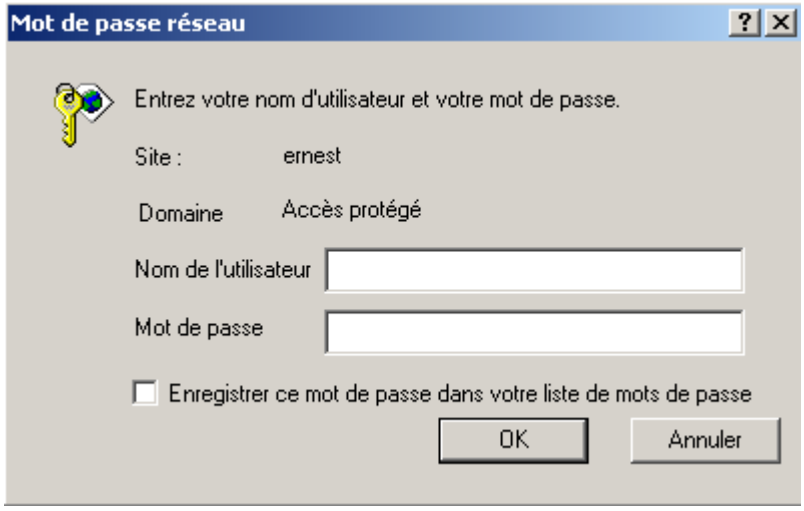
Répertoire cgi-bin dans un dossier public-_html

Dans le fichier **userdir.conf**

```
<IfModule mod_userdir.c>
        UserDir public_html
        UserDir disabled root

        <Directory /home/*/public_html>
                AllowOverride FileInfo AuthConfig Limit Indexes
                Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
                <Limit GET POST OPTIONS>
                        Order allow,deny
                        Allow from all
                </Limit>
                <LimitExcept GET POST OPTIONS>
                        Order deny,allow
                        Deny from all
                </LimitExcept>
        </Directory>
        <Directory /home/*/public_html/cgi-bin/>
        Options ExecCGI
        SetHandler cgi-script
        </Directory>
</IfModule>
```

Accès sécurisé



Doc tirée de http://ernest.cheska.net/htaccess/htaccess.shtml

Il existe beaucoup de méthodes pour securiser une partie d'un site web tournant sur Apache. La methode de l'htaccess est tres souple (on peut ajouter des utilisateurs à la volée sans relancer le serveur apache) mais ce n'est pas forcement la meilleure.

La méthode Basic

Tiré de http://www.ac-creteil.fr/reseaux/systemes/linux/lamp/apache2-authentification.html

- Directives Voici les directives usuelles et leur signification

Directive	Action
`AuthType basic`	type d'authentification communément adopté (fait circuler les mots de passe en clair)
`AuthName texte`	affichera ce texte comme invite dans une boite de dialogue
`AuthUserFile chemin/fichier`	précise le fichier qui contient les comptes et mots de passe des utilisateurs ayant droit d'accès
`Require valid-user` `Require liste-noms`	l'accès s'applique à tous les comptes du fichiers, ou seulement aux comptes énumérés dans la liste

Nous allons ajouter des directives pour le dossier que nous souhaitons protéger comme suit:

```
<Directory /var/www/html/>
AllowOverride AuthConfig
</Directory>
```

Ici je souhaite que le dossier **/var/www/html** accepte la directive AllowOverride AuthConfig. Cela implique que **tous ces sous-dossiers vont également accepter cette directive**.

Que signifie AllowOverride AuthConfig ?

Tout simplement qu'Apache daignera **lire votre fichier .htaccess** si il se trouve dans le repertoire en question (ou un de ses sous-repertoires)
Exemple de procédure Supposons que l'espace privé soit situé dans le répertoire `/var/www/prive` et son accès réservé à un ensemble d'utilisateurs : `admin, webmaster et toto`
Directives dans le fichier *default*

```
<Directory "/var/www/prive">
AuthType Basic
AuthUserFile /etc/apache/users
AuthName "Accès privé"
# autres clauses
# AuthGroupFile /etc/apache/groups

<limit GET>
# ATTENTION : GET en majuscules !
require valid-user
# require user toto dupond
# require group profs
</limit>
</Directory>
```

La procédure de création du fichier .htaccess :

Ça se fait tres simplement avec un éditeur et vous le placez dans le répertoire que vous souhaitez proteger. Voici ce que vous mettez dedans:
Lorsque vous souhaitez créer une zone d'identification sur votre site web via les fichiers **.htaccess** et **.htpasswd**, le contenu d'un fichier htaccess ressemble à ceci :

```
AuthUserFile /repertoire/mesmotsdepasse
AuthGroupFile /dev/null
AuthName "Accès protégé"
AuthType Basic
<limit GET POST>
require valid-user
</Limit>
```

En cas de problèmes, si la mise en place du fichier `.htaccess` ne change rien : le répertoire est peut être sous le contrôle du directive `AllowOverride None`.

Pour vérifier, vous pouvez intentionnellement ajouter une ligne erronnée dans le `.htaccess` et recharger la page. Si vous n'obtenez par la page "Internal Server Error", c'est que la cause du problème réside bien dans la directive `AllowOverride`.

La procédure de création du fichier .htpasswd :

Lorsque vous désirez ajouter ou créer un utilisateur, vous devez savoir crypter le mot de passe que vous désirez lui attribuer puisque mettre un mot de passe en clair dans le fichier ne fonctionnera pas. Apache dispose d'un utilitaire vous permettant de crypter vos mots de passe . *Cet utilitaire se trouve dans le repertoire /usr/sbin/htpasswd sous linux*
Il faut placer ce fichier là ou vous avez indiqué à .htaccess de le chercher.
Vous le créez tout simplement avec un editeur de texte. Il contient une ligne par utilisateur autorisé qui se décompose comme suit:

login:password
login2:password2

Ces mots de passe doivent être cryptés. Vous obtenez le mot de passe crypté au moyen de **la commande htpasswd.** Par exemple je souhaite que l'utilisateur toto accède à ma page web avec le mot de passe 'supertoto', je tape la commande:
htpasswd -c /var/www/.htpasswd toto

[root@Ernest stephane]# htpasswd -c /var/www/.htpasswd toto
New password:
Re-type new password:
Adding password for user toto

Bien sur dans cet exemple on ne voit pas les mots de passe mais j'ai bien tapé supertoto deux fois de suite. Il y a maintenant un fichier .htpasswd dans /var/www avec le user toto et son mot de passe crypté.

Exemple en tant qu'utilisateur local:

[michel@moi_Bureau michel]$ /usr/sbin/htpasswd -c ./.htpasswd tata
New password:
Re-type new password:
Adding password for user tata
[michel@moi_Bureau michel]$ cat .htpasswd
tata:BW9fry1ZFcDdo
[michel@moi_Bureau michel]$ /usr/sbin/htpasswd ./.htpasswd toto
New password:
Re-type new password:
Adding password for user toto
[michel@moi_Bureau michel]$ cat .htpasswd
tata:BW9fry1ZFcDdo
toto:/DSunBSY9WZsY

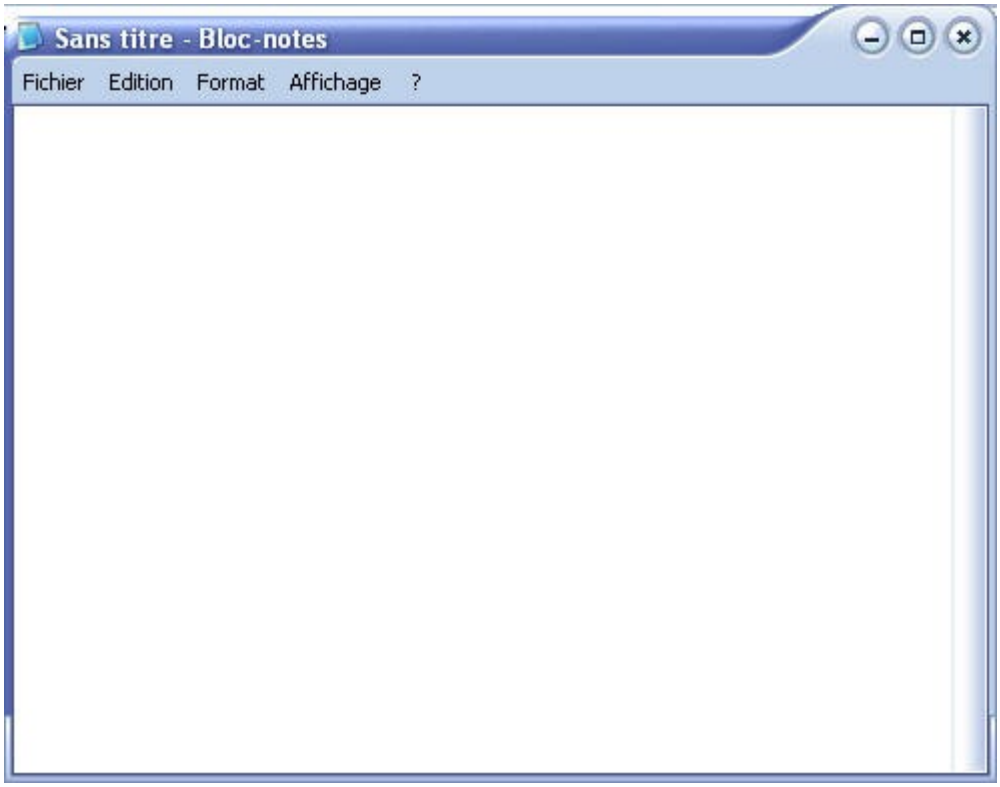
Créer le .htaccess sur un serveur distant

http://www.siteduzero.com/tuto-3-152-1-proteger-un-dossier-avec-un-htaccess.html

La première étape est de créer sur votre disque dur un fichier appelé ".htaccess". Mais là, vous allez certainement avoir un problème (ça commence fort :lol:)
En effet, Windows n'aime pas les fichiers qui commencent par un point. Pour tous les autres systèmes d'exploitation (Mac OS, Linux) vous n'aurez aucun problème. Mais Windows lui il veut pas, allez savoir pourquoi

On va utiliser une astuce : on va dans un premier temps créer un fichier appelé htaccess.txt, et plus tard avec notre logiciel FTP on le renommera en .htaccess (et là ça marchera !).

Commencez donc par ouvrir Bloc-Notes par exemple :



Là dedans, on va rentrer des informations qui n'ont rien à voir avec du HTML ou du PHP : ce sont des instructions pour le serveur. Elles vont expliquer au serveur que seules certaines personnes sont autorisées à accéder au dossier. Copiez-y ce code :

Code : Apache
```
AuthName "Page d'administration protégée"
```

```
AuthType Basic
AuthUserFile "/home/sdz/www/gestion/admin/.htpasswd"
Require valid-user
```

Parmi ces 4 lignes, il y en a 2 que vous allez devoir changer :

- AuthName : c'est le texte qui invitera l'utilisateur à inscrire son login / mot de passe. Vous pouvez personnaliser ce texte comme bon vous semble.
- AuthUserFile : là c'est plus délicat, c'est le chemin absolu vers le fichier .htpasswd (que vous mettrez dans le même répertoire que le .htaccess).

Mais **comment trouver ce chemin absolu ?**

En effet, c'est la plupart du temps délicat à trouver. Heureusement, il existe une fonction PHP qui va beaucoup nous aider : realpath.
Cette fonction donne le chemin absolu vers le fichier que vous indiquez. Vous allez donc faire comme ceci pour trouver le chemin absolu :

1. Créez un fichier appelé "chemin.php".
2. Mettez juste cette ligne de code dedans :
   <? echo realpath('chemin.php'); ?>
3. Envoyez ce fichier sur votre serveur avec votre logiciel FTP. Placez-le dans le dossier que vous voulez protéger.
4. Ouvrez votre navigateur et allez voir ce fichier PHP. Il vous donne le chemin absolu, par exemple dans mon cas :
   /home/sdz/www/gestion/admin/chemin.php
5. Copiez ce chemin dans votre .htaccess, et remplacez le "chemin.php" par ".htpasswd", ce qui nous donne au final par exemple :
   /home/sdz/www/gestion/admin/.htpasswd
6. Supprimez le fichier "chemin.php" de votre serveur, il ne nous sert plus à rien maintenant qu'il nous a donné le chemin absolu :)

La ligne AuthUserFile indique donc où se trouve le fichier .htpasswd qui contient les mots de passe.

Enregistrez le fichier avec le nom "htaccess.txt" pour le moment, on le renommera en ".htaccess" plus tard.
Voilà, on a fini de créer le .htaccess, on peut maintenant passer au .htpasswd

---

Créer le .htpasswd

Créez maintenant un nouveau fichier avec Bloc-Notes.

Le .htpasswd contient la liste des personnes autorisées à accéder aux pages du dossier.
On met une personne par ligne, sous cette forme :
login:mot_de_passe_crypté

Au final, votre fichier .htpasswd devrait ressembler à ceci :

Code : Apache
mateo21:1MEqT//cb$hAVid.qmmSGFW/wDlIfQ81
darkeden:1/lgP8dYa$sQNXcCP47KhP1sneRIZoO0
IAN:1lT7nqnsg$cVtoPfe0IgrjES7Ushmoy.
Leon:1h4oVHp3O$X7Ejpn.uuOhJRkT3qnw3i0

Dans cet exemple, il y a 4 personnes autorisées à accéder au dossier : ce sont mateo21, darkeden, IAN, et Leon.

S'il n'y a qu'une personne autorisée à accéder au dossier, vous n'avez qu'à mettre qu'une ligne. Mais si vous êtes plusieurs admins, il est très pratique de pouvoir créer plusieurs "comptes" avec login / mot de passe

Comment crypter les mots de passe ?
Bonne question
Encore une fois, il y a une super fonction PHP qui va nous tirer d'affaire : crypt. Vous lui donnez un mot de passe et, ne cherchez pas à savoir comment, ça vous le crypte

Par exemple, si mon mot de passe est "kangourou", voici le code PHP que je devrai écrire pour l'obtenir en version cryptée :
<?php echo crypt('kangourou'); ?>

Crypter ses mots de passe est très utile : en effet, si quelqu'un vient un jour à lire votre fichier .htpasswd (quelqu'un qui utilise le même PC que vous par exemple), il ne verra que le mot de passe crypté.
Et là, aucun risque qu'il ne retrouve votre mot de passe : ce cryptage est indéchiffrable. C'est donc très pratique

Un petit script utile.

Code : PHP

```
<p>
<?php
if (isset($_POST['login']) AND isset($_POST['pass']))
{
    $login = $_POST['login'];
    $pass_crypte = crypt($_POST['pass']); // On crypte le mot de passe

    echo 'Ligne à copier dans le .htpasswd :<br />' . $login . ':' .
$pass_crypte;
}

else // On n'a pas encore rempli le formulaire

{
?>
</p>

<p>Entrez votre login et votre mot de passe pour le crypter.</p>

<form method="post">
    <p>
        Login : <input type="text" name="login"><br />
        Mot de passe : <input type="text" name="pass"><br /><br />

        <input type="submit" value="Crypter !">
    </p>
</form>

<?php
}
?>
```

Il y a 2 parties dans ce code,

1. SI les variables $_POST['login'] et $_POST['pass'] existent, alors c'est qu'on vient de valider le formulaire.
   On crypte le mot de passe qu'on a rentré, et on affiche $login: $pass_crypte pour que vous n'ayez plus qu'à copier la ligne dans le .htpasswd ^^
2. SINON, si les variables $_POST['login'] et $_POST['pass'] n'existent pas, donc on affiche le formulaire pour demander d'entrer un login et un mot de passe.
   Le formulaire recharge la même page, car il n'y a pas d'attribut action dans la balise <form> comme on l'a vu dans le chapitre sur les formulaires. Lors du rechargement de la page, les variables $_POST['login'] et $_POST['pass'] existeront puisque vous venez d'entrer le login et le mot de passe. Le mot de passe sera alors crypté !

Créez cette page quelque part sur votre disque dur (ou sur votre serveur peu importe), pour que vous puissiez crypter rapidement vos mots de passe pour le .htpasswd.
Pour un essai voir le site du zero

http://www.siteduzero.com/uploads/fr/ftp/mateo21/exhtpasswd.php

Envoyer les fichiers sur le serveur

Vous avez maintenant 2 fichiers sur votre disque dur : htaccess.txt et htpasswd.txt.

Lancez votre logiciel FTP.
Transférez les fichiers htaccess.txt et htpasswd.txt dans le dossier que vous voulez protéger par mot de passe.
Maintenant que ces fichiers sont sur le serveur, renommez-les (Bouton droit / "Renommer" ça doit marcher). Appelez-les respectivement ".htaccess" et ".htpasswd".

Fichier Favicon.ico

En fonction du navigateur que vous utilisez, vous avez peut-être déjà remarqué dans la barre d'adresse, juste à côté de l'URL, une petite image. Cette image est appelée "favicon.ico".
Avec firefox, par exemple, on retrouve cette icône. On la retrouve aussi normalement avec IE
Cette icône est souvent utilisée pour les favoris, bookmark et autres raccourcis pour Internet.

En standard, elle doit être avec les 16 couleurs standard des OS, et d'une dimension de 16x16 pixels.
Pour créer cette image, vous pouvez par exemple la créer dans n'importe quel éditeur d'image au format bmp, puis ensuite la convertir

Les sites virtuels

La dernière ligne du fichier de configuration, **# Include the virtual host :**

`Include /etc/apache2/sites-enabled/[^.#]*`

permet d'inclure le seul fichier ici présent

/etc/apache2/sites-available/default.

Ce fichier décrit la configuration de base du site web.

La place du répertoire d'accueil du site est précisée par le chemin qui suit la directive

DocumentRoot /var/www/

```
NameVirtualHost *
<VirtualHost *>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
                # This directive allows us to have apache2's default start page
                # in /apache2-default/, but still have / go to the right place
                # Commented out for Ubuntu
                #RedirectMatch ^/$ /apache2-default/
        </Directory>

        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
        <Directory "/usr/lib/cgi-bin">
                AllowOverride None
                Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog /var/log/apache2/error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog /var/log/apache2/access.log combined
        ServerSignature On

    Alias /doc/ "/usr/share/doc/"
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>

</VirtualHost>
```

Monter un Virtual Host avec Apache 2

Par Fabrice le lundi 2 avril 2007, 20:47

Comment faire pour afficher 2 contenus différents avec un seul serveur ?
Le *Virtual Host* est fait pour vous dans ce cas.

Avec HTTP 1.1, le navigateur va **ajouter une en-tête `Host`** permettant au serveur de séléctionner le contenu à renvoyer.

Exemple pris avec un serveur Apache2.2 sur une Debian Etch, ces instructions sont aussi valables sur d'autres distributions sans gands changements (voire aucun).

Les *Virtual Host* sont définis dans le répertoire `/etc/apache2/sites-available/` . Il y a par convention, 1 fichier par *Vhost*.
L'activation du Vhost se fait en faisant un lien symbolique vers la définition du *Vhost* dans le répertoire `/etc/apache2/sites-enabled/`.

Normalement, il y a un *Vhost* nommé * et sans aucune information sur le nom du serveur. Il s'agit du *Vhost* par défaut quand Apache n'a pas reçu d'en-tête `Host:` ou que le nom d'hôte ne correspond à aucun *Vhost* connu.

Voici un exemple minimum de fichier définissant un *Vhost* :

```
<VirtualHost *>
        ServerName vhost.domain.com # nom du Vhost
        DocumentRoot /home/www/vhost.domain.com/ # emplacement des fichiers

        <Directory /home/www/partage.fabroce.net/> # options d'accès
                Options Indexes FollowSymLinks MultiViews
# index automatiques, on suit les liens symboliques et on gère la négociation de
contenus
                AllowOverride None # on ne tient pas compte des .htaccess
                Order allow,deny
                allow from all # autorisation d'accès à tout le monde
                deny from .fabroce.net
        </Directory>
</VirtualHost>
```

Un reload d'Apache avec par exemple un `/etc/init.d/apache2 reload` suffit à prendre en compte le nouveau *Vhost*.

Les Virtual Host avec Apache2 quand on n'a qu'une seule adresse ip ...

Juste un pense bête

Exemple de configuration d'apache:

```
NameVirtualHost *
<VirtualHost *>

ServerAdmin webmaster@thenico.fr.eu.org
DocumentRoot /var/www/
<Directory />
Options FollowSymLinks
AllowOverride None
</Directory>
<Directory /var/www/>
Options Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all
</Directory>
ErrorLog /var/log/apache2/error.log
LogLevel warn
CustomLog /var/log/apache2/access.log combined

</VirtualHost>
<VirtualHost *>

ServerAdmin kmehdi@thenico.fr.eu.org
ServerName lecentral.eu.org
DocumentRoot /usr/chroot/home/kmehdi/public_html
<Directory "/usr/chroot/home/kmehdi/public_html">
Options Indexes MultiViews
AllowOverride FileInfo AuthConfig Indexes
Order allow,deny
allow from all
</Directory>
ErrorLog /home/kmehdi/logs/error.log
LogLevel warn
CustomLog /home/kmehdi/logs/access.log combined

</VirtualHost>
<VirtualHost *>

ServerAdmin webmaster@thenico.fr.eu.org
ServerName xavia.thenico.fr.eu.org
DocumentRoot /var/www/
<Directory />
Options FollowSymLinks
AllowOverride None
```

```
</Directory>
<Directory /var/www/>
Options Indexes FollowSymLinks MultiViews
AllowOverride None
Order allow,deny
allow from all
</Directory>
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
<Directory "/usr/lib/cgi-bin">
AllowOverride None
Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
Order allow,deny
Allow from all
</Directory>
ErrorLog /var/log/apache2/error.log
LogLevel warn
CustomLog /var/log/apache2/access.log combined

</VirtualHost>
```

Ceci configure 2 virtual host en se basant sur le parametre *Host:* de la requette HTTP du navigateur:

1. D'abord le catch-all qui quand aucune information *Host:* n'est donné pointe sur /var/www
2. Puis lecentral.eu.org qui est
3. dans /usr/chroot/home/kmehdi/public_html
4. Et xavia.thenico.fr.eu.org qui est dans /var/www

Pour ajouter un nom vhost, il suffit de rajouter un block `<VirtualHost></VirtualHost>` **en précisant bien le `ServerName`** sinon cela ne fonctionnera pas !

Un autre exemple :

Par exemple, supposez que vous hébergez le domaine `www.domain.tld` et que vous souhaitez ajouter le serveur virtuel `www.otherdomain.tld` qui pointe sur la même adresse IP. Il vous suffit d'ajouter la configuration suivante

```
NameVirtualHost *:80

<VirtualHost *:80>
ServerName www.domain.tld
ServerAlias domain.tld *.domain.tld
DocumentRoot /www/domain
</VirtualHost>

<VirtualHost *:80>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain
</VirtualHost>
```

Site sécurisé:

```
<IfModule mod_ssl.c>
<VirtualHost *:443>
 ServerName    secure.example.org
 ServerAdmin   webmaster@example.org
 ServerSignature On

 DocumentRoot  /var/www/secure.example.org-ssl

 CustomLog     /var/log/apache2/secure.example.org-ssl-access.log combined
 ErrorLog      /var/log/apache2/secure.example.org-ssl-error.log
 LogLevel warn

 SSLEngine On
 SSLCertificateFile /etc/apache2/ssl/crt/secure.example.org.crt
 SSLCertificateKeyFile /etc/apache2/ssl/key/secure.example.org.key

 <Location />
   # Force usage of ssl encryption
   SSLRequireSSL

   # SSL client certs:  none, optional, require
   # Note: optional doesn't work with all browsers
   SSLVerifyClient optional
   SSLVerifyDepth 1

   SSLOptions +StdEnvVars +StrictRequire
   #optional +ExportCertData
 </Location>


 <IfModule mod_rewrite.c>
  RewriteEngine On

  # use RewriteLog to debug problems with your rewrite rules
  # disable it after you found the error our your harddisk will be filled *very
fast*
  # RewriteLog "/var/log/apache2/rewrite_log"
  # RewriteLogLevel 2

  # The following rules will rewrite any access to
https://secure.example.org/zope/example_instance/
  # to the root of the zope instance running at localhost:10080

  RewriteRule ^/zope/main_instance$ \
      http://localhost:10080/VirtualHostBase/https/secure.example.org:443/Virtua
lHostRoot/_vh_zope/_vh_example_instance [L,P]
  RewriteRule ^/zope/main_instance/(.*) \
      http://localhost:10080/VirtualHostBase/https/secure.example.org:443/Virtua
lHostRoot/_vh_zope/_vh_example_instance/$1 [L,P]
 </IfModule>

 <IfModule mod_proxy.c>
  ProxyVia On

  # prevent the webserver from beeing used as proxy
  <LocationMatch "^[^/]">
    Deny from all
  </LocationMatch>
 </IfModule>

 # don't try to cache ssl!
```

```
 # compression (disabled)
 <IfModule mod_deflate.c>
   #SetOutputFilter DEFLATE
 </IfModule>
</VirtualHost>
</IfModule>
```