Einführung in den Apache Webserver*

Walter Neu
email: w.neu@web.de

letzte Aktualisierung: 27. Mai 2003

*Weitergehende Informationen über Apache sind auf der Homepage der Apache Software Foundation unter `http://www.apache.org` zu finden.

Inhaltsverzeichnis

1 Wie alles anfing...

1.1 Internet

In den 60-er Jahren entstand das ARPAnet[1], ein Projekt des US Regierung, um in Kriesenzeiten auf ein ausfallsicheres Datennetz zurückgreifen zu können. In den 70er und 80er Jahren hat es sich vor allem im akademischen Bereich verbreitet. Anfang der 90er Jahre entdeckte dann auch die kommerzielle Welt das Internet und fing an es für ihre Zwecke zu nutzen. Verantwortlich hierfür war in erster Linie das World Wide Web, das seitdem zu einem der wichtigsten Informations- und Werbemedien geworden ist .

1.2 World Wide Web

Der Ursprung des World Wide Web lag in Europa, genauer gesagt am europäischen Kernforschungsinstitut CERN in Genf. Es wurde nach einer Lösung gesucht, wissenschaftliche Dokumente online sichtbar zu machen, wobei einfache Textformatierung und das Einbinden von Grafik möglich sein sollte. Ganz entscheidend war aber auch die Idee, Hypertextfunktionalität einzubauen, so dass Dokumente Verweise auf beliebige andere Dokumente enthalten können, auch, wenn diese auf ganz anderen Internet-Servern liegen.
Die beiden Säulen des Projekts sollten das neue Dateiformat HTML (***H**yper**T**ext **M**arkup **L**anguage)* und das neue Internet-Protokoll HTTP (***H**yper**T**ext **T**ransfer **P**rotocol*) bilden. Neue Endanwender-Software (Browser) sollte die Dateien online anzeigen und Verweise ausführen können. Wegen des Hypertext-Charakters wurde das ganze Projekt World Wide Web (weltweites Netz) getauft.

Gleichzeitig wurde begonnen, WWW-Server einzurichten, die das neue HTTP-Protokoll unterstützten. So galt der ebenfalls am CERN entwickelte Webserver lange Zeit als Referenzimplementierung. Ferner wurden erste Client-Programme für Endanwender entwickelt. Software-Entwickler wurden von der Idee des WWW angesteckt und entwickelten fieberhaft die ersten WWW-Browser.

1.3 Der erste grafische Webbrowser

Die zuerst entwickelten Browser waren nur in der Lage, textuelle Informationen darzustellen. Ende 1993 wurde der erste WWW-Browser für grafische Benutzeroberflächen entwickelt, der Browser Mosaic. Die Programmierer von Mosaic erfanden auch neue Features, die sie in ihren WWW-Browser implementierten, ohne dass diese Features standardisiert waren. Doch andererseits wurde hauptsächlich dadurch jene Lawine ins Rollen gebracht, die wir heute erleben. Marc Andreessen, der den Boom frühzeitig witterte, stieg schliesslich aus dem Mosaic-Projekt aus und wurde Mitbegründer einer neuen Firma für WWW-Software: die Netscape Communications Corporation (`http://www.netscape.com`).

[1] Defence Advanced Research Projects Agency

In der zweiten Jahreshälfte 1993 brachten einflussreiche Blätter wie die New York Times erste Artikel über das neue Fieber in der Internet-Gemeinde. Die Anzahl der Schaulustigen wuchs, ebenso wie die Anzahl von Server-Betreibern im Internet, die sich die frei verfügbare HTTP-Software installierten und damit WWW-fähig wurden. Seit dieser Zeit verbreitet sich das World Wide Web rasend schnell weiter und ist wohl der bekannteste Dienst im Internet geworden. Schätzungen zufolge gibt es im Internet derzeit mehr als zehn Millionen Webserver.

1.4 Die Enstehung des Apache

Ebenso wie der Browser Mosaic lange Zeit der Standardbrowser war, war der am CERN entwickelte Webserver wie bereits erwähnt lange Zeit der Standardserver. Anfang 1995 war jedoch der von Rob McCool entwickelte NCSA (***N**ational **C**enter for **S**upercomputing **A**pplications*) Webserver der meistgenutzte Webserver im Internet. Nachdem jedoch Rob McCool das NCSA verlassen hatte, kam die Entwicklung des NCSA Webservers ins Stocken. Einige Leute begannen den Webserver in eigener Regie zu erweitern und zu verbessern. Es wurden immer mehr Patches hinzugefügt, und letzlich erhielt man einen *gepatchten* Server (*a patchy Server*), woraus dann der Name *Apache* enstand. Kurz nach seiner Veröffentlichung verdrängte der Apache den NCSA Server als Nummer Eins und ist mit Abstand zum meistgenutzten Webserver geworden und läuft mittlerweile sowohl unter Windows NT und UNIX wie auch auf vielen anderen Plattformen. Als eines der meistverbreitesten Open Source Software-Produkte ist er die Basis für mehr als die Hälfte aller Websites weltweit., Tendenz steigend. Der Apache liegt somit weit vor den Konkurrenzprodukten von Microsoft und Netscape.

> *»Der Apache ist einer der populärsten Web-Server im Internet seit dem April 1996. Nach einer Studie der Netcraft Web Server Survey werden 60% aller Sites im Internet vom Apache bedient. Damit ist der Apache der am meist verwendete Webserver überhaupt.«*[2]

Verantwortlich für den Erfolg war nicht zuletzt das modulare Konzept des Apache. Eine Vielzahl von Modulen ist schon in der Apache Distribution enthalten, und viele weitere Module sind für die unterschiedlichsten Anwendungsgebiete zu bekommen. Auch sind keine größere Sicherheitsprobleme mit einem Einsatz des Apache bekannt, so dass der Einsatz in produktiven Systemen ohne große Gewissensbisse durchaus empfohlen werden kann. Auch Performance ist für den Apachen kein Fremdwort.

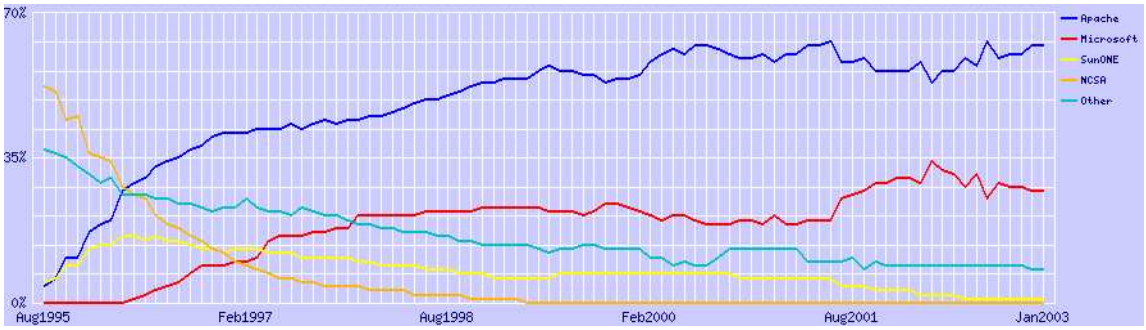

Abbildung 1.1: Marktanteil Webserver

[2]Zitat von der Homepage des Apache Projektes `http://www.apache.org`

Die Abbildung geht aus einer langjährigen Studie der Firma Netcraft (`http://www.netcraft.com/survey/`) hervor und spiegelt die Marktanteil im Webserver-Markt wider. Für die erhobenen Werte im Januar 2003 wurden 35.424.956 Websites abgefragt. Der Apache hatte damit einen Marktanteil von 62,23%. Den zweiten Platz belegt der IIS (Internet Information Server) von Microsoft mit lediglich 27,49%.

2 So arbeiten Webserver - das HyperText Transfer Protocol

Webserver und -client kommunizieren dank des *HyperText Transfer Protocol* (HTTP). Aktuell ist die Version 1.1 dieses Protokolls, man spricht von HTTP/1.1. Das zu Grunde liegende Prinzip ist simpel: Die Übermittlung der Daten erfolgt nach dem Request-Response-Schema. Der HTTP-Client sendet seine Anfrage an den HTTP-Server, der diese bearbeitet und eine Antwort zurücksendet. Im Gegensatz zum FTP-Protokoll sieht HTTP beim Verbindungsaufbau keine mehrstufige Handshake-Phase vor. Weiterer Vorteil: HTTP ist grundsätzlich abwärtskompatibel. So können zum Beispiel Browser, die HTTP/1.1 beherrschen, auch mit HTTP-1.0-kompatiblen Webservern kommunizieren. Die jeweils höhere Versionsnummer muss sich bei diesem Vorgang automatisch an die ältere Version anpassen.

2.1 Geschichte

Die ursprüngliche Version HTTP 0.9 war als einfaches Transportprotokoll für Daten ausgelegt. Sie unterstützte lediglich die GET-Methode. MIME-Unterstützung für die Übertragung von Binärdaten oder ein Authentifizierungsmechanismus waren nicht vorgesehen.

Mit HTTP/1.0 (RFC 1945) wurde ein erweitertes Request-Response-Format eingeführt, das die Übermittlung von mehr Informationen in beide Richtungen erlaubt. Dem eigentlichen Request folgt ein Satz von Headerfeldern, die beispielsweise die Übermittlung des Browsertyps erlauben. Seit Version 1.0 kennt HTTP auch die POST-Methode, mit der zum Beispiel Formulareingaben vom Browser zum Webserver gelangen.

Weitreichendere Verbesserungen brachte die Einführung von HTTP/1.1. Eine der interessantesten Neuerungen: persistente Verbindungen. Während bei HTTP/1.0 jeder Request über eine neue Verbindung zum Server übertragen wird, können in der aktuellen Version Verbindungen aufrechterhalten werden, um mehrere Requests zu übermitteln. Über die neue Funktion *Request Pipelining* sendet ein Client nachfolgende Requests an den Server, bevor die Antwort des Servers auf die vorhergehende Anforderung auf der Clientseite eingetroffen ist. Im Zusammenspiel mit persistenten Verbindungen bringt das einen deutlichen Performancegewinn. Ebenfalls für mehr Performance sorgen die in HTTP/1.1 hinzugekommenen Cache-Kontrollfunktionen. HTTP/1.1 ist in RFC 2616 definiert.

2.2 Ablauf einer HTTP-Verbindung

Die Kommunikation zwischen Client und Webserver erfolgt durch den Austausch von HTTP-Nachrichten. Dazu stellt der Client Rechner (z.B. über den Browser) zunächst im Standardfall eine TCP-Verbindung auf Port 80 zum Webserver her. Anschließend schickt der Client eine Anfrage (sog. *Request*) an den Server indem er die Informationen spezifiziert, die er einsehen möchte. Der Request enthält u.a. die Art

der Anfrage (Methode, s.u.), den URL (*Universal Resource Locater*) und die Protokollversion. Die Art der Anfrage kann, abhängig von der Protokollversion, GET, POST, HEAD, PUT, DEL, TRACE sein, die Protokollversion HTTP/1.0 oder HTTP/1.1. Der Request wird vom entfernten Webserver registriert. Der Server durchsucht daraufhin sein Dateisystem nach der geforderten Datei und lädt sie. Der Server schickt ein Antwort (sog. *Response*), die, falls die Aktion erfolgreich verlaufen ist, die gewünschten Informationen und den MIME Typ der angeforderten Datei enthält. Andernfalls besteht für den Server die Möglichkeit, anstatt von Daten eine Fehlermeldung zu übergeben. Aus dem MIME Typ schließt der Client, was er mit den empfangen Daten anfangen soll, z.B Anzeigen im Browser.

Die Verbindung wird direkt nach dem Senden des Response beendet. So soll erreicht werden, daß die Leitungskapazitäten geschont werden. Die meisten Implementierungen sehen vor, daß der Client die Verbindung zum Server aufbaut und der Server diese, nachdem er eine Response geschickt hat, schließt. Allerdings ist dies nicht ein durch das Protokoll definiertes Vorgehen. Sowohl Client als auch Server müssen in der Lage sein, Fälle zu behandeln, in denen die Verbindung während des Datenaustausch, z.B. bei einer Fehlfunktion einer Software oder durch den Benutzer, abgebrochen wird. Der Request ist damit, egal in welcher Phase er sich gerade befindet, ebenfalls beendet. Da die Verbindung stets geschlossen wird, nachdem auf einen Request geantwortet wurde, bezeichnet man HTTP als ein *zustandsloses Protokoll*. Das Gegenteil, ein *zustandsorientiertes Protokoll*, wie z.B. FTP, unterhält weiterhin eine Verbindung bis z.B. der Client diese ausdrücklich beendet.

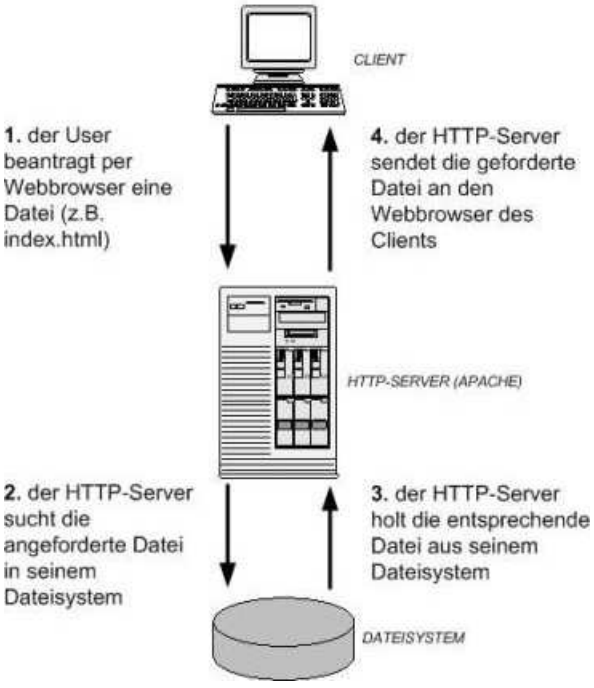

Abbildung 2.1: HTTP (stark vereinfacht)

Zu bemerken ist, dass es sich in Abbildung 2.1 bei einem Webserver nicht um Hardware handelt, wie vielleicht beim Betrachten der Abbildung der Eindruck entstehen kann. Vielmehr ist ein Webserver Software, die auf einem vernetzten Computer installiert ist und dort Anfragen entgegennimmt und entsprechend reagiert.

2.2.1 HTTP-Requests

Eine Request ist durch die Angabe von

- Methode
- URL
- und den Request-Header-Feldern

bestimmt. Dabei kommen beispielsweise die Methoden GET, PUT oder DELETE zum Einsatz. Ein Server antwortet auf jeden Request mit Informationen, ob die gewählte Methode zulässig ist oder nicht.

Eine Methode für sich genommen ist ohne Angabe des Ziels wertlos. Daher gehört zu jeder Methode ein Ziel-URL. Der Client muss dabei einen absoluten URL angeben, damit die Request auch über einen Proxyserver laufen kann. Nach dem Zugriff auf eine Quelle reicht die Angabe von relativen URLs.

Requests-Header weisen folgende Struktur auf:

```
METHOD URL HTTP/version
General Header
Request Headers
Entity Header (optional)
______Leerzeile_______
Request Entity (falls vorhanden)
```

Eine Request, der eine HTML-Seite anfordert, sieht beispielsweise so aus:

```
GET http://www.domain.de/verzeichnis/index.html HTTP/1.1
Date: Mon, 21 Apr 2003 12:28:21 GMT
User-Agent: Mozilla/5.0 Galeon/1.2.7 (X11; Linux i686; U;) Gecko/20030131
Accept: text/html, text/plain
Accept-Language: en-us, en;q=0.50
```

Zuerst übermittelt der Client die Methode, durch Leerzeichen getrennt folgen der URL und die HTTP-Version. Die weiteren Felder übermitteln die Uhrzeit, die Browserversion und welche MIME-Typen der Client akzeptiert. Ein Datenbereich entfällt bei diesem Nachrichtentyp.

2.2.2 HTTP-Methoden

Jeder Client Request wird wie bereits erwähnt durch die Angabe der Methode eingeleitet. Methoden bestimmen die Aktion der Anforderung. Die aktuelle HTTP-Spezifikation sieht acht Methoden vor: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE und CONNECT.

- **GET-Methode** Die mit Abstand wichtigste Methode ist GET. Sie dient zur Anforderung eines Dokuments oder einer anderen Quelle. Eine Quelle wird dabei durch den Request-URL identifiziert. Man unterscheidet zwei Typen: *conditional* GET und *partial* GET. Beim Conditional-GET-Typ ist die Anforderung von Daten an Bedingungen geknüpft. Die genauen Bedingungen sind dabei im Header-Feld "Conditional" hinterlegt. Oft gebrauchte Bedingungen sind zum Beispiel If-Modified-Since, If-Unmodified-Since oder If-Match. Mit Hilfe dieser Bedingung lässt sich die

Netzbelastung deutlich verringern, da nur noch die wirklich benötigten Daten übertragen werden. In der Praxis nutzen zum Beispiel Proxyserver diese Funktion, um die mehrfache Übertragung von Daten, die sich bereits im Cache befinden, zu verhindern.
Das gleiche Ziel verfolgt die partielle GET-Methode. Sie verwendet das Range-Header-Feld, das nur Teile der Daten überträgt, die der Client jedoch noch verarbeiten kann. Diese Technik wird für die Wiederaufnahme eines unterbrochenen Datentransfers verwendet.

- **POST-Methode** Den umgekehrten Weg nimmt die POST-Methode: Sie übermittelt in erster Linie Formulareingaben an einen Webserver. Aber auch die Kommentierung bestehender Quellen, Übermittlung von Nachrichten an Foren und Erweiterung von Online-Datenbanken sind mit POST möglich. Die an den Server übermittelten Daten sind in der Entity-Sektion enthalten. Auch die POST-Methode übermittelt einen URL. In diesem Fall dient dieser lediglich als Referenz, welche Routine auf dem Server die Bearbeitung der Daten übernimmt.

- **OPTIONS-Methode** Über diese Methode kann der Client Informationen über verfügbare Kommunikationsoptionen abrufen. So lassen sich insbesondere Beschränkungen von Quellen auf einem HTTP-Server oder auch einem Proxyserver ermitteln, ohne das eine bestimmte Aktion eingeleitet oder gar ein Datentransfer stattfindet.

- **HEAD-Methode** Diese Methode ist GET in seiner Funktionsweise sehr ähnlich. Einziger Unterschied: HEAD fordert lediglich den Header eines Dokuments oder Quelle an. Im Gegensatz zu GET übermittelt der Server aber nicht die eigentlichen Daten. HEAD eignet sich insbesondere dazu, die Größe von Quellen, Typ oder Objektattribute ausfindig zu machen. Der Server übermittelt auf eine HEAD-Anfrage die Metainformationen, die identisch mit den Informationen der GET-Request sind.

- **PUT-Methode** Dieser Typ erlaubt die Modifikation bestehender Quellen beziehungsweise Erzeugung neuer Daten auf dem Server. Im Unterschied zur POST-Methode identifiziert der URL im PUT-Request die mit der Anforderung gesendeten Daten selbst, und nicht die Quelle.

- **DELETE-Methode** Mit Hilfe dieses Typs werden Daten auf dem HTTP-Server gelöscht, die durch den URL identifiziert sind. Das Interessante dabei: Der Löschvorgang muss nicht unmittelbar nach dem Eingang der Anforderung, sondern kann auch zu einen späteren Zeitpunkt erfolgen. Der Server soll laut Spezifikation zumindest die Annahme der Request bestätigen.

- **TRACE-Methode** Über diese Methode kann der Client Requests verfolgen, die über mehrere Knotenpunkte laufen. Dies ist insbesondere bei der Übermittlung der Request über einen oder mehrere Proxyserver interessant. Das letzte Glied der Kette generiert die Antwort. Die TRACE-Methode dient in erster Linie der Diagnose von Client-Server-Verbindungen. Über das Max-Forwards-Header-Feld bestimmt der Client die maximale Anzahl an Hops. Im Header-Feld "Via" der Antwortnachricht sind alle durchlaufenen Server protokolliert.

- **CONNECT-Methode** Die CONNECT-Methode ist in der HTTP/1.1-Spezifikation für Verbindungen reserviert, bei denen Proxyserver dynamisch als Tunnel agieren. In der Praxis kann es sich beispielsweise um SSL-Tunnel handeln. Der Tunnelmechanismus ist in der ersten Linie als Durchgang für SSL-gesicherte Verbindungen durch eine Firewall gedacht. In Proxyserver wie MS Proxy, Netscape Proxyserver oder auch WinGate ist diese Methode bereits implementiert. Der Client bestimmt durch die CONNECT-Methode samt Portangabe den Zielrechner. Der Proxyserver baut

dann einen Tunnel zum angegebenen Rechner auf, und übermittelt Daten und Kommandos zwischen Client und Server. Details zu diesem Verfahren sind im Internet Draft "Tunneling TCP based protocols through Web proxy servers" festgelegt.

2.2.3 HTTP-Response

Die Struktur aus Header und Nachrichten-Body ist bei Request und Response gleich, auch wenn beide unterschiedliche Informationen enthalten. Der Aufbau einer HTTP-Response ist daher ähnlich zum Request:

```
HTTP/version Status-Code Reason-Zeile
General Header
Response Header
Entity Header (optional)
______Leerzeile_______
Resource Entity (falls vorhanden)
```

Ein kompletter Response, der eine HTML-Datei vom Server übermittelt, sieht beispielsweise so aus:

```
HTTP/1.1 200 OK
Date: Mon, 21 Apr 2003 12:28:21 GMT
Via: HTTP/1.1 proxy_server_name
Server: Apache/1.3.27
Content-type: text/html, text, plain
Content-length: 78

<html>
<head>
<title>HTTP</TITLE>

</head>
<body>
<p> HTTP/1.1-Demo</p>
</body>
</html> >
```

Der Server übermittelt zunächst die HTTP-Version der Nachricht. Der zweite Eintrag ist die Statusmeldung. "200 OK" bedeutet in diesem Fall, dass kein Fehler aufgetreten ist. Wichtig für die weitere Bearbeitung durch den Client sind die Einträge Content Type und Content Length. Content Type beschreibt den MIME-Typ der im Datenbereich übermittelten Datei. Im Header-Feld Content Length gibt der Server die Länge der Daten in Byte an. Der Einsatz des Feldes ist dabei nicht zwingend vorgeschrieben. Sollte das Feld fehlen, ist die Ermittlung der Datenlänge vom Typ der gesendeten Daten abhängig.

2.2.4 Response-Codes

Die Antwort des HTTP-Servers beinhaltet die Statuszeile und Response-Header-Felder. Die Statuszeile wiederum führt die Protokollversion, den Status Code und Reasons Phrase auf. Beim Status Code

handelt es sich um einen dreistelligen Integer-Wert, der dem Client wichtige Informationen über Verfügbarkeit, erfolgreiche Bearbeitung oder aber auch Fehlermeldungen liefert. Die Reasons Phrase enthält die Klartext-Bezeichnung der Meldung. Bekannte Fehlermeldungen sind beispielsweise 404 für "File not Found" (Datei nicht gefunden) oder 403 für "Forbidden" (Zugriff verweigert).

Diese Meldungen sind in fünf Kategorien eingeteilt:

- 1xx: Informelle Meldungen: Request erhalten, Bearbeitung wird durchgeführt.
- 2xx: Erfolg: Request wurde erfolgreich erhalten, verstanden und angenommen.
- 3xx: Weiterleiten: Weitere Aktionen müssen eingeleitet werden, damit eine Request vollständig bearbeitet werden kann.
- 4xx: Clientfehler: Die Request enthält ungültigen Syntax oder kann nicht bearbeitet werden.
- 5xx: Serverfehler: Der Server kann eine gültige Request nicht bearbeiten.

3 Apache 2.0 und 1.3

Zum Zeitpunkt der Erstellung des Skriptes liegt der Apache bereits in der Version 2.0 vor. Die Konfiguration von Version 2.0 ist zu einem grossen Teil abwärtskompatibel zu Version 1.3. Auf neue Funktionalitäten und Unterschiede zu Apache 1.3 wird, soweit möglich, entsprechend hingewiesen. Gelten Erläuterungen eines ganzen Unterkapitels nur für Apache 2.0 oder 1.3 wird im Titel entsprechend darauf hingewiesen. Ansonsten finden sich im Text entsprechende Hinweise (1.3) oder (2.0) wenn eine bestimmte Funktion nur für eine der beiden Versionen gilt.

4 Installation des Apache

4.1 Verfügbarkeit der Apache Distribution

Es gibt drei verschiedene Möglichkeiten den Apache zu installieren. Es gibt die Installation als

- RPM (RedHat Packet Manager)
- als fertig kompilierte Binary-Dateien
- oder als Quellcode.

Die Binary-Dateien repräsentieren eine vorkonfigurierte Standardversion, bei der die Gefahr entfällt, Fehler beim Kompilieren zu begehen; jedoch haben sie den Nachteil, dass sie nicht anpassbar sind. Ähnlich verhält es sich mit den durch die LINUX-Distributoren vertriebenen RPMs. Diese sind jedoch leicht zu aktualisieren und werden von den Distributoren gepflegt. Optimale Anpassbarkeit bietet eine Installation über den vom Anwender für die jeweilige Umgebung zu kompilierenden Quellcode. Jedoch ist dies die Variante, die das meiste Know-How erfordert.

Die aktuellste Version der Apache Distribution findet sich immer auf dem Webserver der Apache Software Foundation `http://www.apache.org` oder einem der zahlreichen Mirror-Server.

5 Konfigurationsdateien

In der Regel finden sich die Serverkonfiguration des Apache im Verzeichnis `/etc/httpd/conf`, wobei alle Konfigurationsanweisungen in der Datei `httpd.conf` gespeichert sind. In älteren Versionen des Apache wurden die Anweisungen auf drei Dateien aufgeteilt, und zwar zusätzlich zur `httpd.conf` auf die Access-Konfiguration (`access.conf`) und die Ressource-Konfiguration (`srm.conf`). In aktuellen Apache Versionen sind diese Dateien jedoch leer (1.3) bzw. gar nicht mehr vorhanden (2.0). Bei Version 1.3 können sie bei Bedarf jedoch weiterhin verwendet werden.

Beim Aufruf des Apache Daemons `httpd` kann die Datei mit der Serverkonfiguration über die Option `-f` spezifiziert werden. Wenn keine Angabe erfolgt, wird die Datei `/etc/httpd/conf/httpd.conf` geladen, sofern sie vorhanden ist.

Die Access- und Ressource-Konfigurationsdateien werden in der Serverkonfiguration des Apache 1.3 über die Anweisungen `AccessConfig` und `ResourceConfig` angegeben, wobei beim Start des Apache 1.3 zuerst die Resource- und dann die Access-Konfiguration eingelesen werden würde.

5.1 Mehrzeilige Konfigurationsanweisungen

Ab Version 1.3 des Apache wird die unter Unix-Systemen übliche Fortsetzung von Zeilen mittels Backslash unterstützt, d.h. lange Konfigurationseinträge können auf mehrere zeilen aufgeteilt werden, um sie übersichtlicher darzustellen:

```
AddIconByType (BIN,/icons/binary.gif) \
              application/octet-stream
```

5.2 Bedingte Konfigurationsanweisungen

Bedingte Abschnitte umfassen Anweisungen, die bei der Abarbeitung der Konfigurationsdateien ignoriert werden, falls die Bedingung nicht erfüllt ist. Bedingte Abschnitte dürfen ineinander geschachtelt werden.

5.2.1 <IfDefine>

Syntax	`<IfDefine [!]`*`Parameter`*`> ... </IfDefine>`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>, <Directory>, <Location>, <Files>, .htaccess

Mit Parameter sind hierbei die Parameter gemeint, die beim Starten des Apache auf der Kommandozeile mit der Befehlszeilen Option `-D` spezifiziert wurde oder wenn er nicht definiert wurde und dem

Parameter ein Ausrufezeichen (!) vorangestellt wird.

Beispiel:

```
<IfDefine !STAGING>
  DocumentRoot /usr/local/apache/htdocs
</IfDefine>

<IfDefine STAGING>
  DocumentRoot /usr/local/apache/devdocs
  <Location />
    Order deny,allow
    deny from all
    allow from 123.45.
  </Location>
</IfDefine>
```

Wird nun der Apache mit `-DSTAGING` gestartet, so werden nur die Konfigurationsanweisungen der zweiten `<IfDefine>` Sektion ausgeführt, d.h. die `DocumentRoot` wird auf `/usr/local/apache/devdocs` gesetzt und der Zugriff nur für bestimmte Rechner erlaubt.

Es können nur die mittels `-D` angegebenen Parameter bei `<IfDefine>` verwendet werden, wobei Groß- und Kleinschreibung beachtet werden muss.

5.2.2 <IfModule>

Syntax	`<IfModule [!]`*`Modulname`*`> ... </IfModule>`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>, <Directory>, <Location>, <Files>, .htaccess

Die eingeschlossene Anweisung wird nur ausgeführt, wenn das angegebene Modul in den Apache eingebunden wurde. Ist dem Modul ein Ausrufezeichen (!) vorangestellt, werden die eingeschlossenen Anweisungen ausgeführt, wenn das Modul nicht aktiv ist.

Als Modulname wird der Source-Dateiname des Modules angegeben, zum Beispiel `mod_env.c` oder `mod_negotiation.c`.

Beispiel:

```
<IfModule mod_env.c>
  SetEnv TEST 0815
</IfModule>
```

Die `SetEnv` Anweisung in obigem Beispiel wird nur ausgeführt, wenn das Modul `mod_env` auch eingebunden wurde. Jetzt stellt sich allerdings die Frage, warum obige Anweisung überhaupt sinnvoll ist, da die Variable `TEST` sowieso nicht gestzt werden kann, wenn das Modul `mod_env` nicht zur Verfügung steht, weil der Apache dann die Anweisung `SetEnv` gar nicht kennt? Der Punkt ist jedoch der, dass der Apache nicht startet, wenn die obige Anweisung in der Konfiguration auftaucht, aber das Modul `mod_env` nicht

in den Apache eingebunden wurde. Ein gutes Beispiel ist die Default-Konfiguration des Apache: einige der Konfigurationsanweisungen sind in entsprechende `<IfModule>`-Sektionen eingebunden um sicherzustellen, dass der Apache auch dann startet, wenn bestimmte Module nicht in den Apache eingebunden wurden.

Stellt man dem Modulnamen ein Ausrufezeichen (!) voran, negiert man die Anweisung, d.h. die enthaltenen Anweisungen werden nur dann ausgeführt, wenn das jeweilige Modul nicht im Apache eingebunden ist.

Beispiel:

```
<IfModule !mod_negotiation.c>
  <FilesMatch "\.var$">
    deny from all
  </Files>
</IfModule>
```

Wenn das Modul `mod_negotiation` nicht zur Verfügung steht, wird der Zugriff auf alle `.var`-Dateien (Typ-Maps) gesperrtt, damit diese nicht im Klartext heruntergeladen werden können.

Durch die Verwendung von vorangestellten Ausrufezeichen (!) lassen sich *If-Then-Else* ähnliche Konstrukte erstellen.

6 Modulkonfiguration

6.1 Dynamic Shared Objects

Der Webserver Apache ist heute so groß geworden und kennt so viele spezielle Features, die nicht immer verwendet werden sollen, daß er nicht mehr *monolithisch* in einem Block aufgebaut ist, sondern *modular*. Der eigentliche Kern-Server (core-server) kennt nur die grundlegenden Eigenschaften eines Webservers. Alle zusätzlichen Fähigkeiten, wie Authentifizierung, GCI-Scripts, PHP-Scripts, Session-Management und vieles mehr, werden mit Hilfe von Modulen eingebaut.

Durch die DSO-Funktionalität ist es möglich, Module dynamisch in den Apache einzubinden oder auch wieder aus ihm zu entfernen. Der Apache muss hierzu nicht neu übersetzt werden, aber das Modul `mod_so`, dass die DSO-Funktionalität zur Verfügung stellt, muss statisch im Apache eingebunden sein. Zudem müssen die Module als sog. DSO-Dateien vorliegen. Gebräuchlich sind auch die Bezeichnungen *Shared Objects* und *Shared Libraries*. Interessant wird die DSO-Funktionalität vor allem dadurch, dass die Module zur Laufzeit des Apache verändert werden können, und zwar ohne eventuell aktive Verbindungen unterbrechen zu müssen.

Die meisten Module, die für den Apache erhältlich sind, werden entweder über eine spezielle Konfigurationsanweisung aktiviert respektive deaktiviert oder stellen nur einen Handler zur Verfügung, der erst mit `SetHandler` oder `AddHandler` aktiviert werden muss. Es gibt jedoch auch Module, die grundsätzlich aktiviert sind, wie z.B. `mod_userdir`.

Wenn der Apache statisch übersetzt werden soll, muss man sich vorher Gedanken machen, welche Module benötigt werden und welche nicht. Bei Verwendung von DSO kann jedoch auch nachträglich die Modulzusammenstellung geändert werden. Zur Konfiguration dienen hierbei die beiden Anweisungen `LoadModule` und `LoadFile`. Mit `LoadFile` können zusätzliche Objekt-Dateien oder Bibliotheken in den Apache eingebunden werden, die zur Verwendung von Zusatzmodulen benötigt werden. Für die Verwendung der Standardmodule, wie auch der meisten Zusatzmodule, werden jedoch in der Regel keine zusätzlichen `LoadFile`-Anweisungen benötigt.

6.1.1 LoadModule

Syntax	`LoadModule` *`Modulname Dateiname`*
Modul	mod_so
Apache	1.3 und 2.0
Kontext	Serverkonfiguration

Wenn `mod_so` in den Apache eingebunden wurde, können beliebige Module über eine `LoadModule`-Anweisung in den Apache geladen werden. Bei Start repektive Neustart werden alle `LoadModule`-Anweisungen der Reihe nach durchgegangen, und es wird versucht, das jeweils angegeben Modul in den Apache einzubinden und zu aktivieren.

Als erstes Argument wird der interne Modulname erwartet und als zweites Argument der Dateiname des Moduls, der absolut oder auch relativ zum `ServerRoot`-Verzeichnis angegeben werden kann. In Apache 1.3 hat das Modul-Verzeichnis den Namen `libexec` und ab Apache 2.0 `modules`.

Beispiel:

```
LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
```

Die Abfolge der `LoadModule`-Anweisung spiegelt in der 1.3er Version die Priorität der Module in umgekehrter Reihenfolge wider, d.h. ein später geladenes Modul hat eine höhere Priorität als ein zuvor geladenes. In Apache 2.0 ist diese Form der Priorisierung nicht mehr notwendig.

6.2 Modulaktivierung (Apache 1.3)

Die Priorität der einzelnen Module, die von entscheidender Bedeutung für die korrekte Funktion des 1.3er Apache ist, sollte jedoch durch ein `ClearModuleList` gefolgt von `AddModule`-Anweisungen festgelegt werden. Wenn alle Module, mit Ausnahme von `mod_so`, mit `LoadModule` in den Apache eingebunden werden, ist die Angabe von `AddModule` zwar nicht zwingend notwendig, sofern die korrekte Priorität bereits durch die `LoadModule`-Anweisung festgelegt wurde. Die beiden Konfigurationsanweisungen werden in der Regel nur dann benötigt, wenn `mod_so` im Spiel ist, d.h. alle oder ein Teil der Module als DSO-Dateien in den Apache geladen werden. Allerdings ist die Neuordnung der Module durch entsprechende `AddModule`-Anweisungen ein Muss, wenn sowohl statische als auch dynamisch geladene Module verwendet werden.

6.2.1 ClearModuleList (Apache 1.3)

Syntax	`ClearModuleList`
Modul	(alle eingebundenen Module sind aktiv)
Apache	1.3
Kontext	Serverkonfiguration

Mit der Anweisung ClearModuleList wird die interne Liste der aktiven Module gelöscht, mit Ausnahme des Apache-Kerns (`httpd_core`) Mit »aktiv« ist hierbei gemeint, dass sie in der internen Liste für aktive Module des Apache stehen und somit bei Anfragen die Möglichkeit bekommen, diese zu bearbeiten bzw. entsprechende Funktionen auszuführen.

Nachdem `ClearModuleList` in der Serverkonfiguration ausgeführt wurde, müssen alle gewünschten Module mittels `AddModule` wieder aktiviert werden.

6.2.2 AddModule (Apache 1.3)

Syntax	`AddModule` *`Modulname`*
Modul	(alle eingebundenen Module sind aktiv)
Apache	1.3
Kontext	Serverkonfiguration

Über die Konfigurationsanweisung `AddModule` wird ein Modul aktiviert, also in die Apache interne Modulliste aufgenommen, sofern es noch nicht aktiv war, z.B. weil kurz zuvor `ClearModuleList` aufgerufen wurde.

Als Modulname wird hierbei der Dateiname des jeweiligen Moduls angegeben, so wie es in den Apache eingebunden wurde.

Beispiel:

```
ClearModuleList
AddModule mod_env.c
AddModule mod_log_config.c
...
```

Werden obige Beispielanweisungen der Serverkonfiguration hinzugefügt, so sind nach dem Start des Apache auch nur diese bei `AddModule` genannten Module und der Apache-Kern aktiv und können genutzt werden. Alle anderen Module, die in den Apache statisch oder über `LoadModule` eingebunden wurden, sowie deren Konfigurationsanweisungen stehen nicht mehr zur Verfügung.

7 Konfigurationsanweisungen

Die Konfigurationsdatei `httpd.conf` des Webservers kennt hunderte verschiedener Anweisung, die hier nicht alle dargestellt werden können. Wir werden uns hier also nur auf einige wenige der wichtigsten Anweisungen beschränken.

Die Konfigurationsdatei ist in drei Bereiche aufgeteilt:

- ❒ die globalen Einstellungen
- ❒ die Einstellungen des Hauptservers
- ❒ und die Einstellungen für die virtuellen Server.

7.1 Globale Einstellungen (Section 1: Global Environment)

Unter »Globale Einstellungen« werden die Bedingungen verstanden, unter denen der Apache Webserver arbeiten soll. Zu dieser Arbeitsumgebung gehören u.a.:

- ❒ **Angaben zum Servertyp**, d.h. der `httpd` Daemon kann einerseits durch den `inetd` gestartet werden oder als `standalone` laufen (wobei für den Einsatz auf einem Windows System ausschließlich der Typ `standalone` gewählt werden kann). Eher üblich ist allerdings die `standalone` Version. Der Nachteil des `inetd` ist offensichtlich: Bei jeder Anfrage muss der Apache neu gestartet werden, was einen Performance Verlust mit sich bringt. Auch wenn nur eine handvoll Zugriffe pro Tag erwartet werden, sollte die `standalone`-Variante den Vorzug bekommen, damit Anfragen schnell beantwortet werden können.

Syntax	`ServerType standalone | inetd`
Default	`standalone`
Apache	1.3
Kontext	Serverkonfiguration

- ❒ die **Definition des Hauptverzeichnisses** (Root Verzeichnis für den Server), in dem sich die Verzeichnisse für die Logfiles, Konfigurations- und Protokolldateien befinden. Relative Pfadangaben für andere Anweisungen, wie zum Beispiel `AccessConfig`[1], beziehen sich auf dieses Verzeichnis.

Syntax	`ServerRoot` *`Verzeichnis`*
Default	`/usr/local/apache`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration

[1] Sekundäre Konfigurationsdatei, die ursprünglich Anweisungen für die Zugriffskontrolle enthielt. Ab Apache 2.0 nicht mehr vorhanden.

- ein paar **Festlegungen für Zugriffszeiten und -zahlen** sowie

- die **Liste der zur Laufzeit einzubindenden Module**. Bei diesen Definitionen der Arbeitsumgebung gibt es noch nicht allzuviele Variationsmöglichkeiten. Wichtig ist die Liste der Module, die nicht ohne Grund zur globalen Umgebung zählen. Im Standardfall ist die gesamte Liste der zuladbaren Module erst einmal deaktiviert. Es lohnt sich aber, die Dokumentation zu den Modulen gründlich nachzulesen und dann die Module zu aktivieren, mit denen man arbeiten möchte. Auch das Einbinden individuell erstellter weiterer Module ist hier möglich.

7.2 Einstellungen des Hauptservers (Section 2: Main Server Configuration)

Nachdem die Arbeitsumgebung definiert worden ist, folgen als nächstes im Abschnitt »Main Server Configuration« die Anweisungen zur Arbeitsweise. Arbeitsweise bedeutet dabei im wesentlichen:

- **Festlegung des vom Server beanspruchten Ports** (für das http-Protokoll in der Regel Port 80). Es kann sinnvoll sein, einen anderen Port für ein lokales LAN zu wählen, um möglichen Konflikten auszuweichen. Dann sollte es einer der nicht für Standards vorgesehenen Ports unter 1024 sein[2]. Falls Apache als `standalone` laufen soll, wird eine Portnummer unbedingt benötigt, auf der der Daemon lauschen soll. Die Voreinstellung ist Port 80.
  Wenn eine abweichende Port-Nummer gewählt wird, so muß der Aufrufer dem URL die Portnummer hinzufügen, z.B. `http://www.mydomain.de:999/` bei gewählter Portnummer 999.

Syntax	`Port` *`Portnummer`*
Default	`80`
Apache	1.3
Kontext	Serverkonfiguration, <VirtualHost>

  Wenn auch für SSL (siehe Kapitel 9, Secure Webserver) gesorgt werden soll, muss der Standard-HTTP-Port ebenso genutzt werden können wie der Standard-HTTPS-Port:

```
<IfDefine SSL>
Listen 80
Listen 443
</IfDefine>
```

  Ab Apache 2.0 wird der Port in Verbindung mit der Servername Anweisung angegeben, s.u.

- **Administrator-Adresse** Mit Hilfe der Anweisung `ServerAdmin` wird die Adresse angegeben, an die eventuelle Probleme mit dem Server gemeldet werden können. Diese Adresse erscheint auf allen Dokumenten, die vom Server generiert werden, zum Beispiel Fehlermeldungen.

[2] Alle wichtigen Netzwerkdienste liegen auf bekannten Ports im unteren Bereich von 1 bis 1023. Man nennt diese Ports auch *privilegierte Ports*. Sie gehören zu Programmen, die mit elevierten Privilegien laufen, d.h. als Benutzer root. Die oberen Ports von 1024 bis 65535 heißen *unprivilegierte* Ports.

Syntax	`ServerAdmin` *`Email-Adresse`*
Default	[keine gültige Adresse]
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>

❒ **Servername** `ServerName` erlaubt es, einen Hostnamen festzulegen, über den der Webserver angesprochen werden soll, zum Beispiel kann "www" statt des realen Hostnamens festgelegt werden. Allerdings kann nicht einfach irgendeinen Hostnamen erfunden und erwartet werden, daß er funktioniert. Ein Name, der hier angegeben wird, muß ein gültiger DNS-Name sein. Wenn der Host nicht über einen DNS-Namen verfügt, kann hier auch seine IP-Adresse angegeben werden. Sie können dann auf jeden Fall über diese IP-Adresse Zugriff bekommen (z.B. http://192.168.0.1). 127.0.0.1 ist die lokale Loopback-Adresse für das TCP/IP-Protokoll, meist als "localhost" bezeichnet. Ihre Maschine müßte sich mit dieser Adresse immer selbst erkennen. Wenn Sie Apache ausschließlich dazu einsetzen, lokale Tests und Entwicklungsarbeiten durchzuführen, können Sie hier 127.0.0.1 als Servernamen eintragen.

Syntax	`ServerName` *`FQDN [:Port]`*
Default	[FQDN des Serverrechners]
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>

In Apache 2.0 wurde die `Port`- und `ServerName` Anweisung kombiniert, d.h. es gibt ab dieser Version nur noch die `ServerName` Anweisung, welche die optionale Angabe einer Portnummer erlaubt. Per Default wird entweder Port 80 (HTTP) oder Port 443 (HTTPS) verwendet, je nachdem ob für den jeweiligen (virtuellen Server) SSL aktiviert wurde oder nicht:

Beispiel (Apache 2.0):

```
ServerName new.host.name:80
```

❒ **Identität des Webusers** Der Apache Server unter UNIX muss zunächst unter der Benutzerkennung »root« gestartet werden, um an den Port 80 gebunden werden zu können. Durch Verwendung der `User`- und `Group`-Anweisung wird jedoch der Benutzer und die Gruppe festgelegt, unter dem der Apache seine Requests bearbeiten soll. Ressourcen, die von Apache zur Verfügung gestellt werden sollen, müssen von diesem Benutzer, bzw. dieser Gruppe gelesen werden können. Wird der als `standalone` konfigurierte Server vom User »root« gestartet, so wird - nachdem der Port geöffnet wurde - zu der unter `User` und `Group` angegebenen UID respektive GID gewechselt.

Syntax	`User` *`Benutzername|#Benutzer-ID`*
Default	`-1`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>

und entsprechend für die Gruppe

Syntax	`Group` *`Gruppenname|#Gruppen-ID`*
Default	`-1`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>

❒ Pfadangaben zu Dokument-, Benutzer- und Scriptverzeichnissen

– **Dokumentverzeichnis** Änlich wie `ServerRoot`, welches das Home Verzeichnis des Apache angibt, wird mit `DocumentRoot` das Verzeichnis definiert, das die HTML Dokumente und sonstige Dateien enthält, die über den Webserver abrufbar sein sollen. Es ist das Wurzelverzeichnis für alle veröffentlichen Dokumente.

Syntax	`DocumentRoot` *`Verzeichnisname`*
Default	`/usr/local/apache/htdocs`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>

Wenn der Client ein Dokument anfordert, so wird die Pfadangabe in der Anforderung (hinter dem Domainnamen) umgesetzt in eine Pfadangabe relativ zu diesem Verzeichnis
Mit der Anweisung
`DocumentRoot /web/htdocs`

würde also bei einem Client Zugriff auf
`http://www.mydomain.de/index.html`

die Datei
`/web/htdocs/index.html`

an den Client ausgeliefert, sofern sie vorhanden ist und keine `Alias`- bzw. `Redirect` Anweisungen zum Tragen kommen.

– **Benutzerverzeichnis**. Beim Stöbern auf Webservern trifft man häfig auf URLs der Form `http://wwwsomewhere.com/~user/`, bei denen der URL-Pfad mit einer Tilde »~« beginnt. Wie auch der Apache verwenden die meisten Webserver diese Art der URLs, um Benutzerverzeichnisse zu kennzeichnen. Der Name nach der Tilde ist hierbei der Account-Name des jeweiligen Benutzers. Um also den verschiedenen Usern eines Unix-Systems zu ermöglichen, auch selbst im Netz Seiten anzubieten, gibt es die Möglichkeit, innerhalb des Home-Verzeichnisses des Users ein Unterverzeichnis anzulegen, das dann auch vom Webserver verwaltet wird. Der Name dieses Verzeichnises wird in der zentralen Konfigurationsdatei des Webservers mit der Direktive `UserDir`. In der Regel heißen diese Verzeichnisse `public_html`.

Syntax	`UserDir` *`Verzeichnis`*
Default	`public_html`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>

Um diese Funktionalität zu erhalten muß jedoch das Modul `mod_userdir` geladen sein.

Wenn ein User also jetzt in seinem Homeverzeichnis ein Verzeichnis `public_html` anlegt, kann er dort beliebige Webseiten veröffentlichen, die dann unter `http://Rechnername/~Username`

abrufbar sind. Allerdings muß dafür der User, unter dessen UID der Webserver läuft (z.B. wwwrun) Lese- und Durchsuchungsrechte auf die Userverzeichnisse haben.

Typischerweise werden in der Konfigurationsdatei des Webservers für alle diese Userverzeichnisse in einem Aufwasch bestimmte Einstellungen vorgenommen, wie etwa:

```
<Directory /home/*/public_html/>
AllowOverride FileInfo AuthConfig Limit
Options MultiViews Indexes IncludesNoExec
</Directory>
```

- **Scriptverzeichnis**, siehe Kapitel 10 , Common Gateway Interface.

❒ **Verzeichnisindex**, dh. der Name (oder die Namen) der Datei, die in einem Verzeichnis aufgerufen werden soll, wenn nur der Verzeichnisname angegeben wurde. Wenn mehrere Namen angegeben werden, müssen diese durch Leerzeichen voneinander getrennt werden.

Syntax	`DirectoryIndex` *`Dateiname...`*
Default	`index.html`
Apache	1.3 und 2.0
Kontext	Serverkonfiguration, <VirtualHost>, <Directory>, <Location>, <Files>, .htaccess

❒ **Formatierung der Protokolldateien**

❒ **Aufgabenzuweisung an die geladenen Module**. Während die Festlegung des Ports, die Namensgebung (ein Servername wird zwingend verlangt, sonst startet Apache nicht) sowie die Bestimmung der Pfade von Server Verzeichnissen und die Inhaltsvorgabe für die Protokolldateien in der Regel keine Probleme darstellen, liegt die Hauptarbeit eines Administrators wahrscheinlich darin, den Modulen die gewünschten Aufgaben zur Erledigung zuzuweisen. Ob diese Zuweisungen von Apache befolgt werden können, ist davon abhängig, ob das entsprechende Modul überhaupt geladen ist. Mit `<IfModule modul_name>` wird diese Bedingung (ist das Modul geladen?) überprüft, dann folgt der entsprechende Befehl, der zur Ausführung an das Modul übergeben werden soll. Bei einer Standard-Installation betrifft das u.a. folgende Module:

- `<IfModule mod_userdir.c>` - bestimmt den Pfad und die Zugriffsmöglichkeiten für Benutzerverzeichnisse
- `<IfModule mod_dir.c>` - legt u.a. fest, welche Datei als Standard-Inhaltsdatei angesprochen wird; z. B. `index.htm`
- `<IfModule mod_mime.c>` - gibt die Liste der MIME-Typen an, die der Server bearbeiten darf. Als Standard wird hier eine externe Datei mit dem Namen `mime.types` eingebunden, es ist jedoch auch möglich, die Liste unmittelbar hier einzutragen
- `<IfModule mod_mime_magic.c>` - ähnlich dem vorher aufgerufenen Modul liest auch dieses eine externe Datei ein, in der jedoch vorrangig Formate für Sound, Grafik, komprimierte Archive (ZIP) oder Applikationen enthalten sind
- `<IfModule mod_alias.c>` - steuert die Erkennung von Aliasnamen und die Zuweisung solcher Aliase an Verzeichnisse. Die Vergabe von Aliasnamen bereitet manchmal Probleme.

- `<IfModule mod_autoindex.c>` - legt u.a. das Erscheinungsbild von servergenerierten Dokumenten (FTP-Stil) fest. Darüber hinaus wird mit Hilfe dieses Moduls bestimmt, welche Sprache der Sever als Standardsprache ansehen soll
- `<IfModule mod_mime.c>` - mit dem zweiten Aufruf können jetzt die verschiedenen dem Server inzwischen bekannten Informationen weiter aufbereitet werden. Das Modul erkennt die verschiedenen Dateitypen beigegebenen "Meta-Informationen" und übergibt sie nötigenfalls an den aufrufenden Browser. Außerdem können hier "Handler" für bestimmte Dateitypen eingesetzt werden, mit deren Hilfe dann beispielsweise serverseitig CGI-Prozesse oder ähnliches aufgerufen und gesteuert werden können.
- `<IfModule mod_setenvif.c>` - legt bestimmte Verhaltensweisen fest, falls in den Umgebungsvariablen Probleme mit inkompatibler Software o.ä. auftauchen
- `<IfModule mod_proxy.c>` - bestimmt, ob und wie die Proxyfunktion wahrgenommen werden soll

Grundsätzlich können von den Modulen abzuarbeitende Befehle auch "verschachtelt" werden, das heißt, innerhalb eines Moduls kann ein weiteres aufgerufen werden.

Zwischen diesen Definitionsblöcken der Module finden sich einzelne weitere Befehlszeilen, die zwar in der Regel auch von einem bestimmten Modul ausgeführt werden (gilt nicht für Pfadangaben), wobei dieses Modul aber nicht in einem eigenen Container explizit angesprochen werden muß. Wichtig sind zum Beispiel die bei Bedarf gleich nach der Definition von Benutzerverzeichnissen und Index-Datei vorgenommenen Festlegungen zur Verwendung einer Datei `.htaccess`. Wird die Verwendung solcher Dateien zugelassen, kann ein beliebiger Benutzer selbst darüber entscheiden, wie die Zugriffsrechte auf seine Verzeichnisse und Dateien gestaltet werden; und zuständig dafür ist das Modul `http_core.c`.

Die Schreibweise `<IfModule modulname>` mag manchen verleiten, die Konfigurationsblöcke für etwas ähnliches wie HTML-tags zu halten, vor allem, weil ein geöffneter Container auch mit `</IfModule>` wieder geschlossen werden muß. Das ist allerdings ein Irrtum. Diese Anweisungsblöcke werden *Container* genannt, da in ihnen jeweils zu einem Modul oder einer bestimmten Struktur passende Befehlszeilen enthalten sind. Neben `<IfModule modulname>` können noch weitere Container zur Anwendung kommen, zum Beispiel `<Files - "Dateiname">` für bestimmte Dateien wie `.htaccess`, oder `<Location /xxxx>` für bestimmte Speicherbereiche innerhalb des Verzeichnisbaums.

Eine Übersicht über die mehr als 200 möglichen Befehle findet sich in der mitgelieferten Hilfe. Die Datei `/var/www/manual/mod/directives.html.en` enthält eine Liste dieser Befehle (Direktiven).

7.3 Virtuelle Server (Section 3: Virtual Hosts)

Die dritte Sektion der zentralen Konfigurationsdatei von Apache dient der Definition von *virtuellen Servern*[3]. Apache war der erste Webserver, welcher die Möglichkeit von virtuellen Servern (auch virtuelle Hosts genannt) ermöglichte.

[3] Die Bezeichnung »virtuell« besagt, dass was vorgetäuscht wird, und ein »Server« ist in diesem Fall ein Rechner. Es wird also vorgetäuscht, dass jede Website auf einem eigenen, physikalischen Rechner befindet.

Wenn man davon redet, dass ein Webserver Unterstützung für virtuelle Server bietet, so meint man damit, dass er gleichzeitig unter mehr als einem Hostnamen agieren und dabei jeweils ein eigenes Dokumentverzeichnis, Logfiles, usw. verwalten kann.

Das heißt im Klartext, wenn mehrere Websites gleichzeitig betrieben werden sollen, braucht man weder mehrere Rechner noch muss man mehrere unabhängige Instanzen des Apache starten (obwohl auch das möglich ist). Normalerweise ist ein einziger Rechner durchaus leistungsfähig genug für mehrere Sites, und auch von Apache muss nur eine einzige Kopie laufen. Sowohl die Kosten als auch der Konfigurations- und Wartungsaufwand ist dabei geringer, als wenn für jede Website ein eigener Rechner verwendet werden würde. Selbst wenn zur Performancesteigerung mehrere Rechner im Einsatz sind, ist es meist sinnvoller, dass diese von mehreren virtuellen Hosts gemeinsam genutzt werden.

Abbildung 7.1: Virtuelle Hosts

Dabei gibt es zwei Arten von virtuellen Servern: zum einen dienen die normalen *IP-basierten* virtuellen Servern, die die Vergabe von einer IP-Adresse pro Server verlangen, und die mit HTTP/1.1 eingeführten *namensbasierten* viruellen Server (*Non-IP Virtual Hosts*). Als dritte von Apache unterstützte Variante gibt es noch die sogenannten port-basierten virtuellen Server (Port-based Virtual Hosts), die prinzipiell auch nur IP- oder namensbasierte Server sind, nur eben mit unterschiedlichen Portnummern.

7.3.1 IP-basierte virtuelle Hosts

Die ursprüngliche Form von virtuellen Webservern setzt voraus, dass jeder virtuelle Server eine eigene IP-Adresse bekommt (IP-basierte virtuelle Server). Diese Limitierung beruht auf einer Restriktion von HTTP/1.0, da hier ein Client nicht kenntlich machen kann, auf welche Website (Hostname) er zugreifen will. Verwendet man jedoch für jeden virtuellen Server eine eigene IP-Adresse (A-Record im Nameserver), erkennt der Webserver anhand der IP-Adresse, welches Dokument er zurückliefern soll.

Die zentrale Anweisung bei der Konfiguration von virtuellen Servern ist die `<VirtualHost>` Anweisung (siehe 7.3.2.1).

Innerhalb der `<VirtualHost>` Klammerung können also alle Direktiven stehen, die auch schon für die Konfiguration des Hauptservers zur Anwendung kamen. Es sind also vollständig autarke Server, denen sogar eigene CGI-Verzeichnisse gegeben werden können. Das folgende Beispiel zeigt zwei virtuelle Hosts, die adressenbasiert aufgebaut sind und je ein eigenes `cgi-bin` Verzeichnis besitzen:

```
<VirtualHost 10.230.1.105>
     ServerAdmin root@mydomain.de
```

```
    DocumentRoot /www1/htdocs
    ScriptAlias /cgi-bin/ "www1/cgi-bin/"
    ServerName virtual1.mydomain.de
</VirtualHost>

<VirtualHost 10.230.1.106>
    ServerAdmin hans@mydomain.de
    DocumentRoot /www2/htdocs
    ScriptAlias /cgi-bin/ "www2/cgi-bin/"
    ServerName virtual2.mydomain.de
</VirtualHost>
```

7.3.2 Namensbasierte virtuelle Hosts

Mit HTTP/1.1 wurden die sog. *Name-based* bzw. *Non-IP Virtual Hosts* eingeführt, d.h. es lassen sich virtuelle Server realisieren, ohne jeweils eine eigene IP-Adresse vergeben zu müssen. Im Nameserver muss lediglich ein Hostname-Alias (CNAME) eingetragen werden (oder aber zum Testen auf dem lokalen Rechner die entsprechenden Einträge in der `/etc/hosts` setzten).

Wieso sollte man eine IP Adresse mehrfach verwenden?
Die mehrfach Verwendung von IP Adressen ist aus verschieden Gründen sinnvoll:

1. man hat nicht genügend IP Adressen
2. man hat nur eine einzige IP Adresse
3. wird beim Betreiben von Webservern von der RIPE (`http://www.ripe.net`) vorgegeben. RIPE steht für *Réseaux IP Européens* und ist ein Zusammenschluß europäischer ISPs, der das RIPE NCC betreibt. Ziel der 1989 gegründeten Vereinigung mit Sitz in Amsterdam ist es, die organisatorischen und technischen Voraussetzungen für einen europaweiten IP-Verkehr zu schaffen.
4. Zeitersparnis des Administrators, er muß einem Rechner nicht mehrere IP Adressen zuweisen.

Wie bereits erwähnt hat allerdings die Verwendung solcher namensbasierten virtuellen Server ohne eigene IP-Adresse einen Haken, denn sie funktioniert nur, wenn auch der Webclient diesen Teil von HTTP/1.1 unterstützt. Realisiert wird diese Art der virtuellen Server dadurch, dass ein Client bei jedem Zugriff den HTTP-Header `Host` mitschickt, in dem er den Hostnamen der Website übermittelt, auf die er zugreifen will.

Beispiel-Header:

```
Host: www.asw-berufsakademie.de
```

Mit Hilfe des Mozilla Websniffers (`http://webtools.mozilla.org/web-sniffer/`) ist das sehr schön nachzuvollziehen.

Übermittelt der Client solch einen Header nicht, kann der Webserver nicht herausfinden, welcher virtuelle Host angesprochen werden soll. Er verwendet in solch einem Fall den für diese IP-Adresse konfigurierten Default-Server. Dies ist immer der erste in der Serverkonfiguration aufgeführte virtuelle Server für diese IP-Adresse.

Heutzutage sollte die Nutzung von namensbasierten virtuellen Servern jedoch keine Schwierigkeiten mehr machen, denn es werden nur noch sehr wenig alte Webclients verwendet, die noch keinen Host-Header mitliefern. So wird diese Funktionalität beim Netscape Navigator und dem Microsoft Internet Explorer bereits seit Version 3.0 unterstützt. Wenn keine anderern Gründe dagegen sprechen namensbasierte Server zu verwenden, so sollten diese gegenüber den IP-basierten Servern unbedingt bevorzugt werden.

Neben der zentrale Anweisung `<VirtualHost>` bei der Konfiguration von virtuellen Servern, kommt bei namensbasierten Servern noch die Anweisung `NameVirtualHost` und im Bedarfsfall noch die Anweisung `ServerAlias` und `ServerPath` hinzu. Somit sind nur sehr wenige Konfigurationsanweisungen im Spiel, aber dennoch ist die Konfiguration von virtuellen Servern ein oft diskutiertes Thema, bei dem zahlreiche Fehler gemacht werden. Im Folgenden werden zuerst die genannten Anweisungen erläutert und danach an einigen Beispielen verdeutlicht.

7.3.2.1 <VirtualHost>

Die Angabe der `<VirtualHost>` Anweisung ist nur in der Serverkonfiguration erlaubt. Mit ihr wird ein virtueller Server eingerichtet bzw. aktiviert.

Als Argument wird eine IP-Adresse oder ein Hostname erwartet, wobei optional eine Portnummer angegeben werden kann. Verwendet man den Stern »*« als Portnummer, so sind alle Ports gemeint, auf denen der Apache auf Anfragen lauscht.

```
<VirtualHost 192.168.251.200>
    ....
</VirtualHost>


<VirtualHost www.mydomain.de:8080>
    ....
</VirtualHost>


<VirtualHost 192.168.251.200:*>
    ....
</VirtualHost>
```

Soll ein virtueller Server mehreren Hostnamen bzw. IP-Adressen zugewiesen werden, können entsprechend viele Argumente zugewisen werden:

```
<VirtualHost 192.168.251.100 192.168.251.200>
```

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

Die <VirtualHost> Anweisung kann prinzipiell beliebig oft in der Serverkonfiguration auftauchen. Allerdings verbraucht jeder virtuelle Server Ressoourcen, die nicht unbegrenzt zur Verfügung stehen.

Die meisten Anweisungen, die auch in der Serverkonfiguration auftauchen dürfen, können auch in der <VirtualHost> Sektion benutzt werden, so z.B. ServerName, ServerAdmin, ErrorLog und insbesondere die DocumentRoot. Jeder virtuelle Server erbt die komplette Konfiguration des Hauptservers. Wird zum Beispiel nicht explizit innerhalb der <VirtualHost> Abschnittes die ErrorLog Anweisung gesetzt, wird die Logdatei des Hauptservers genutzt. Dasselbe gilt für nahezu alle anderen in der Serverkonfiguration gemachten Angaben.

Eine minimale und sinnvolle Definition eines virtuellen Servers sollte zumindest die DocumentRoot und die ServerName Anweisung enthalten:

```
<VirtualHost 192.168.251.200>
    ServerName www.mydomain.de
    DocumentRoot /var/www/http/mydomain/html
</VirtualHost>
```

In den meisten Fällen will man aber zusätzlich jedoch auch eigene Logfiles für jeden virtuellen Server haben:

```
<VirtualHost 192.168.251.200>
    ServerName www.mydomain.de
    DocumentRoot /var/www/http/mydomain/html
    ErrorLog /var/www/mydomain/logs/error_log
    CustomLog /var/www/mydomain/logs/access_log combined
</VirtualHost>
```

Bei der Deklaration eines virtuellen Servers sollte man immer bevorzugt die IP-Adresse angeben, denn ansonsten muss der Apache beim Start jeweils einen DNS-Lookup machen, um die Adresse herauszufinden. Sofern ein angegebener Hostname im Nameservice nicht eingetragen ist oder der DNS-Lookup schlägt aus anderen Gründen fehl, wird der virtuelle Server beim Start des Apache nicht aktiviert.

Bei namensbasierten Servern ist zwingend erforderlich, in der ServerName Anweisung einen Hostnamen anzugeben, denn irgendwie muss man dem Kind einen Namen geben.

7.3.2.2 NameVirtualHost

Um überhaupt namensbasierte Server verwenden zu können, wird die NameVirtualHost Anweisung benötigt. Diese Anweisung wurde mit Apache 1.3 eingeführt, um eine einfachere bzw. eindeutigere Konfiguration der virtuellen Servern zu ermöglichen.
Sollen ein oder mehrere namensbasierte virtuelle Server verwendet werden, müssen die NameVirtualHost zur Kennzeichnung der jeweiligen IP-Adresse und ggfs. die Portnummer angegeben werden. Als Adressen können Hostnamen angegeben werden, wobei jedoch eine explizite Verwendung von IP-Adressen vorzuziehen ist (s. oben).

Wird die Portangabe weggelassen, wird die Standard-Portnummer verwendet, bei der es sich in den meisten Fällen um Port 80 handeln dürfte.

Sollen beispielsweise die beiden virtuellen Servern `www1.mydomain.de` und `www2.mydomain.de` unter der IP-Adresse 192.168.251.200 eingerichtet werden, ergibt sich folgende Konfiguration:

```
NameVirtualHost 192.168.251.200

<VirtualHost 192.168.251.200>
    ServerName www1.mydomain.de
    ...
</VirtualHost>

<VirtualHost 192.168.251.200>
    ServerName www2.mydomain.de
    ...
</VirtualHost>
```

Wenn der Server nun eine Anforderung für die in der `NameVirtualHost`-Direktive angegebene IP-Adresse erhält, durchsucht er die `<VirtualHost>`-Blöcke nach dem gewünschten Servernamen und weiss so, an welchen virtuellen Host die Anforderung gerichtet ist (und welche Dokumente folglich zu liefern sind). Die Position der `NameVirtualHost` Anweisung innerhalb der Serverkonfiguration ist nicht relevant, jedoch ist die Reihenfolge der zugehörigen `<VirtualHost>` Abschnitte wichtig. Der erste virtuelle Server, der in der Konfiguration für eine zuvor mit `NameVirtualHost` definierte IP-Adresse angegeben wird, stellt den Default-Server für diese IP-Adresse dar. Dies bedeutet, dass eine Anfrage, die an diese IP-Adresse gestellt wurde, jedoch keinem der zugehörigen namensbasierten virtuellen Server zugeordnet werden kann, vom ersten virtuellen Server bedient wird.

Die Reihenfolge aller weiteren virtuellen Server der jeweiligen IP-Adresse kennzeichnet die Priorität in absteigender Folge. Eine Rolle spielt diese Priorität dieser virtuellen Server jedoch nur in Bezug auf eventuell vorhandene `ServerAlias` oder `ServerPath` Anweisungen, da immer der erste virtuelle Server verwendet wird. Der mittels `ServerName` angegebene Name hat jedoch grundsätzlich die Priorität gegenüber `ServerAlias` und `ServerPath`.

7.3.2.3 ServerAlias

Mit `ServerAlias` ist es möglich, namensbasierten virtuellen Servern weitere Hostnamen zuzuweisen:

```
ServerAlias www.mydomain.de host1.mydomain.de
```

Abgesehen von der Angabe eines kompletten Hostnamens können auch partielle Namen angegeben werden:

```
ServerAlias host1 host2 www.mydomain.de
```

Weiterhin ist die Benutzung von Mustern mit »*« oder »?« möglich:

```
ServerAlias *.myotherdomain.de host?.mydomain.de
```

Allerdings ist nicht zu vergessen, dass alle Namen, die in der `ServerAlias` Anweisung benutzt werden, auch im jeweiligen Nameserver als Alias für die mittels `NameVirtualHost` definierte IP-Adresse eingetragen werden müssen!

7.3.2.4 ServerPath

Vor HTTP/1.1 war die Zuordnung des richtigen virtuellen Hosts anhand des Servernamens nicht möglich, da der Webserver nicht feststellen konnte, an welchen Server der Browser seine Anforderung gesendet hatte. Falls der Browser keine Informationen über den Hostnamen schickt, versucht der Apache die entsprechende Seite auf dem Default-Server zu finden[4]. Mithilfe der `ServerPath` Direktive kann dies umgangen werden.

Da aber praktisch alle heute üblichen Browser HTTP/1.1 beherschen, wird an dieser Stelle auf eine weitere Betrachtung verzichtet.

[4] Wie bereits erwähnt, ist das immer der erste virtuelle Server, der für diese IP-Adresse in der Serverkonfiguration definiert wurde.

8 .htaccess - Server-Reaktionen kontrollieren

8.1 Allgemeines

Abgesehen von den normalen Server Konfigurationsdateien bietet der Apache die Möglichkeit von Verzeichniskonfigurationsdateien.

Mit `AccessFileName` können ein oder mehrere Dateinamen für die Konfigurationsdatei eingestellt werden, wobei typischerweise `.htaccess` benutzt wird:

```
# AccessFileName: The name of the file
# to look for in each directory
# for access control information.
#
AccessFileName   .htaccess
```

`.htaccess`-Dateien sind also Konfigurationsdateien für Verzeichnisse, die zum Web-Angebot gehören. So ist die `.htaccess`-Technik beispielsweise der übliche Weg, um nur bestimmten Benutzern den Zugriff auf bestimmte Daten zu erlauben. `.htaccess` ist also die Antwort auf die immer wieder gestellte Frage nach einem "richtigen" Passwortschutz. Doch mit `.htaccess`-Dateien kann man noch viel mehr anstellen als Daten mit Passwörtern zu schützen. So ist es auch möglich ganze Benutzerkreise automatisch (ohne Passwortzugang) aussperren oder alle bis auf bestimmte aussperren. Es können Optionen zum so genannten Verzeichnis-Browsing (Web-Browser ruft ein Verzeichnis auf, in dem keine Default-Datei wie z.B.index.htm liegt) eingestellt werden. Es können automatische Weiterleitungen eingestellt oder eigene Regelungen für den Fall von HTTP-Fehlermeldungen geschaffen werden. Es können sogar abhängig von bestimmten Bedingungen alternative Inhalte angeboten werden, beispielsweise Web-Seiten mit unterschiedlichen Landessprachen abhängig von der Sprache des benutzten Web-Browsers - ohne CGI, ohne JavaScript. Und schließlich kann man sogar einstellen, dass Daten komprimiert an den aufrufenden Browser übertragen werden.

Web-Server, die zum NCSA-Server kompatibel sind, kennen das Konzept der `.htaccess`-Dateien. Heutzutage trifft das vor allem auf den immer dominanter werdenden und die defacto-Standards im Web setzenden Web-Server Apache zu. Wenn auf dem Server-Rechner also ein Apache Web-Server läuft, dann ist die Technik der `.htaccess`-Dateien einsetzbar.

Dateien mit dem Namen `.htaccess` (das erste Zeichen ist ein Punkt) sind im Grunde Teil der Konfiguration des Web-Servers. Der Web-Server verfügt zwar auch über zentrale Konfigurationsdateien. Doch diese liegen normalerweise unterhalb des Programmverzeichnisses des Web-Servers und sind nicht unbedingt den Benutzern zugänglich, die ein Web-Projekt verwalten. `.htaccess`-Dateien werden dagegen in dem Verzeichnisbereich abgelegt, in dem das Web-Angebot liegt. Also dort, wo die HTML-Dateien, Grafikdateien usw. liegen, werden also auch die `.htaccess`-Dateien abgelegt.

Dass der Name `.htacess` mit einem Punkt beginnt, ist eine Tradition aus der Unix-Welt. Dort beginnen viele verzeichnis-spezifische Konfigurationsdateien mit einem Punkt. Bekannt ist beispielsweise

die Datei `.profile`. In der zentralen Konfiguration des Apache Web-Servers lässt sich allerdings auch ein anderer Dateiname als `.htaccess` einstellen.

`.htaccess`-Dateien sind reine Textdateien, die mit einem Texteditor bearbeitet werden müssen. Wenn man Telnet- oder SSH-Zugang zu dem Server-Rechner hat, auf dem das Web-Angebot liegt, können solche Dateien direkt auf dem Server erstellt und bearbeitet werden, indem dort ein geeigneter Texteditor aufgerufen wird (auf Unix-Systemen z.B. den vi-Editor). Wenn allerdings nur FTP-Zugang zu dem Server-Rechner besteht, können solche Dateien auf dem lokalen PC mit einem Texteditor erstellt und dann hochgeladen werden.

Alle in diesem Abschnitt einzeln vorgestellten Möglichkeiten können auch in einer einzigen `.htaccess`-Datei kombiniert werden. So kann man also beispielsweise zugleich einen Passwortschutz einrichten und gleichzeitig Optionen zum Verzeichnis-Browsing oder für das Senden alternativer Inhalte notieren.

`.htaccess`-Dateien gelten stets für das Verzeichnis, in dem sie abgespeichert werden, sowie für alle Unterverzeichnisse davon und für deren Unterverzeichnisse. Wenn in einem der Unterverzeichnisse andere Regelungen gewünscht sind, muss dort wiederum eine entsprechende `.htaccess`-Datei abgelegt werden. Die Angaben der jeweils aktuellen Datei überschreiben die Angaben aus den Verzeichnissen oberhalb.

8.2 Verzeichnisse und Dateien mit Passwort schützen

`.htaccess`-Dateien sind verzeichnis-spezifisch. Das heißt die `.htaccess`-Datei wird also in dem Verzeichnis abgespeichert, in dem die geschützten Daten liegen. Es kann wahlweise das ganze Verzeichnis mit all seinen Unterverzeichnissen oder nur bestimmte Dateien oder Dateitypen geschützt werden. Der Passwortschutz kann außerdem wahlweise für einzelne Benutzer oder für ganze Benutzergruppen eingerichtet werden. Auch Kombinationen beider Formen sind möglich. Damit der Verzeichnisschutz mit Passwort funktioniert, genügt die `.htaccess`-Datei alleine allerdings nicht. Es wird zusätzlich eine Datei, in der die Benutzer und ihre Passwörter stehen, benötigt. Falls mit Benutzergruppen gearbeitet wird, benötigt man außerdem noch eine Datei, in der die Benutzergruppen definiert werden. Auch diese beiden anderen Dateien werden mit einem Texteditor erstellt. Eine `.htaccess` könnte beispielsweise folgendermaßen aufgebaut sein:

```
# .htaccess-Datei fuer Web-Verzeichnis /service
AuthType Basic
AuthName "Service-Bereich"
AuthUserFile /usr/verwaltung/web/.htusers
AuthGroupFile /usr/verwaltung/web/.htgroups
require user Klara Dieter Heidi
require group Servicetechniker
```

Die erste Zeile der Beispieldatei ist lediglich ein Kommentar. Solche Kommentarzeilen müssen mit dem Gatterzeichen # eingeleitet werden. Alle anderen Zeilen haben einen einheitlichen Aufbau: sie beginnen mit Schlüsselwörtern, und dahinter folgen, durch ein oder mehrere Leerzeichen getrennt, Angaben dazu. Leerzeilen sind erlaubt. Zeilen mit Angaben, die der Web-Server nicht interpretieren kann, führen jedoch möglicherweise zu einem "Internal Server Error" (HTTP-Statuscode 500).

Um einen Passwortschutz einzurichten, werden folgende Schlüsselwörter benötigt:

- **AuthType** Mit `AuthType` wird die Art der Authentifizierung bezeichnet. Die übliche Angabe ist Basic - dabei stehen die Benutzer und Passwörter in einer noch anzugebenden Datei. Eine anderer denkbarer Authentifizierungstyp wäre PGP (Pretty Good Privacy).
- **AuthName** Mit Hilfe dieser Anweisung lässt sich dem Benutzer mitteilen, für welchen Bereich er sich mit User-Name und Passwort authentifizieren soll. Der Bereich kann eine beliebige Zeichenkette sein. Enthält die Zeichenkette Leerzeichen, muss sie in Anführungszeichen eingeschlossen werden.
- **AuthUserFile** Bei `AuthUserFile` wird die Datei angegeben, in der die autorisierten Benutzer und ihre Passwörter stehen. Es muss der vollständige absolute Pfadname angegeben werden. Also ist nicht der Pfad ab dem Wurzelverzeichnis Ihres Web-Projekts, sondern ab dem Wurzelverzeichnis des Server-Rechners. Im Interesse der Sicherheit ist es auch besser, die Datei mit den Benutzern und Passwörtern außerhalb des Web-Projekts auf dem Server abzulegen. Aber nicht bei allen Hosting-Angeboten ist es möglich, auf den Server-Rechner außerhalb des eigenen Web-Projekts zuzugreifen. In diesem Fall sollte für die Datei auf jeden Fall einen Dateinamen gewählt werden, der mit `.ht` beginnt. Denn im Apache-Server werden solche Dateien per Voreinstellung vor Besuchern verborgen. Das Gleiche gilt für die Gruppendatei, falls mit Benutzergruppen gearbeitet wird. Diese wird mit `AuthGroupFile` angegeben.
- **AuthGroupFile**, wenn Benutzergruppen verwendet werden sollen
- **require** Bei `require` geben Sie als zweites Schlüsselwort entweder `user` oder `group` an, je nachdem, ob Sie einzelne Benutzer oder Benutzergruppen meinen. Dahinter können ein oder mehrere Namen von Benutzern oder Benutzergruppen folgen. Sollen alle in der Passwortdatei aufgeführten Benutzer Zugriff auf das geschützte Verzeichnis gewährt werden, dann ist als zweites Schlüsselwort `valid-users` anzugeben.

Im obigen Beispiel werden die drei Benutzer Klara, Dieter und Heidi sowie alle Benutzer der Gruppe Servicetechniker angegeben. Damit der Passwortschutz funktioniert, müssen nun die angegebenen Dateien mit den Benutzern und den Gruppen angelegt werden. Besipielhaft könnte eine solche Passwortdatei (im nachfolgenden `.htusers` genannt) folgendermaßen aussehen:

```
# Benutzerdatei fuer Web-Projekt
Klara:INY8m5KMwIc
Janine:INw2mPEH.owe2
Anke:INh6DHvyejvf2
Bernd:INboWuvjjwQ7E
Karin:INwOXOz96UQOU
Christina:INXo9kh0M.anc
Andreas:INeRD/cUQIFP6
Dieter:INUnlKdkNZ6RQ
Heidi:IN20ffIEEV1H6
```

Auch in solchen Dateien sind Kommentarzeilen erlaubt, eingeleitet durch #. Ansonsten enthält jede Zeile der Benutzerdatei einen Benutzernamen, und gleich dahinter, durch einen Doppelpunkt getrennt, das Passwort. Bei Unix-Systemen sind das aber nicht die Passwörter selbst, sondern die verschlüsselten Passwörter. Dazu müssen die Passwörter mit Hilfe des crypt-Mechanismus verschlüsselt werden. Zum Erzeugen und Verwaltung der User-Datei gibt es den Unix-Befehl `htpasswd`.

8.2.1 htpasswd

Die Passwortdatei kann mithilfe des Apache Utility htpasswd erzeugt werden. Die Aufrufsyntax des Kommandos lautet:

```
htpasswd [Optionen] Passwortdatei User [Passwort]
```

Die Angabe von `-c` führt dazu, dass eine neue datei erstellt wird. Beim folgenden Beispiel wird `htpasswd` für die einzelnen User aufgerufen. Bei der ersten User-ID wird also die Option `-c` angegeben, damit die Datei `.htusers` angelegt wird:

```
[root@delphin]$ htpasswd -c /var/www/.htusers Klara
Adding password for Klara.
New Password:
Re-type new password:
```

Die weiteren User werden entsprechend dem Beispiel zu Klara eingetragen, allerdings ohne der Option `-c`, da die Datei bereits vorhanden ist (s.o):

```
[root@delphin]$ htpasswd /var/www/.htusers Janine
Adding password for Janine.
New Password:
Re-type new password:
```

Um das Passwort zu ändern, wird `htpasswd` in gleicher Weise wie auch beim Eintragen eines Users aufgerufen:

```
[root@delphin]$ htpasswd /var/www/.htusers Janine
Changing password for user Janine.
New Password:
Re-type new password:
```

8.2.2 Die Gruppendatei

Gruppendateien bestehen aus Einträgen, bei denen zunächst ein Gruppenname notiert wird und dahinter, nach einem Doppelpunkt, die Namen von Benutzern, die zu dieser Gruppe gehören. Es müssen Benutzernamen sein, für die in der Benutzerdatei ein Eintrag angelegt wurde.

Die Gruppendatei wird im Beispiel nur benötigt, weil in der `.htaccess`-Datei eine Benutzergruppe angegeben wurde. Während die Benutzerdatei in jedem Fall benötigt wird, ist die Gruppendatei nur erforderlich, wenn Gruppennamen benutzt werden. Nachfolgend findet sich ein Beispiel einer Gruppendatei:

```
# Gruppendatei fuer Web-Projekt
Servicetechniker: Andreas Karin Janine
```

Alle Besucher des Web-Projekts, die nun versuchen, auf das Verzeichnis mit der `.htaccess`-Datei zuzugreifen, bekommen von ihrem Browser einen Dialog angeboten, in dem sie Benutzernamen und Passwort eingeben müssen. Nur Besucher, die sich mit einer gültigen Kombination aus Benutzernamen und Passwort anmelden, haben Zugriff auf das Verzeichnis.

8.3 Schutz von Dateien, Dateitypen oder Zugriffsmethoden

So wie im obigen Beispiel gezeigt, gilt der Zugangsschutz für das Verzeichnis, in dem die `.htaccess`-Datei liegt, und für alle Verzeichnisse unterhalb davon. Sie können den Schutz aber auch auf bestimmte Dateien, Dateitypen oder Zugriffsmethoden einschränken.

Beispiel:

```
# .htaccess-Datei fuer Web-Verzeichnis /service
AuthType Basic
AuthName "Service-Bereich"
AuthUserFile /usr/verwaltung/web/.htusers
AuthGroupFile /usr/verwaltung/web/.htgroups
<Files *.htm>
   require user Werner Dieter Heidi
   require group Servicetechniker
</Files>
```

Um den Schutz einzuschränken, benutzt man ähnlich wie in HTML oder XML Tags mit spitzen Klammern. Im einleitenden Tag kann hinter der öffnenden spitzen Klammer entweder Files stehen, wie im obigen Beispiel, oder Limit. Dahinter können man genau eine einschränkende Angabe machen. Mit *.htm wie im Beispiel wird der Schutz auf HTML-Dateien beschränkt. Mit einer Angabe wie geheim.htm würde nur diese eine Datei geschützt. Bei Limit kann die Zugriffsmethode z.B. auf GET oder POST eingeschränkt werden.

Falls es mit dem Schützen von Verzeichnissen nicht klappen will , dann könnte der Grund sein, dass in der zentralen Konfiguration des Apache Web-Servers beim Eintrag `AllowOverride` zu wenig erlaubt ist. In diesem Fall muss z.B. `All` als Wert zugewiesen werden.

Schutzmechanismen, die mit Hilfe von `.htaccess`-Dateien erstellt werden, sind auf HTTP-Ebene wesentlich sicherer als solche, die mit Hilfe von CGI-Scripts oder gar mit Hilfe von JavaScript erstellt werden. Dennoch sollten man wissen, dass `.htaccess` keinen Generalschutz bietet. Der Schutz gilt nur, wenn Web-Browser oder andere Web-Clients über den Web-Server geschützte Daten anfordern. Er gilt nicht, wenn der Zugriff z.B. mit einem anderen Internet-Protokoll wie FTP erfolgt.

8.4 IPs, IP-Bereiche oder Namensadressen zulassen/ausschließen

Weiterhin ist es möglich, bestimmte IP-Adressen oder IP-Bereiche davon auszuschließen, auf Web-Seiten zuzugreifen. Ebenso können alle IP-Adressen ausgeschlossen und nur ganz bestimmten den Zugriff erlaubt werden. Anwender, die über eine nicht autorisierte IP-Adresse zugreifen, erhalten dann eine HTTP-Fehlermeldung (Seite HTTP-Statuscode 403), und der Zugriff wird ihnen verweigert.

Sinnvoll ist das beispielsweise, wenn das Web-Angebot nur den Mitarbeitern einer Firma zugänglich sein soll, oder wenn bestimmte Anwender, die mit festen IP-Adressen unterwegs sind, wegen Fehlverhaltens etwa in einem von Ihnen angebotenen, web-basierten Forum oder Chat ausschließen möchten.

Beispiel:

```
# Datei zum Regeln von IP-Bereichen
Order deny,allow
Deny from .aol.com
```

```
Deny from 192.168
Allow from 192.168.220.102
```

Zunächst wird mit Order die logische Interpretationsreihenfolge der nachfolgenden Angaben festgelegt. Möglich ist die Angabe `deny,allow` wie im Beispiel, oder auch die umgekehrte Reihenfolge.

In Zeilen, die mit `Deny from` oder `Allow from` beginnen, steht eine konkrete IP-Adresse, einen Teil davon, eine Namensadresse oder einen Teil davon an. Mit `Deny from` verbietet man den Zugriff für den oder die angegebenen Benutzer, und mit `Allow from` wird der Zugriff erlaubt. Per zentraler Voreinstellung ist normalerweise der Zugriff für alle Benutzer erlaubt. In `.htaccess`-Dateien ist es deshalb sinnvoll, vor allem einschränkende Verbote zu formulieren. Im obigen Beispiel werden alle Benutzer ausgesperrt, die mit einer AOL-Kennung surfen, (.aol.com), sowie alle Benutzer mit numerische IP des Bereichs 192.168. Um aber einem bestimmten Benutzer aus diesem Bereich doch den Zugriff zu erlauben, wird im Beispiel anschließend noch mit `Allow from` dessen IP-Adresse angegeben.

Die Angabe wird einfach als Teilzeichenkette interpretiert. Wenn ein Client eine Web-Seite in dem Verzeichnis mit der `.htaccess`-Datei aufruft, vergleicht der Web-Server, ob eine der notierten Zeichenketten in der Zeichenkette vorkommt, die der aufrufende Client dem Server übermittelt.

Anstelle einer bestimmten Zeichenkette kann man auch `Allow from all` bzw. `Deny from all` notieren, um eine generelle Erlaubnis bzw. ein generelles Verbot zu formulieren.

Beim Aussperren und Einschließen bestimmter IPs, IP-Bereiche oder Namensadressen sind die gleichen nach oben erweiterten Möglichkeiten erlaubt wie beim Passwortschutz. Die Anweisungen mit `Deny from` und `Allow from` können dabei in entsprechende Tags eingeschlossen werden. So lässt sich z.B. das Aussperren bestimmter Benutzer oder Benutzerkreise auf bestimmte Dateien oder Zugriffsmethoden beschränken.

8.5 Verzeichnis-Optionen einstellen

Mit Hilfe einer `.htaccess`-Datei können Sie alle zentralen Verzeichnisoptionen, die in der Konfiguration des Web-Servers eingestellt sind, für das entsprechende Verzeichnis und seine Unterverzeichnisse ändern. Voraussetzung ist allerdings, dass in der zentralen Web-Server-Konfiguration `AllowOverride All` eingetragen ist. Wenn nicht, dann werden Angaben für andere Verzeichnisoptionen ignoriert.

```
# Datei fuer Verzeichnis /bilder
Options +ExecCGI -Indexes
DirectoryIndex erste.htm start.htm
ErrorDocument 403 "Auf dieses Verzeichnis ist der Zugriff verboten"
ErrorDocument 404 /spezial/404.html
ErrorDocument 500 http://www.mydomain.de/spezial/500.html
```

Mit `Options` können man verschiedene wichtige Grundeinstellungen ändern, die der Web-Server für Verzeichnisse verwendet. Hinter `Options` können eine oder mehrere der folgenden Angaben notiert werden:

- **`+ExecCGI`** erlaubt das Ausführen von CGI-Scripts im Verzeichnis, falls es zentral nur im definierten CGI-Verzeichnis erlaubt ist, und **`-ExecCGI`** verhindert es, sofern es zentral in allen Verzeichnissen erlaubt ist.

- **+Includes** erlaubt das Ausführen von Server Side Includes im Verzeichnis, falls es zentral verboten ist, und **-Includes** verhindert es, sofern es zentral erlaubt ist.
- **+IncludesNOEXEC** erlaubt das Ausführen von Server Side Includes, die kein CGI-Script ausführen, und **-IncludesNOEXEC** verbietet solche Server Side Includes.
- **+Indexes** erlaubt Verzeichnis-Browsing, falls es zentral verboten ist, und **-Indexes** verhindert es, sofern es zentral erlaubt ist. Beim Verzeichnis-Browsing wird der Inhalt des Verzeichnisses im Browser aufgelistet, falls nur das Verzeichnis aufgerufen wird (z.B. http://www.ihr-name.de/bilder/) und dort keine Datei mit einem Namen zu finden ist, der "Index-Funktion" hat.
- Mit **+MultiViews** wird das Definieren alternativer Inhalte erlaubt, mit **-MultiViews** schalten man es aus.

Im Normalfall ist der Web-Server so konfiguriert, dass er das "Stöbern" im Verzeichnis zulässt, falls dort keine Datei mit "Index-Funktion" existiert. Das sind in der Einstellung der meisten Web-Server Dateien mit Namen wie `index.htm` oder `index.html`. Mit `DirectoryIndex` können andere Dateinamen bestimmt werden, die in diesem Verzeichnis und seinen Unterverzeichnis als Dateien mit Index-Funktion gelten. Dabei können beliebig viele Dateinamen angeben werden.

Per Voreinstellung zeigt der Web-Server eine in seinen Konfigurationen enthaltene HTML-Datei an, wenn ein Zugriffsfehler passiert, z.B. wenn eine angeforderte Datei nicht existiert. Auch dies ist veränderbar: Mit `ErrorDocument` definiert man eine eigene Meldung oder Datei, die im Fehlerfall angezeigt wird. Hinter dem Schlüsselwort wird der gewünschte HTTP-Statuscode notiert - typische Fehler, die auftreten können, sind jene mit den Nummern 403, 404 und 500. Hinter der Nummer mit dem Statuscode kann dann entweder in Anführungszeichen eine eigene Fehlermeldung notieren werden , oder es wird die Adresse einer Web-Seite angegeben, die im Fehlerfall angezeigt werden soll. Bei Adressen auf der eigenen Domain kann mit absoluten Pfadnamen gearbeitet werden, wobei das Wurzelverzeichnis des Web-Projekts der Ausgangspunkt ist. Falls nur eine Fehlermeldung beschrieben wurde, wird diese bei Eintreten des Fehlers als reiner Text im Browser angezeigt. Angegebene Web-Seiten können dagegen nach Lust und Laune gestaltet werden.

8.6 Verzeichnis-Browsing einstellen

Wenn Verzeichnis-Browsing erlaubt ist, so dass besispielsweise also bei einem Aufruf der Adresse `http://www.domain.de/bilder/` der Verzeichnisinhalt aufgelistet wird, kann man die Optik beeinflussen, mit der die Verzeichnislisten im Browser angezeigt werden. Leichter verständlich wird das, wenn man sich klarmacht, dass der Web-Browser beim Anzeigen eines Verzeichnislistings einfach dynamisch HTML-Code generiert, um den eingelesenen Verzeichnisinhalt anzuzeigen. Mit den hier beschriebenen Optionen kann die HTML-Ausgabe des Web-Servers beeinflusst werden.

Abbildung 8.1 zeigt, wie ein Browser ein Verzeichnislisting anzeigen könnte, bei dem eigene Dateibeschreibungen und Symbolgrafiken zum Einsatz kommen.

Die entsprechende `.htaccess` könnte beispielsweise folgendermaßen aufgebaut sein:

```
# Datei fuer Verzeichnis /bilder
FancyIndexing On
AddDescription "HTML-Datei, anzeigbar" *.htm *.html
```

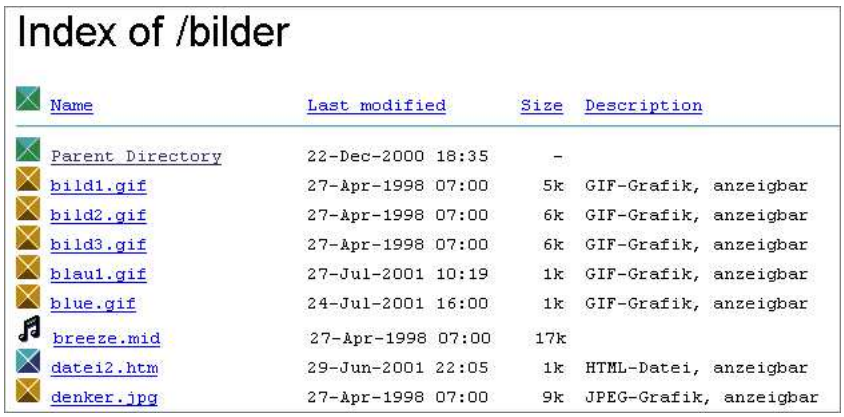

Abbildung 8.1: angepasste Verzeichnisdarstellung

```
AddDescription "GIF-Grafik, anzeigbar" *.gif
AddDescription "JPEG-Grafik, anzeigbar" *.jpg
AddDescription "ZIP-Archiv, downloadbar" *.zip
AddIcon /src/blau.gif html htm
AddIcon /src/gelb.gif gif jpg
AddIcon /src/gruen.gif ^^DIRECTORY^^
AddIcon /src/blau.gif ^^BLANKICON^^
```

Mit `FancyIndexing On` schaltet man eigene Optionen zur Darstellung des Verzeichnislistings ein. Die Einträge der übrigen Zeilen im obigen Beispiel definieren solche Anzeigeoptionen.

Mit `AddDescription` kann ein kurzer Beschreibungstext angegeben werden, der hinter einer oder mehreren bestimmten Dateien stehen soll. Zuerst wird in Anführungszeichen der gewünschten Text angegeben, und dahinter für welche Datei oder Dateien der Text gelten soll. Dabei können auch wie im Beispiel oben Wildcards benutzen.

Mit `AddIcon` wird für eine oder mehrere bestimmte Dateien eine eigene Symbolgrafik bestimmt. Dazu wird der URL der gewünschten Symbolgrafik angegeben. Bei Adressen auf der eigenen Domain kann man mit absoluten Pfadnamen arbeiten, wobei das Wurzelverzeichnis des Web-Projekts der Ausgangspunkt ist. Hinter der Angabe zur gewünschten Grafikdatei werden die Dateiendungen gesetzt. Dateien mit dieser Endung erhalten dann bei der Anzeige die entsprechende Symbolgrafik. Anstelle von Dateiendungen ist es aber auch möglich, reservierte Ausdrücke wie `^^DIRECTORY^^` (Verzeichnisse erhalten diese Symbolgrafik) oder `^^BLANKICON^^` (Dateien ohne Dateiendung oder nicht zuzuordnende Dateien erhalten diese Symbolgrafik) anzugeben.

9 Secure Webserver

Für die kommerzielle Nutzung des Internet ist es oftmals notwendig, eine sichere Kommunikation zu ermöglichen, z.B. wenn Kreditkartendaten übertragen werden sollen. Bei einer gesicherten Übertragung spielt in erster Rolle die Verschlüsselung der übertragenen Daten eine Rolle, aber auch die Authentikation ist hierbei nicht unwesentlich, denn es nützt wenig, wenn z.B. die Kreditkartendaten zwar verschlüsselt, aber an die falsche Website übermittelt wurden. Andersherum soll der Webserver ebenfalls die Möglichkeit haben, eine Authentikation des Webclients durchzuführen.

9.1 Das eigentliche Problem

Man stelle sich zwei Kommunikationspartner vor. Der Einfachheit halber nennen wir sie A und B. Diese kennen einander nicht. Daher besitzen sie kein gemeinsames Wissen von dem sie annehmen können, dass kein anderer dieses Wissen mit ihnen teilt.

A und B selbst sitzen in einer geschützten Umgebung. Alles, was sie sozusagen innerhalb ihrer eigenen vier Wände treiben, bleibt für Außenstehende unsichtbar. Leider ist jedoch der Kommunikationskanal, der von A nach B führt, öffentlich und kann von jeder man abgehört werden. A und B müssen daher mit einem C rechnen, das von Anfang an ihre Gespräche belauscht. Und nicht nur das, C hat auch die Möglichkeit beliebig Nachrichten aus diesem Nachrichtenkanal herauszunehmen und diese verändert an den Bestimmungsort weiter zu senden, ohne dass der Empfänger dies bemerkt.

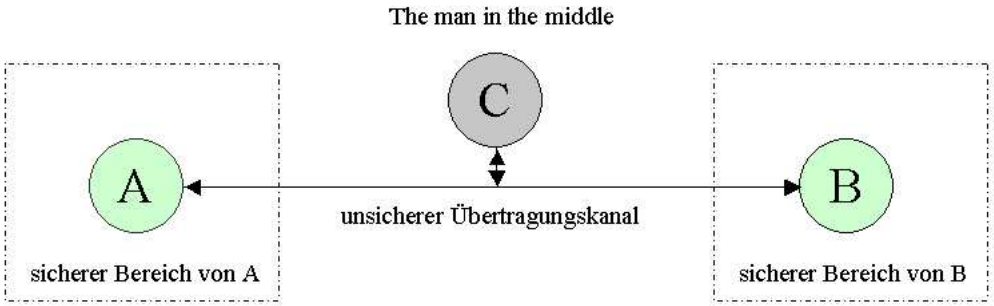

Abbildung 9.1: »The man in the middle«

Dieses C wir oft "The man in the middle" genannt. Wenn dieser von der Möglichkeit gebrauch macht Nachrichten zu verändern, so spricht man von aktiven sonst von passiven Angriffen.

A und B stehen nun vor der Aufgabe sich auf eine Sprache zu einigen, die C nicht versteht. Dies ist in der im Abb. 9.1 dargestellten Situation unmöglich. Denn wenn C ein aktiver Angreifer ist, stehen für A und B keine Möglichkeiten zur Verfügung das in Abb. 9.2 dargestellte Problem zu lösen.

Hier gibt sich von Anfang an C aus der Sicht von A als B aus und aus der Sicht von B scheint es so,

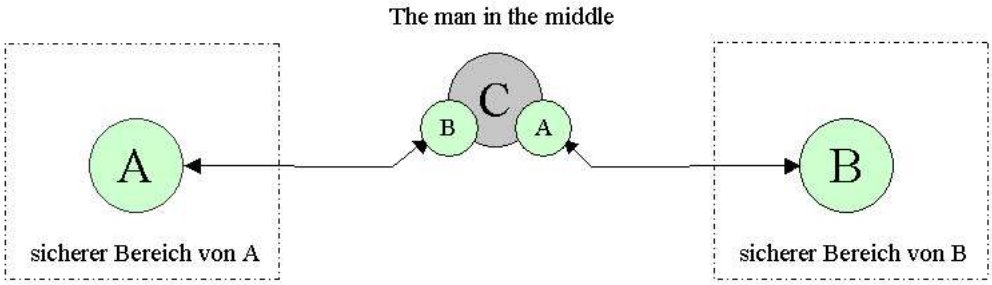

Abbildung 9.2: ein aktiver »man in the middle«

als wäre C A. Wann immer A eine Nachricht an B schickt, nimmt C diese entgegen und leitet diese an B weiter. Liefert B schließlich die Antwort, nimmt C diese zur Kenntnis und stellt sie A zu. Wenn A eine verschlüsselte Verbindung mit B aufbaut, so tut A dies in Wirklichkeit mit C. Dieses richtet als Folge eine sichere Verbindung mit B ein, so als würde dies der Wunsch von A sein. Selbst wenn A nun eine verschlüsselte Nachricht an B schickt, kann C diese entschlüsseln, diese zur einsehen und anschließend wieder für B verschlüsseln usw.

9.1.1 Die Lösung

Dieses Problem wäre gelöst, wenn zum Beispiel B einen Ausweis besitzt, mit dem es sich gegenüber A eindeutig ausweisen könnte. Dieser Ausweis muss natürlich so gestaltet sein, dass sich kein anderer als B selbst mit diesem Ausweis ausweisen kann. Zumindest muss es später für A eine Möglichkeit geben, festzustellen, ob der Ausweis direkt von B stammt oder ob sich ein C fälschlicherweise als B ausgibt.

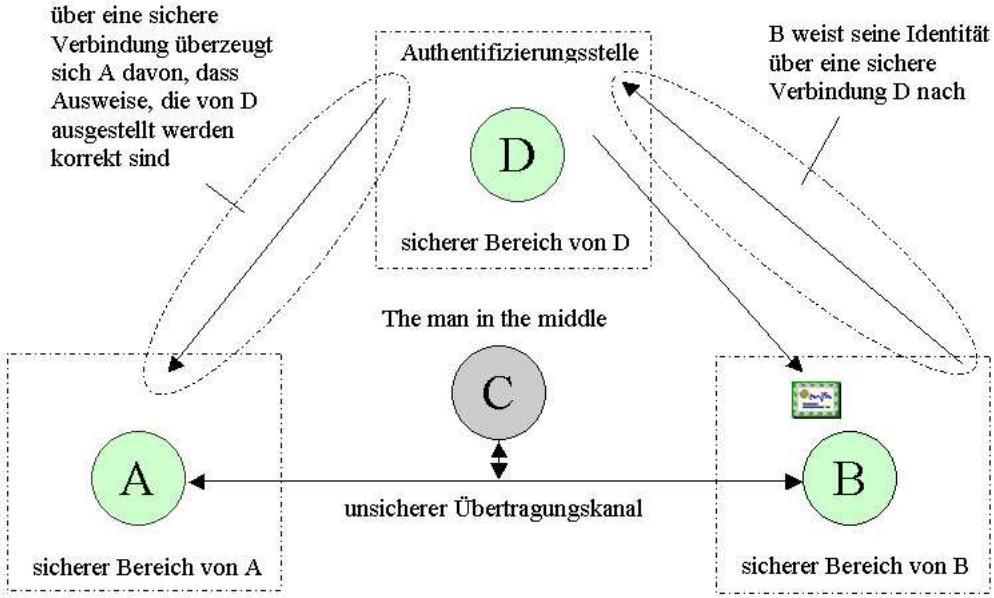

Abbildung 9.3: Die Rolle der Authentifizierungsstelle

Theoretisch zeigt sich, dass C jeden Ausweis fälschen kann! Dies kann jedoch C so stark erschwert werden, das die Kosten die C aufwenden müsste, um diesen Ausweis zu fälschen höher sind als der Gewinn, den C durch Abhören der Nachrichten zwischen A und B erreichen könnte.

Nun ist noch die Frage zu klären, wer den Ausweis für B ausstellt. B darf sich diesen Ausweis nicht selbst ausstellen können, denn sonst könnte sich auch C einen Ausweis anfertigen, in dem sich C unter dem "Namen" von B ausweist.

Wie Abb. 9.3 zeigt, wird neben A und B noch eine dritte vertrauenswürdige Person D benötigt. Hier muss sich A einmalig davon überzeugen, dass alle Ausweise die von D ausgestellt werden korrekt sind.

Bevor die Authentifizierungsstelle einen Ausweis ausstellt, muss der Antragsteller seine Identität eindeutig nachweisen. Dies muss oder soll nicht auf elektronischem Wege geschehen. Idealerweise muss sich B physisch an den Aufenthaltsort von D begeben und dort seinen Ausweis beantragen.

Die im Abb. 9.3 dargestellte Situation zeigt die Rahmenbedingungen, in der SSL (siehe Abschnitt 9.3) nun eine sichere Verbindung zwischen A und B gewährleisten kann.

9.2 Verschlüsselungsverfahren allgemein

Bei allen Verschlüsselungsverfahren werden bestimmte Wörter oder Zeichenkombinationen als Schlüssel verwendet. Das Verschlüsselungsprogramm berechnet aus dem unverschlüsselten Text und dem Schlüssel den verschlüsselten Text. Beim Entschlüsseln wird umgekehrt aus dem verschlüsselten Text und dem Schlüssel der Originaltext rekonstruiert. Die Sicherheit von modernen Verschlüsselungsverfahren beruht also darauf, dass die Schlüssel, die zur Ent- bzw. Verschlüsselung von Nachrichten benötigt werden, nur sehr schwer oder praktisch gar nicht gefunden werden können.

Grundsätzlich werden im Internet für die gesicherte Kommunikation zwei Verschlüsselungsverfahren eingesetzt: *symmetrische* und *asymmetrische Verschlüsselung*.

- **Symmetrische Verschlüsselung** Bei der symmetrischen Verschlüsselung dient derselbe Schlüssel zum Ver- und Entschlüsseln der Nachricht. Dieser Schlüssel muss allen Beteiligten bekannt sein. Dazu müssen die Schlüssel zunächst den Kommunikationspartnern irgendwie bekannt gemacht

Abbildung 9.4: Symmetrische Verschlüsselung

werden. Dieses Verfahren ist also dann ungeeignet, wenn sich die Kommunikationspartner nicht persönlich begegnen, da vor Einsatz der gesicherten Kommunikation dieser Schlüssel ausgetauscht werden muß und dieser dann ungeschützt über das Internet übertragen werden müßte. Gerät der Schlüssel dabei in die Hände eines Dritten ist es für diesen relativ einfach, die übertragenen Daten zu decodieren. Der Hauptvorteil der symmetrischen Verschlüsselung ist die Geschwindigkeit mit

der die Daten ver- und entschlüsselt werden. Daher wird versucht den Großteil der zu übertragenden Datenmengen mit symmetrischen Verfahren zu verschlüsseln. Sie sind dafür ausgelegt, große Datenmengen in sehr kurzer Zeit zu Ver- und Entschlüsseln. Symmetrische Algorithmen sind unter anderem *DES*, *TripleDES* und *Rijndael*. Symmetrische Algorithmen können nur da verwendet werden, wo der Schlüssel nicht übermittelt werden muß. Außer man übermittelt den Schlüssel für symmetrische Verschlüsselung über einen Kanal, der mit asymmetrischer Verschlüsselung gesichert ist.

- **Assymmetrische Verschlüsselung** Bei den *asymmetrischen* Verschlüsselungsmethoden verwenden der Sender und der Empfänger unterschiedliche Schlüssel zur Ver- bzw. zur Entschlüsselung. Bei diesen Schlüsseln handelt es sich um sogenannte Schlüsselpaare, welche immer aus einem privatem Schlüssel (dem *Private Key*), den nur der Sender kennt, und einem öffentlichen Schlüssel (dem *Public Key*), der allgemein zugänglich ist, bestehen. Daten, die mit dem privaten Schlüssel verschlüsselt wurden, können nur mit dem öffentlichen Schlüssel entschlüsselt werden. Umgekehrt, kann eine mit dem öffentlichen Schlüssel verschlüsselte Nachricht nur mit dem dazugehörigen privaten Schlüssel auch wieder entschlüsselt werden.
  Der Empfänger der Daten erzeugt einen öffentlichen und einen privaten Schlüssel. Der öffentliche Schlüssel wird an die Gegenstelle übermittelt. Diese verwendet den öffentlichen Schlüssel des Empfängers um die Daten zu verschlüsseln. Entschlüsselt werden können die Daten jedoch nur vom Besitzer des passenden privaten Schlüssels, der unbedingt geheim bleiben muss. Ein Angreifer, der die Datenübertragung abhorcht, hat somit keine Chance die Daten zu entschlüsseln, da ihm der private Schlüssel fehlt.

Abbildung 9.5: Verschlüsselung mit privatem Schlüssel

In dieser Situation kann sich B sicher sein, dass jede Nachricht, die sich mit dem öffentlichen Schlüssel von A entschlüsseln lässt, auch tatsächlich von A verschlüsselt wurde, da nur A den nötigen privaten Schlüssel kennt.

In diesem Fall kann sich B wiederum sicher sein, dass die Nachricht die mit dem öffentlichen Schlüssel von A verschlüsselt wurde, auch nur von A entschlüsselt werden kann.

Das bekannteste asymmetrische Verschlüsselungsverfahren ist sicherlich *RSA*. Diese Methode nützt die Tatsache, dass es sehr schwierig ist große Zahlen in ihre Primfaktoren zu zerlegen. So besteht bei diesem Verfahren der öffentliche Schlüssel im wesentlichen aus dem Produkt zweier großer Primzahlen. Um nun vom Public Key auf den Private Key schließen zu können, müsste eine solche Zerlegung gefunden werden. Und dies ist eben sehr aufwendig, da alle heute bekannten

Abbildung 9.6: Verschlüsselung mit öffentlichem Schlüssel

Methoden zur Faktorisierung einer Zahl in Abhängigkeit von der Größe dieser Zahl exponentielle Zeit benötigen.
Der Nachteil von asymmetrischen Verschlüsselungsmethoden besteht im hohen Rechenaufwand beim Ver- und Entschlüsseln. Der Vorteil hingegen liegt in der einfachen Methode des Schlüsselaustausches. Da den öffentlichen Schlüssel jeder kennen darf, kann dieser dem Kommunikationspartner einfach zugesandt werden.

9.3 Secure Socket Layer

Um einen sicheren Webserver einzurichten, bietet sich die Verwendung eines Webservers an, der SSL unterstützt. Das SSL Protokoll verwendet sowohl symmetrische als auch asymmetrische Verschlüsselung. Dabei wird die asymmetrische Verschlüsselung in SSL nur zu Beginn zum Verbindungsaufbau und zur Vereinbarung des symmetrischen Schlüssels verwendet. Die Abkürzung SSL steht für ***S**ecure **S**ocket **L**ayer*, was wörtlich übersetzt so was wie sichere Verbindungsschicht heißt. Hierbei handelt es sich um ein Protokoll für die sichere, verschlüsselte Übertragung von Nachrichten im Internet. Entgegen der landläufigen Meinung, SSL habe nur etwas mit der Verschlüsselung von HTTP-Sessions zu tun, ist es vielmehr so aufgebaut, dass sich eigentlich beliebige Dienste, wie z.B. SMTP, POP3 oder auch FTP verschlüsseln lassen.

Das SSL-Protokoll wurde von Netscape entwickelt. Die ersten Entwürfe wurden im Juli 1994 als quasi Version 1.0 veröffentlicht. Im Dezember des selben Jahres wurde letztlich die Version 2.0 publiziert, mit der auch die eigentliche Nutzung des Protokolls begann - so wurde es zum Beispiel im Netscape Navigator 2 implementiert. Die SSL-Version 2.0 wurde gut angenommen, und im darauffolgenden Jahr 1995 gab es schon diverse, unabhängige Implementationen dieses Standards. Nach einigen Optimierungen und kleineren Fehlerkorrekturen wurde im November 1995 die Version 3.0 veröffentlicht, welche heutzutage noch genutzt wird und weit verbreitet ist.

Die SSL Integration in den Apache ist für den Apache in der Version 1.3 vergleichsweise kompliziert, da die benötigten SSL-Schnittstellen nicht im Apache enthalten sind. Aufgrund der ehemals bestehenden Exportbestimmungen der USA, die es untersagten, das kryptografische Software aus den USA exportiert wird, durften weder die SSL Funktionalität noch die benötigten Schnittstellen im normalen Apache enthalten sein. Da die meisten der Apache Entwickler aus den USA stammne, wurde deshalb darauf verzichtet, SSL-Unterstützung in den Apache zu integrieren. Die Exportrestriktionen wurden mittlerweile gelockert, wodurch es möglich ist, die SSL Funktionalität in den Apache zu integrieren. Bei Apache 1.3 muss weiterhin auf zusätzliche Module und Erweiterungen zur SSL Integration zurückgegriffen werden.

Ab Apache 2.0 beeinhaltet die Standard Distribution des Apache bereits `mod_ssl`.

9.3.1 mod_ssl und OpenSSL

Bei `mod_ssl` handelt es sich um das Modul, was dem Apachen die SSL Funktionalität letztendlich zur Verfügung stellt. Dieses Modul implementiert jedoch nicht selbst das SSL Protokoll, sondern es handelt sich hierbei um die Schnittstelle zwischen Apache und einer entsprechenden SSL Bibliothek. Vorraussetzung für ein funktionsfähiges `mod_ssl` ist daher ein korrekt installiertes `OpenSSL`-Packet, welches eine Bibliothek der grundlegensten kryptographischen Funktionen auf dem entsprechendem System bereitstellt.

9.4 Verbindung über SSL

Wie funktioniert eine Verbindung über den Secure Socket Layer? Der Aufbau zu einer gesicherten Website erfolgt durch Angabe des Schemas `https` im URL. HTTPS-Verbindungen laufen über TCP, der Standard-Port für HTTPS-Verbindungen ist `443`. Für diese verschlüsselte Verbindung wird ein geheimer Schlüssel gebraucht. Es handelt sich um ein Schlüssel für symmetrische Verschlüsselung, d.h. beide Seiten der Verbindung müssen ihn kennen und sonst niemand!

Eine sichere Verbindung zwischen Browser und Server wird immer durch den Protokolldesignator ”https” angezeigt, z.B.

```
https://www.asw-berufsakademie
```

SSL führt vor dem Aufbau einer Verbindung eine Initialisierung durch das sog. *Handshake-Protokoll* vor. Dieses legt die Sicherheitsstufe fest, auf das sich der Client und der Server einigen, übernimmt die notwendigen Echtheitsbestätigungen für die Verbindung und handelt einen Sitzungsschlüssel (*Session Key*) für die Verschlüsselung aus. Dies geschieht wie folgt:

- ❒ Zu Beginn stellt der Client eine Anfrage an den Server und schickt ihm die Verschlüsselungsverfahren, die er unterstützt.
- ❒ Der Server wählt ein Verfahren aus und übermittelt dem Client sein Zertifikat mit dem öffentlichen Schlüssel des Servers.
- ❒ Anschliessend generiert der Client den sog. Sitzungsschlüssel (*Session Key*) für ein symmetrisches Verschlüsselungsverfahren. Dieser Session Key wird nun mit dem öffentlichen Schlüssel des Servers verschlüsselt und zum Server übertragen. Nur dieser kann ihn mit dem privaten Schlüssel wieder entschlüsseln.
- ❒ Optional kann nun eine Clientauthentifizierung erfolgen, die ähnlich der Serveridentifikation abläuft. Der Client muss jedoch über ein gültiges Zertifikat verfügen.

Da der Sitzungsschlüssel nun ausgetauscht ist, kann die Verschlüsselung beginnen. während die Verbindung besteht, übernimmt SSL lediglich die symmetrische Ver- und Entschlüsselung des Datenstroms mithilfe des Sitzungsschlüssels.

Da symmetrische Verschlüsselungsverfahren wesentlich schneller als asymmetrische Verfahren sind, können Daten schnell übertragen werden. Die asymmetrische Verschlüsselung wird nur zum Austausch des Sitzungsschlüssels und zur Authentisierung benutzt.

Sämtliche Informationen, sowohl die der Client-Anfrage als auch die der Server-Antwort, werden nun vollständig verschlüsselt. Dazu gehört auch der Pfad der angeforderten Webseite (URL), in Formularen übermittelte Informationen (z.B. Kreditkartennummer), die Informationen zur Echtheitsbestätigung (Benutzernamen und Kennwörter), sowie alle weiteren Daten, die sich Server und Client schicken.

Das alles passiert automatisch und transparent für die Benutzerin beim Handshake des Protokolls, bis auf die eine Entscheidung, sich zu verbinden oder nicht, in der Dialogbox Security Information. Nur wenn der Server völlig unbekannt ist und sein öffentlicher Schlüssel benötigt wird, muß der Benutzer eingreifen. Diese Technik ist zielführend, da die komplette asymmetrische Verschlüsselung aller Webseiten zu rechenintensiv wäre.

9.5 Host- und Client-Authentisierung

Verschlüsselung allein reicht jedoch nicht aus, um eine Verbindung sicher zu machen. Die Frage ist: Wer sitzt am anderen Ende? Der Benutzer kann sich ziemlich sicher sein, daß der Server der gemeinte ist, wenn sie beim Eintragen des öffentlichen Schlüssel des Servers aufgepaßt hat, diese Schlüssel nicht von einer unsicheren Quelle zu beziehen. Der Server jedoch hat keine Möglichkeit, zwischen verschiedenen Benutzern zu unterscheiden! Diese Art der Verbindung nennt sich deshalb https mit *Host Authentisierung*, weil nur der Host (der Server) seine Identität nachweisen muß, durch den Besitz des rechtmäßigen privaten Schlüssels, der zu dem des Benutzers bekannten öffentlichen Schlüssel paßt. Wenn sich auch der Benutzer ausweisen muß (z.B. beim Home-Banking), dann spricht man von *Client Authentisierung*. Bei einer Verbindung mit Client Authentisierung muß auch der Server sich auf sichere Weise den öffentlichen Schlüssel des Benutzers holen. Er kann aber nicht einfach den Benutzer selbst fragen, sonst ist die Sicherheit nicht gegeben, denn der Benutzer könnte sich für jemand anderes ausgeben und irgendeinen Schlüssel angeben, zu dem er den Privatschlüssel besitzt. Der öffentliche Schlüssel einer Person muß mit ihrer Identität offiziell und nachprüfbar verknüpft sein, sonst funktioniert die Authentisierung nicht.

9.6 x.509 Zertifikate

Die Garantie, daß ein öffentlicher Schlüssel einer bestimmten Person oder Entität gehört, wird im Internet durch ein Zertifikat geleistet. Der Internet-Standard X.509 beschreibt die Form eines Software-Zertifikats. X.509 ist ein Standardformat der ITU-T für Zertifikate. Es beinhaltet den Namen und die digitale Signatur des Ausstellers, sowie Informationen über die Identität des Inhabers. Auf dem X.509-Format basieren zum Beispiel SSL und S/MIME.

Das Zertifikat wird von einer Zertifizierungsstelle ausgestellt, die allgemein anerkannt ist. Sie verbürgt sich dafür, dass die Zuordnung eines Public Key's zu einer bestimmten Institution korrekt ist und dass die Institution, die den passenden privaten Schlüssel besitzt, tatsächlich existiert. Die erste international bekannte, in Netscape eingetragene Zertifizierungsstelle (auf Englisch: ***C**ertification **A**uthority* oder kurz *CA*) war VeriSign. Eine inzwischen recht bekannte Zertifizierungsstelle in Deutschland ist TC TrustCenter. Es ist die einzige deutsche CA, die in Netscape schon automatisch eingetragen ist. Es gibt

viele andere solche Stellen, bei denen der Benutzer jedoch selber die Eintragung machen muß, indem er die Zertifikate der CA aus einer sicheren Quelle holt.

9.6.1 x.509 Zertifikatsformat

Heutzutage wird das X.509-Zertifikat oft für die Internetnutzung, wie e-Commerce verwendet, und nicht mehr primär für den X.500 Verzeichnisdienst (deswegen wurden in der Version 3 auch die untenstehenden Erweiterungen übernommen). Wenn Alice eine Ware von Bob über das Internet kaufen möchte, aber diese Transaktionen über einen sicheren Weg vollziehen will, dann empfiehlt es sich, dies mittels Verschlüsselung und Authentifizierung zu machen. Bei einem Public Key-Verfahren ist es für Alice kein Problem, den öffentlichen Schlüssel von Bob zu bekommen; aber wie weiß Alice, daß Bob wirklich Bob ist? Dies ist die Aufgabe von Zertifikaten. Zertifikate binden die Identität, den Public-Key, und eine Menge anderer Information aneinander. Der schematische Aufbau eines Zertifikates kann auch aus Abb. 9.7abgelesen werden.

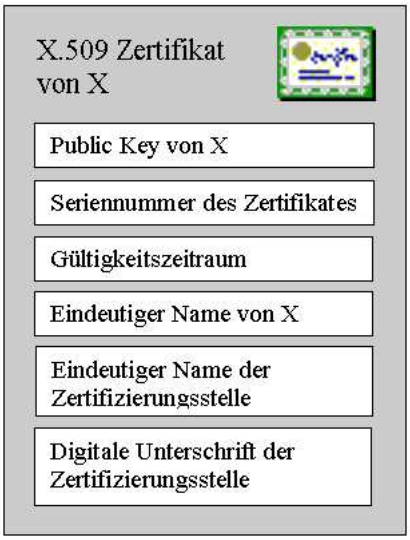

Abbildung 9.7: Wesentliche Inhalte eines X.509 Zertifikates

9.7 Erstellen von Zertifikaten

Das Erstellen eines Zertifikats lässt sich grob in drei Schritte gliedern. Zunächst muss ein Schlüsselpaar, d.h. ein privater und ein öffentlicher Schlüssel, erstellt werden. Der öffentliche Schlüssel wird anschließend zusammen mit anderen Informationen (Namen und anderen Parametern des Schlüsselinhabers) als s.g. ***C**ertificate **S**igning **R**equest* (CSR) an die *Certificate Authority* (=Zertifizierungsinstanz, CA) oder auch *Trust Center* genannt, geschickt. Aufgabe dieser CA ist es, durch geeignete Maßnahmen die Richtigkeit dieser Angaben zu überprüfen und den CSR mit der Signatur der CA zu bestätigen, sozusagen zu beglaubigen, und als fertiges Zertifikat zurück zum s.g. Antragsteller zuschicken.

Um zu so einem Zertifikat zu kommen gibt es zwei Möglichkeiten:

Wenn man es richtig machen will, was aber im Fall der nur internen Verwendung der SSL-Verschlüsselung nicht unbedingt notwendig ist, sollte man sich das Server-Zertifikat von einem offiziellen Trust-Center

wie verisign oder thawte signieren lassen. Die andere Möglichkeit ist, selber Trust-Center zu spielen. In diesem Fall muss ein weiteres Zertifikat für die Certificate Authority generiert werden, mit welcher dann das eigentliche Server-Zertifikat verifiziert werden kann. Der einzige Nachteil dieser Methode ist, das ein Browser, der eine gesicherte Verbindung zu so einem Server aufbaut, wegen der unbekannten Certificate Authority im signierenden Zertifikat erst mal eine Warnmeldung ausspucken wird.

9.7.1 Certificate Signing Request

Im nachfolgenden Teil werden die erforderlichen Schritte zur Key-Generierung und Installation eines eigenen Zertifikats genauer dargestellt:

Als erstes soll ein Server Key für den Apache Webserver erstellt werden (welcher TripleDES und PEM verschlüsselt sein wird):

```
$ openssl genrsa -des3 -out server.key 1024
```

Diese `server.key` Datei sollte auf alle Fälle gebackuped werden, auch das entsprechende Passwort sollte gut verwahrt werden. Wer sich für die Details dieses Keys interessiert kann diese mit dem folgenden Befehl ausgeben:

```
$ openssl rsa -noout -text -in server.key
```

Auch eine entschlüsselte PEM-Version des Schlüssel kann erstellt werden. Dies wird besonders im Zusammenhang mit dem automatischen Start des Servers wichtig. Allerdings sollte man sich darüber im klaren sein, welches Potential dieses entschlüsselte Format hat und entsprechende Vorrichtungen treffen, um einen Zugriff von unbefugten auf diese Datei zu unterbinden.

```
$ openssl rsa -in server.key -out server.key.unsecure
```

Als nächstes muss ein *Certificate Signing Request* (CSR) mit dem privaten Schlüssel des Servers erstellt werden. Die Ausgabe ist PEM-formatiert. PEM ist das unter Unix am gebraeuchlichsten Format fuer Schluessel und Zertifkat. PEM orientiert sich am "privacy enhanced mail" Format.

```
$ openssl req -new -key server.key -out server.csr
```

Im nächsten Schritt ist es von enormer Bedeutung, den korrekten FQDN (**F**ully **Q**ualified **D**omain **N**ame) des Servers einzugeben, wenn OpenSSL nach dem Common Name fragt. Der Common Name einer Webseite, welche über `https://www.foo.dom/` später angesprochen wird, ist zum Beispiel `www.foo.dom`. Auch die hier angemachten Angaben lassen sich später überprüfen, und zwar mit dem Befehl:

```
$ openssl req -noout -text -in server.csr
```

Sobald von der Certificate Authority eine Antwort eintrifft kann diese mit dem Befehl:

```
$ openssl x509 -noout -text -in server.crt
```

genauer betrachtet werden.

Nun existieren zwei Dateien: `server.key` und `server.crt`. Diese können nun in die Konfigurationsdatei `httpd.conf` des Apachen eingebunden werden, und zwar mit den folgenden beiden Optionen:

```
SSLCertificateFile /path/to/this/server.crt
SSLCertificateKeyFile /path/to/this/server.key
```

Will man sich zusätzlich die Eingabe des Passworts bei jedem Serverstart erfahren (z.B. weil der Server automatisch gestartet werden soll), sollte als `CertificateKeyFile` der entschlüsselte Server-Key angegeben werden. Die Datei `server.csr` wird nach der Ausstellung des Zertifikats nicht weiter benötigt!

9.7.2 Eigenes Trustcenter

In diesem Abschnitt soll nun auf das Erstellen eines eigenen Trust Centers genauer eingegangen werden. Obwohl im Normfall strenge Sicherheitsanforderungen an einen Trust Center gestellt werden, so erfüllt in den meisten Fällen auch eine eigen CA alle Anforderungen.

Als erstes muss wie im vorhergehenden Fall ein privater RSA-Key für das Trust Center generiert werden. (auch dieses wird TripleDES verschlüsselt sein und im PEM-Format vorliegen). Natürlich kann auch ein bestehendes CA-Zertifikat verwendet werden. In diesem Fall sind die nachfolgenden Schritte zu überspringen :

```
$ openssl genrsa -des3 -out ca.key 1024
```

Die Datei `ca.key` sollte gebackuped werden und das entsprechende Passwort dazu an einer sicheren Stelle aufbewahrt werden. Auch hier können die Details des Schlüssels wieder mit dem nachfolgenden Befehl ausgegeben werden.

```
$ openssl rsa -noout -text -in ca.key
```

Und auch hier kann wieder eine entschlüsselte Version des Keys erstellt werden, was aber im Zusammenhang mit einer Certificate Authority nicht unbedingt anzuraten ist.

```
$ openssl rsa -in ca.key -out ca.key.unsecure
```

Als nächstes muss ein selbstunterschriebens CA Certificate (im X509 Format) mit dem RSA Schlüssel der Certificate Authority erstellt werden. Auch hier ist die Ausgabe wieder PEM formatiert:

```
$ openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

Wie mittlerweile hinreichend bekannt sein sollte lassen sich auch hier wieder Details des Zertifikats anzeigen, und zwar mit dem Befehl

```
$ openssl x509 -noout -text -in ca.crt
```

Damit kann nun das Zertifikat der CA benutzt werden, um CSR's der Webserver zu unterschreiben, um echte SSL Zertifikate zu generieren.

Das Signieren eines einzelnen Requests erfolgt durch folgenden Befehl:

```
openssl ca -name ca_name -keyfile ca.key -in server.crt -out server.crt
```

Der Wert von `ca_name` steht hier für den gewünschten Abschnitt der Konfigurationsdatei von OpenSSL, der `openssl.cnf`, also z.B. `Client_CA`.

Nach Eingabe des Befehls kommt eine Meldung folgender Art:

```
Using configuration from /usr/local/ssl/lib/openssl.cnf
Enter PEM pass phrase:    "passphrase"
Check that the request matches the signature Signature ok
The Subjects Distinguished Name is as follows
countryName            :PRINTABLE:'DE'
stateOrProvinceName    :PRINTABLE:'Saarland'
localityName           :PRINTABLE:'St. Ingbert'
organizationName       :PRINTABLE:'ASW'
organizationalUnitName :PRINTABLE:'eBusiness'
commonName             :PRINTABLE:'Klara Otto'
emailAddress :IA5STRING:'k.otto@asw-berufsakademie'
Certificate is to be certified until
Mar 18 09:51:41 2003 GMT (365 days)
Sign the certificate? [y/n]: "y"

1 out of 1 certificate requests certified, commit? [y/n] "y"
Write out database with 1 new entries
Data Base Updated
```

Möglicherweise sind hierzu noch Anpassungen innerhalb der Konfigurationsdatei von OpenSSL zu machen. Nähere Informationen hierzu sind zu finden im OpenSSL Handbuch des DFN-CERT Zentrum für sichere Netzdienste GmbH unter `http://www.dfn-pca.de/certify/ssl/handbuch/ossl095/ossl095.html`.

10 Common Gateway Interface (CGI)

Das ***C**ommon **G**ateway **I**nterface* (CGI) ist eine Schnittstelle des Web-Servers. Sie erlaubt es, die Anfragen eines Web-Browsers an Programme (auf dem Web-Server) weiterzureichen und von diesen auszuführen zu lassen. Die Programme auf dem Server können dabei in verschiedene Programmiersparachen wie z.B. C, C++ oder Java oder auch spezielle Skriptsprachen wie Perl oder PHP geschrieben sein. Solche Programme (oder Scripts) können beispielsweise Formulareingaben aus HTML-Dateien verarbeiten, auf dem Server-Rechner Daten speichern und dort gespeicherte Daten auslesen. Auf diese Weise werden Web-Seiten zu Oberflächen für "Anwendungen", beispielsweise für elektronische Warenbestellung oder zum Abfragen von Datenbanken.

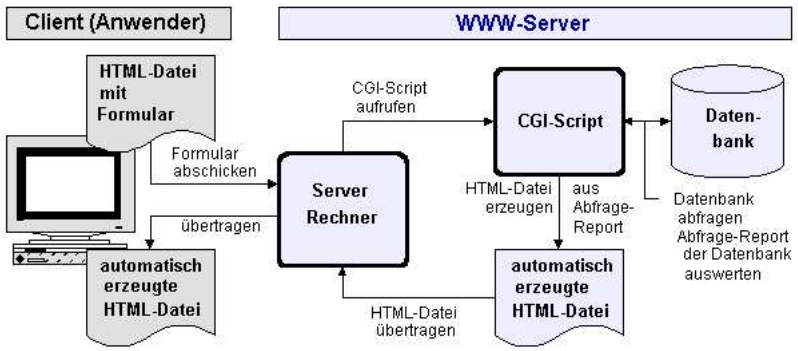

Abbildung 10.1: CGI

Die CGI-Schnittstelle ruft nicht nur das jeweils auszuführende Programm auf und leitet dessen Antwort weiter, sondern stellt auch eine Reihe von Daten bereit, die der Web-Server speichert, und die ein CGI-Script auslesen kann (und zum Teil auslesen muss), um Daten verarbeiten zu können. Diese Daten speichert der Web-Server in so genannten Seite CGI-Umgebungsvariablen.

Derzeit liegt die CGI-Schnittstellendefinition in der Version 1.1 vor. Andere Schnittstellen für ausführbare Programme im Web, die von kommerziellen Herstellern wie Netscape oder Microsoft eingeführt wurden, machen der klassischen CGI-Schnittstelle Konkurrenz. Von Netscape stammt beispielsweise die API-Schnittstelle (auch NSAPI genannt), von Microsoft die ISAPI-Schnittstelle. Beide Schnittstellen sind für die Server-Software-Produkte der jeweiligen Hersteller optimiert, erreichen allerdings auch eine Performance bei der Datenverarbeitung, die ein Mehrfaches der CGI-Schnittstelle beträgt. Ein entscheidender Vorteil der CGI-Schnittstelle bleibt jedoch die Tatsache, dass es sich - ähnlich wie bei HTML - um einen kommerziell unabhängigen, kostenlosen und produktübergreifenden Standard handelt.

Um CGI-Skripte auf eine der nachfolgend beschriebenen Arten zu benutzen, muss das Modul `mod_cgi` in den Apache eingebunden sein.

10.1 Verzeichnis als CGI-Verzeichnis deklarieren

Um die CGI-Schnittstelle nutzen zu können, benötigt der Client Zugriff auf ein bestimmtes Verzeichnis auf dem Server-Rechner, das CGI-Programme enthalten darf. Meist erhält dieses Verzeichnis den Namen `cgi-bin`. Die Skripte können dann über den URL `http://www.mydomain.de/cgi-bin/scriptname` angesprochen werden. Damit der Apache weiß, dass sich in diesem Verzeichnis CGI-Programme oder CGI-Scripts befinden, die beim Aufruf ausgeführt werden sollen, muss das Verzeichnis über die Konfigurationsanweisung `ScriptAlias` definiert werden, z.B.

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

Über `ScriptAlias` wird ein Verzeichnis angegeben, das ausschließlich CGI-Skripte enthält. Bei einem Zugriff auf eine Datei innerhalb dieses Verzeichnisses wird die Datei als CGI-Skript angesehen und ausgeführt, sofern es die Dateirechte zulassen.

Seit der Apache jedoch über die sogenannten Handler verfügt, braucht und sollte die `ScriptAlias`-Anweisung nicht mehr verwendet werden.

Der nach apache.org korrekte Weg, um ein Verzeichnis als CGI-Verzeichnis zu deklarieren, führt somit über das Setzen des Handlers[1] `cgi-script`. Der Handler wird über die Konfigurationsanweisung `SetHandler` aktiviert. Zusätzlich muss mit der `Options`-Anweisung die Ausführung von CGI-Skripten für dieses Verzeichnis erlaubt werden.

Beispiel:

```
<Directory /usr/local/apache/cgi-bin>
  AllowOverride None
  Options ExecCGI
  SetHandler cgi-script
</Directory>
```

10.1.1 Eigenes CGI-Verzeichnis für jeden Benutzer

Wie im Abschnitt 7.2 auf Seite 24 beschrieben, ist es möglich Benutzerverzeichnisse anzulegen, in denen UNIX Benutzer ihre Webseiten ablegen können. Um jedem Benutzer neben seiner `DocumentRoot` auch ein eigenes CGI-Verzeichnis zu geben, benutzt man die `<Directory>` Direktive, um ein Unterverzeichnis des Homeverzeichnisses cgi-fähig zu machen.

```
<Directory /home/*/cgi-bin/>
  Options ExecCGI
  SetHandler cgi-script
</Directory>
```

[1] Beim Apche ist es möglich, bei jedem Zugriff auf eine Datei mit einer best. Endung oder einer best. Lokation eine definierte Aktion ausführen zu lassen. Damit der Apache weiß. auf welche Dateien welche Aktion ausgeführt werden soll. wird zu diesem Zweck ein sog. Handler definiert.

10.2 Datei als CGI-Skript deklarieren

Analog zur zuvor beschriebenen Methode kann eine einzelne Datei als CGI-Skript deklariert werden.

Beispiel:

```
<Directory /usr/local/apache/htdocs/test>
  <Files script.sh>
    SetHandler cgi-script
  </Files>
</Directory>
```

Das CGI-Skript kann somit unter der URL `http://www.mydomain.de/test/script.sh` angesprochen werden und wird entsprechend ausgeführt. Da jedoch die `Options`-Anweisung nicht innerhalb einer `<Files>`-Sektion benutzt werden kann, muss sichergestellt werden, dasss `ExecCGI` in diesem Fall für das jeweilige Verzeichnis gesetzt ist.

10.3 Dateiendung zur Kennzeichnung von CGI-Skripten

Wenn eine generelle Nutzung von CGI-Skripten in den normalen Verzeichnissen gewünscht ist, empfiehlt es sich, den `cgi-script`-Handler für eine bestimmte Dateiendung zu setzten.

Beispiel:

```
AddHandler cgi-script cgi
AddHandler cgi-script pl
```

Ist der Handler aktiviert, können in allen Verzeichnissen, für die die Option `ExecCGI` gesetzt ist, CGI-Skripte ausgeführt werden, sofern sie die jeweils deklarierte Dateiendung (hier .cgi und .pl) haben.

11 Logdateien

Um einen Überblick über die Funktion und Auslastung des Servers zu bekommen, kann der Apache verschieden Logfiles führen. In erster Linie sind hier das Error-Log und das Access-Log zu nennen. Ein Web-Server protokolliert jeden an ihn gerichteten Request in einer AccessLog-Datei. Falls bei der Bearbeitung oder Beantwortung Fehler auftreten, werden diese in der ErrorLog-Datei registriert. Falls ein Server auch virtuelle Server unterstüzt, so können für jeden virtuellen Server eigene Access- und ErrorLog-Dateien geführt werden, was den Verwaltungsaufwand erheblich erleichtert. Insbesondere das Access-Log ist für einen Betreiber eines Webservers interessant, da hiermit ein Überblick über die Auslastung des Servers verschafft werden kann[1].

11.1 Error Log

Über dieKonfigurationsanweisung `ErrorLog` in der Konfigurationsdatei wird dem Apache mitgeteilt, in welche Datei er seine Fehlermeldungen schreiben soll. Eintragungen in einer `ErrorLog`-Datei enthalten Informationen über Fehler, die während des Betriebs des Servers beim Bearbeiten oder Beantworten von Anfragen erfolgen, oder allgemeine Informationen über den Betrieb des Servers.

Beispiele für ErrorLog-Einträge sind folgende:

```
[Tue Aug 5 11:52:51 1997] access to /users/wiss/data/hauck.gif
failed for 80.131.4.23, reason: File does not exist
```

oder

```
[Tue Aug 5 16:15:05 1997] access to /users/wiss/data failed for
delphin.pillipalli.de, reason: Client denied by server configuration
```

11.2 Access Log

Die Informationen in der AccessLog-Datei werden per Default im ***C**ommon **L**og **F**ormat* (CLF)[2] für Log-Dateien abgespeichert:

```
host rfc931 authuser [date/time] "request" status bytes
```

Die einzelnen Werte bedeuten folgendes:

[1] Zur Auswertung der Access-Log gibt es spezielle Tools, wie z.B. den Webalizer (http://www.webalizer.org/)
[2] http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format

- **host** An dieser Stelle wird die IP-Adresse des abrufenden Rechners gespeichert. Je nach Serverkonfiguration wird auch der DNS-Name zu dieser IP aufgelöst (Ident), wodurch sich der Zugangsanbieter (Provider, Firma, Universität,...) und oft auch das Herkunftsgebiet ermitteln läßt. In Verbindung mit der Abrufzeit und der jeweiligen Anfrage kann man hiermit den Ablauf eines Seitenbesuches nachvollziehen.

- **[date/time]** Datum und Uhrzeit, wann die Anfrage ankam. Die einzelnen Einträge liegen zwar schon in chronologischer Reihenfolge vor, der genaue Zeitpunkt der Anfrage wird jedoch explizit gespeichert. Anhand der Verteilung der Abrufzeiten kann man erkennen, wann die Seite bevorzugt besucht wird (und dementsprechend daraus schließen, zu welchem Anteil Heimanwender, Werktätige und Schüler vertreten sind). Ebenso kann verfolgt werden, wie lange auf einer Seite verweilt wird und wie lange der Besuch insgesamt dauert. Hinweise darauf, wie schnell einzelne Dateien übertragen werden und ob der Browser mehrere Dateien parallel lädt kann man ebenfalls gewinnen, allerdings nur mit sehr begrenzter Genauigkeit.

- **request** Der request-Eintrag, der protokolliert wird, kann in weitere Einheiten aufgeteilt und entsprechend untersucht werden. Dies ist z.B. dann notwendig, falls ein Fehler bei der Suche nach einer Datei aufgetreten oder ein unerlaubter Zugriff auf eine Datei erfolgt ist. So kann z.B. erkannt werden, daß bestimmte Dateien immer wieder bestimmte Fehler hervorrufen. Ein request-Eintrag kann also in folgende Angaben unterteilt und untersucht werden:

  - **HTTP-Version** Die verwendete Version des HTTP-Protokolls ist relativ uninteressant, da nur einige alte Browser ausschließlich HTTP 1.0 können. Gegebenfalls kann man hier jedoch erkennen, ob es mit bestimmten Protokollversionen Probleme gibt.
  - **HTTP-Methode**, mit der auf die Datei zugegriffen wird (GET, HEAD, PUT, POST, DELETE). Per GET oder POST werden die Dateien normal geladen, während mit HEAD nur Informationen über die Datei abgefragt werden. Letzteres ist typisch für Suchdienste, Browser und Downloadprogramme, die die Dateien schon bei einem vorhergehenden Besuch geladen haben und nur prüfen, ob die Datei sich in der Zwischenzeit geändert hat. Auch Programme und Onlinedienste, die Webseiten auf Aktualisierungen überprüfen, verraten sich durch HEAD-Anfragen.
  - **URL** der abgerufenen Datei. Daß sich hiermit die Abfragehäufigkeit und somit auch die Begehrtheit einer Seite ermitteln läßt, dürfte jedem offensichtlich sein. Aber der Dateiname verrät indirekt noch mehr Informationen: Daran, daß nach dem Abruf einer Datei die darin eingebundenen Bilder, Frames, Stylesheets, Skripte, Applets oder per Object-Tag eingebundene Bestandteile abgerufen werden oder nicht, erkennt man, ob diese vom Besucher überhaupt verwendet wurden. Wird nur ein Teil dieser Daten geladen, können Verbindungsprobleme, evtl. durch zu große Dateien, die Ursache dafür sein.

- **status** HTTP Status Code, der an den Client nach Bearbeiten der Anfrage zurückgegeben wurde.

- **bytes** (ohne Header) Primär dient die Angabe der übertragenen Daten zur Ermittlung des übermittelten Datenvolumens als auch die Spitzenzeiten der Serverauslastung. Weicht die Angabe jedoch von der Größe der übermittelten Datei ab, so deutet dies entweder auf Verbindungsabbrüche oder auf Downloadclients mit mehreren Verbindungen bzw. nach einem Abbruch fortgesetzte (=Resume) Übertragungen hin.

Das Format der Eintragungen in den Log-Dateien kann nach Belieben mit Hilfe von festgelegten Formatanweisungen an eigene Bedürfnisse angepaßt werden, die hier aber nicht weiter erläutert werden. Über die Konfigurationsanweisung LogFormat innerhalb der Serverkonfiguration wird bestimmt, welche Werte in dem mit CustomLog spezifizierten Logfile protokolliert werden. So protokollieren manche Server auch den Referrer und den Namen und Version des zugreifenden Browsers.

- **Referrer** In der Referrer-Angabe wird die Adresse der zuvor besuchten Seite gespeichert. Dadurch läßt sich die Reihenfolge der besuchten Seiten nachvollziehen als auch die vorher besuchte Seite feststellen. Auf diese Weise findet man externe Links, die nicht nur von anderen Webseiten, sondern auch von Bookmarks oder auf dem Rechner des Besuchers gespeicherten Seiten stammen können. Die Häufigkeit, mit der ein Referrer vorkommt, ist auch ein gutes Indiz für die Verlinkung und die Popularität einer Seite. Externe Referrer finden sich aber auch, wenn andere Seiten unerlaubterweise eigene Inhalte in Frames einbinden oder Bilder und Downloads ohne Genehmigung verlinken.

  Suchdienste finden sich ebenfalls im Referrer. Üblicherweise werden die Suchbegriffe als auch die Positionierung des Suchergebnisses als GET-Variablen im Referrer mitgespeichert. Man erkennt dadurch, ob man überhaupt durch Suchdienste gefunden wird und ob die Seite gut zu der jeweiligen Anfrage paßt oder nicht. Je nach Bedarf sollte man seine Stichwörter anpassen oder - wenn eine Seite nicht gefunden werden soll - die robots.txt anpassen.

  Referrer von anderen Seiten verraten zudem, welche Sprachen die Besucher beherrschen und aus welcher Gegend sie stammen., von dem aus der Zugriff erfolgt.

- **Browserkennung** Neben der Bezeichung des verwendeten Browsers und dessen Programmversion finden sich hier meist weitere Informationen, etwa zur verwendeten Sprachversion, des verwendeten Betriebssystems (teilweise mit Angabe der jeweiligen Version) oder zu angepaßten Browserversionen, wie sie von einigen Firmen und Providern verbreitet werden.

  Suchmaschinen und Downloadprogramme verwenden üblicherweise eigene Kennungen, die oft auch die URL des Anbieters enthalten.

Ein Beispieleintrag in der AccessLog-Datei würde folgendermaßen aussehen:

```
10.1.1.240 - - [10/Apr/2003:10:05:17 +0200] "GET
/apache.html HTTP/1.1" 404 1170 "-"
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
```

oder

```
myserver.mydomain.tld - - [08/Oct/1997:10:44:51 +0200]
"GET /htdocs/MNMPub/index.shtml HTTP/1.0" 200 3094
```

12 Beispielkonfiguration Apache 1.3 mit Kommentaren

```
###############################################################
#
# httpd.conf - die Konfigurationsdatei für den Apache-Webserver
# Version 1.3.19-48
#
# Einsatzplattform LINUX
#
###############################################################
###############################################################
#
# Bei dieser Fassung wurden die Kommentare ins Deutsche übersetzt
# und geringfügig ergänzt bzw. erweitert.
#
# Walter Neu
#
###############################################################
###############################################################
#
# Einleitung
# ----------
#
# Diese Datei basiert auf den originalen Angaben von Rob McCool, mit denen
# der NCSA-Server konfiguriert wurde (NCSA = National Center for Supercomputing
# Applications, University of Illinois, Urbana-Champaign).
#
# Es handelt sich hier um die zentrale Konfigurationsdatei für den Apache.
# Sie enthält Konfigurations-Anweisungen, mit denen der Server seine Instruktionen
# bekommt (siehe <http://www.apache.org/docs/> für detailliertere Informationen).
#
# Sie sollten diese Anweisungen nicht einfach nur lesen, ohne genau zu verstehen,
# was sie bewirken. Sie stehen hier eigentlich auch nur als kurze Notizen zur
# Erinnerung. Wenn Sie sich unsicher fühlen, sollten Sie die Online-Dokumentation
# gründlich studieren.
#
#
# Wenn diese Datei abgearbeitet worden ist, sucht der Server nach den Dateien
# srm.conf und access.conf, falls Sie sie hier nicht mit den Einstellungen für
# ResourceConfig und/oder AccessConfig überschrieben haben.
#
# Die Anweisungen zur Konfiguration sind in drei Haupt-Abschnitte gegliedert:
# 1. Anweisungen, die die gesamten Verhaltensweisen des Apache Web Servers festlegen
# (die "globale Umgebung").
# 2. Anweisungen, die die Parameter für den Standard-Server definieren, der alle die
# Anfragen bearbeitet, die nicht von virtuellen Hosts entgegengenommen werden.
# Diese Angaben gelten auch als Standardwerte für virtuelle Hosts, sofern sie nicht
# in den entsprechenden Anweisungen dort überschrieben werden.
# 3. Festlegungen für virtuelle Hosts. Damit werden Web-Anfragen für verschiedene
# IP-Adressen oder Hostnamen ermöglicht, die jeweils von demselben Server-Prozeß
# behandelt werden sollen.
#
```

```
# Namen für Konfigurations- und Protokolldateien: Wenn die von Ihnen angegebenen
# Dateinamen für viele der Kontrolldateien mit einem Slash ( "/" ) beginnen, wird
# der Server das als den ausdrücklich zugewiesenen Pfad ansehen. Sofern die
# Dateinamen NICHT mit "/" beginnen, wird das ServerRoot-Verzeichnis
# davorgeschrieben.
# So wird zum Beispiel "logs/foo.log" als "/usr/local/apache/logs/foo.log"
# interpretiert, wenn das "ServerRoot"-Verzeichnis mit "/usr/local/apache"
# zuvor festgelegt wurde.
#
########################################################################
########################################################################
#
# 1. Abschnitt: Globale Umgebung
# ------------------------------
#
# Die Anweisungen in diesem Abschnitt bestimmen die grundsätzliche Arbeitsweise
# Ihres Apache-Servers.
#
#
# "ServerType": Der Server-Typ ist entweder "inetd" oder "standalone".
# "inetd" wird allerdings nur von UNIX-Plattformen unterstützt/verwendet.
#
ServerType inetd
#
# "ServerRoot" bezeichnet die Spitze des Verzeichnisbaums, unter dem die
# Server-Konfiguration, Fehleranzeigen und Protokolle zu finden sind.
#
# ACHTUNG!
# Falls Sie vorhaben, dieses Verzeichnis auf einem mit Hilfe des Netzwerks
# eingebundenen Dateisystem (NFS oder anderes) abzulegen, sollten Sie
# unbedingt die Dokumentation zu "LockFile" studieren (verfügbar unter
# http://www.apache.org/docs/mod/core.html#lockfile); Sie ersparen sich
# damit eine Menge Ärger.
#
# Am Ende dieser Zeile DARF KEIN Slash stehen.
#
ServerRoot "/usr/local/httpd"
#
# "LockFile" bestimmt den Pfad zur lock-Datei, die dann benötigt wird, wenn
# Apache mit USE_FCNTL_SERIALIZED_ACCEPT oder USE_FLOCK_SERIALIZED_ACCEPT
# kompiliert wurde. Diese Anweisung sollte normalerweise ihren Standardwert
# behalten. Ein Grund, das abzuändern, liegt dann vor, wenn das Protokoll-
# verzeichnis (/logs) über einen Netzwerkpfad gemountet wird. Wenn das nicht
# der Fall ist, MUSS die lock-Datei auf einer lokalen Partition abgelegt
# werden. Die PID des Server-Hauptprozesses wird automatisch mit dem
# Dateinamen verbunden.
#
LockFile /var/lock/subsys/httpd/httpd.accept.lock
#
# "PidFile": das ist die Datei, in der der Server jedesmal dann seine
# "process identification number" niederlegen sollte, wenn er gestartet
# wird.
#
PidFile /var/run/httpd.pid
#
# "ScoreBoardFile": die Datei, die interne Informationen über Server-Prozesse
# aufnimmt. Nicht alle Architekturen verlangen eine solche Datei. Aber wenn Ihre
# Architektur sie vorsieht (Sie erfahren das, sobald Sie Apache starten, womit
# die Datei erforderlichenfalls generiert wird), müssen Sie dafür sorgen,
# daß zwei (oder mehr) Apache-Aufrufe nicht auf dieselbe Scoreboard-Datei
# zugreifen.
#
ScoreBoardFile /var/run/httpd.scoreboard
#
```

```
# In der Standard-Konfiguration wird der Server die Dateien httpd.conf (diese
# Datei), srm.conf, und access.conf in dieser Reihenfolge abarbeiten.
# Die beiden letztgenannten Dateien werden jetzt leer belassen, da empfohlen
# wird, zur besseren übersicht sämtliche Anweisungen in einer einzigen
# Konfigurationsdatei zusammenzufassen.
# Die auskommentierten Werte unten sind jeweils die vorgegebenen
# Standardwerte. Sie können festlegen, daß der Server diese Dateien
# ignoriert, indem Sie den Anweisungen den Parameter "/dev/null" mit-
# geben.
#
#ResourceConfig conf/srm.conf
#AccessConfig conf/access.conf
#
# "Timeout" legt die Sekundenanzahl fest, ehe ein Timeout (eine lauf-
# zeitbedingte Unterbrechung) gesendet wird.
#
Timeout 300
#
# "KeepAlive": legt fest, ob persistente Verbindungen (mehr als eine Anfrage
# pro Verbindung) zulässig sind. Wird mit "Off" deaktiviert
#
KeepAlive On
#
# "MaxKeepAliveRequests": die Höchstzahl der während einer persistenten Verbindung
# zulässigen Anfragen. Wird hier 0 angegeben, ist unbegrenzter Zugriff möglich.
# Empfohlen wird ein hoher Wert, um eine hohe Performance zu erhalten.
#
MaxKeepAliveRequests 200
#
# "KeepAliveTimeout": Zeitspanne (Zahl an Sekunden), um auf die nächste Abfrage
# desselben Clients mit derselben Verbindung zu warten.
#
KeepAliveTimeout 15
#
# Größenanpassung des Server-Pools: Ähnlich wie Sie selbst abschätzen würden, wieviele
# Server-Prozesse Sie brauchen, paßt sich Apache dynamisch der Belastung an, die er
# erkennt - das bedeutet, er versucht, genügend Server-Prozesse zur Bewältigung der
# aktuellen Serverlast bereitzustellen und zusätzlich noch ein paar Ersatz-Server, damit
# kurzzeitige Spitzenbelastungen (z.B. multiple simultane Requests von einem einzelnen
# Netscape-Browser) bewältigt werden können.
#
# Das verläuft so, daß periodisch abgefragt wird, wieviele Server gerade auf eine Rück-
# meldung warten. Wenn das weniger sind als in "MinSpareServers" vorgegeben,
# wird ein neuer erstellt. Sind das mehr als in "MaxSpareServers" vorgegeben, werden
# einige beendet (sie "sterben"). Die Standardvorgaben sind vermutlich für die meisten
# Seiten so in Ordnung.
#
MinSpareServers 1
MaxSpareServers 1
#
# "StartServers": Anzahl der Server, die zu Beginn gestartet werden. Das sollte eine
# vernünftige Vorgabe sein.
#
StartServers 5
#
# "MaxClients": Höchstzahl der gleichzeitig laufenden Server. Damit wird auch die Zahl
# der Clients eingegrenzt, die simultan mit dem Server verbunden werden können. Wenn
# diese Zahl erreicht ist, werden weitere Clients ausgeschlossen; also sollte Ihre Vorgabe
# kein zu kleiner Wert sein.
# Diese Anweisung soll vor allem verhindern, daß ein "durchdrehender" Serverprozeß
# das gesamte System mitnimmt, wenn er sich beendet.
#
MaxClients 150
#
```

```
# "MaxRequestsPerChild": Höchstzahl an Anfragen, die jeder "Kind-Prozeß" (der Webserever
# startet bei Anfragen solche "Kind-Prozesse", in denen die Threads ablaufen können)
# behandeln darf, ehe er beendet wird. Das Beenden von "Kind-Prozessen" ist wichtig, um
# Probleme zu vermeiden, falls Apache (und möglicherwiese die von ihm verwendeten
# Bibliotheken) den Speicher oder andere Ressourcen aufgebraucht hat. Auf den meisten
# Plattformen wird das zwar kaum einmal vorkommen, aber ein paar, wie zum Beispiel Solaris,
# verfügen über beachtliche "Lecks" in ihren Bibliotheken. Für solche Plattformen
# sollten Sie hier 10 000 oder einen ähnlichen Wert festlegen. Wenn Sie 0 (Null) vorgeben,
# ist die Anzahl unbegrenzt.
#
# Beachten Sie: mit Ausnahme der Initial-Anfrage gilt dieser Wert NICHT für
# "Keepalive"-Anfragen. Wenn ein "Kind-Prozeß" beispielsweise mit einer
# Initial-Anfrage startet, der weitere zehn Anfragen folgen, so würde hier
# lediglich 1 gezählt werden.
#
MaxRequestsPerChild 0
#
# "Listen": Diese Anweisung gestattet es, Apache mit spezifischen IP-Adressen
# und/oder Ports zu verbinden, zusätzlich zu den Standardvorgaben. Bitte auch
# Abschnitt 3 ( <VirtualHost> ) vergleichen.
#
#Listen 3000
#Listen 192.168.0.1:80
#
# "BindAddress": mit dieser Option können virtuelle Hosts unterstützt werden.
# Die Anweisung wird genutzt, um dem Server mitzuteilen, welche IP er wählen
# soll. Sie kann entweder einen Asterisk (*) oder einen vollständigen
# Internetnamen enthalten.
# Siehe auch die Anweisungen "<VirtualHost>" und "Listen".
#
BindAddress *
#
# Die folgenden Module werden geladen, wenn das zugehörige Paket auf Ihrem
# System installiert ist. Die entsprechenden Variablen werden in /sbin/init.d/apache
# definiert und kontrollieren, welche Module dynamisch geladen werden.
#
<IfDefine PHP>
LoadModule php3_module /usr/lib/apache/libphp3.so
</IfDefine>
<IfDefine PHP4>
LoadModule php4_module /usr/lib/apache/libphp4.so
</IfDefine>
<IfDefine PERL >
LoadModule perl_module /usr/lib/apache/libperl.so
</IfDefine>
<IfDefine DA V>
LoadModule dav_module /usr/lib/apache/libdav.so
</IfDefine>
<IfDefine BACKHAND>
LoadModule backhand_module /usr/lib/apache/mod_backhand.so
</IfDefine>
<IfDefine SSL>
LoadModule ssl_module /usr/lib/apache/libssl.so
LoadModule sxnet_module /usr/lib/apache/mod_sxnet.so
</IfDefine>
<IfDefine SAP_CGI>
LoadModule fastcgi_module /usr/lib/apache/mod_fastcgi_sap.so
</IfDefine>
<IfDefine MODULES>
# LoadModule allowdev_module /usr/lib/apache/mod_allowdev.so
LoadModule cookie_auth_module /usr/lib/apache/mod_auth_cookie.so
LoadModule cookie_file_access_module /usr/lib/apache/mod_auth_cookie_file.so
LoadModule external_auth_module /usr/lib/apache/mod_auth_external.so
LoadModule inst_auth_module /usr/lib/apache/mod_auth_inst.so
```

```
LoadModule auth_system_module /usr/lib/apache/mod_auth_system.so
LoadModule eaccess_module /usr/lib/apache/mod_eaccess.so
LoadModule bandwidth_module /usr/lib/apache/mod_bandwidth.so
LoadModule cache_module /usr/lib/apache/libcache.so
LoadModule urlcount_module /usr/lib/apache/mod_urlcount.so
LoadModule disallow_id_module /usr/lib/apache/mod_disallow_id.so
LoadModule lock_module /usr/lib/apache/mod_lock.so
LoadModule peephole_module /usr/lib/apache/mod_peephole.so
LoadModule put_module /usr/lib/apache/mod_put.so
LoadModule qs2ssi_module /usr/lib/apache/mod_qs2ssi.so
LoadModule session_module /usr/lib/apache/mod_session.so
LoadModule fastcgi_module /usr/lib/apache/mod_fastcgi.so
LoadModule cvs_module /usr/lib/apache/mod_cvs.so
LoadModule roaming_module /usr/lib/apache/mod_roaming.so
LoadModule w3c_ip_forwarding_module /usr/lib/apache/mod_ip_forwarding.so
LoadModule macro_module /usr/lib/apache/mod_macro.so
LoadModule layout_module /usr/lib/apache/mod_layout.so
LoadModule cgisock_module /usr/lib/apache/mod_cgisock.so
LoadModule ticket_module /usr/lib/apache/mod_ticket.so
LoadModule random_module /usr/lib/apache/mod_random.so
LoadModule mysql_auth_module /usr/lib/apache/mod_auth_mysql.so
LoadModule dynvhost_module /usr/lib/apache/mod_dynvhost.so
LoadModule gzip_module /usr/lib/apache/mod_gzip.so
LoadModule throttle_module /usr/lib/apache/mod_throttle.so
</IfDefine>
<IfDefine LDAP>
LoadModule auth_ldap_module /usr/lib/apache/auth_ldap.so
</IfDefine>
<IfDefine PYTHON>
LoadModule python_module /usr/lib/apache/mod_python.so
</IfDefine>
#
# Unterstützung für dynamisch verbundene Objekte (Module)
#
# Um die Funktionalität eines als dynamisch verbundenes Ojekt eingebundenen Moduls
# nutzen zu können, muß die korrespondierende Zeile 'LoadModule' hier eingetragen
# sein, damit die in diesem Modul vorhandenen Anweisungen aktuell verfügbar sind,
# BEVOR sie aufgerufen werden. Bitte lesen Sie in der Datei README.DSO Ihrer
# Apache-Distribution nach, wenn Sie Genaueres über das Konzept der dynamisch
# verbundenen Objekte (DSO) erfahren möchten, und lassen Sie sich mit 'httpd -l'
# auf der Befehlszeile eine Liste der aktuell verfügbaren statisch eingebundenen
# Module Ihres httpd-Verzeichnisses anzeigen.
#
# Beachten Sie: die Reihenfolge, in der die Module geladen werden, ist wichtig.
# Wenn es nicht absolut nötig ist oder wenn Sie nicht absolut präzise wissen,
# was Sie ändern möchten, sollten Sie auch diese Liste unverändert lassen.
#
LoadModule mmap_static_module /usr/lib/apache/mod_mmap_static.so
LoadModule vhost_alias_module /usr/lib/apache/mod_vhost_alias.so
LoadModule env_module /usr/lib/apache/mod_env.so
LoadModule define_module /usr/lib/apache/mod_define.so
LoadModule config_log_module /usr/lib/apache/mod_log_config.so
LoadModule agent_log_module /usr/lib/apache/mod_log_agent.so
LoadModule referer_log_module /usr/lib/apache/mod_log_referer.so
LoadModule mime_magic_module /usr/lib/apache/mod_mime_magic.so
LoadModule mime_module /usr/lib/apache/mod_mime.so
LoadModule negotiation_module /usr/lib/apache/mod_negotiation.so
LoadModule status_module /usr/lib/apache/mod_status.so
LoadModule info_module /usr/lib/apache/mod_info.so
LoadModule includes_module /usr/lib/apache/mod_include.so
LoadModule autoindex_module /usr/lib/apache/mod_autoindex.so
LoadModule dir_module /usr/lib/apache/mod_dir.so
LoadModule cgi_module /usr/lib/apache/mod_cgi.so
LoadModule asis_module /usr/lib/apache/mod_asis.so
```

```
LoadModule imap_module /usr/lib/apache/mod_imap.so
LoadModule action_module /usr/lib/apache/mod_actions.so
LoadModule speling_module /usr/lib/apache/mod_speling.so
LoadModule userdir_module /usr/lib/apache/mod_userdir.so
LoadModule alias_module /usr/lib/apache/mod_alias.so
LoadModule rewrite_module /usr/lib/apache/mod_rewrite.so
LoadModule access_module /usr/lib/apache/mod_access.so
LoadModule auth_module /usr/lib/apache/mod_auth.so
LoadModule anon_auth_module /usr/lib/apache/mod_auth_anon.so
LoadModule dbm_auth_module /usr/lib/apache/mod_auth_dbm.so
LoadModule db_auth_module /usr/lib/apache/mod_auth_db.so
LoadModule digest_module /usr/lib/apache/mod_digest.so
LoadModule proxy_module /usr/lib/apache/libproxy.so
LoadModule cern_meta_module /usr/lib/apache/mod_cern_meta.so
LoadModule expires_module /usr/lib/apache/mod_expires.so
LoadModule headers_module /usr/lib/apache/mod_headers.so
LoadModule usertrack_module /usr/lib/apache/mod_usertrack.so
LoadModule unique_id_module /usr/lib/apache/mod_unique_id.so
LoadModule setenvif_module /usr/lib/apache/mod_setenvif.so
<IfDefine DUMMYSSL>
LoadModule ssl_module /usr/lib/apache/libssl.so
</IfDefine>
# Wiederherstellung der kompletten Liste sämtlicher verfügbaren Module
# (statisch und dynamisch verbunden), um ihre korrekte Abarbeitung
# erreichen zu können.
#
# ACHTUNG !! Falls Sie die Liste unter "LoadModule" oben verändert haben,
# müssen Sie dieselben Korrekturen auch hier vornehmen.
#
ClearModuleList
AddModule mod_mmap_static.c
AddModule mod_vhost_alias.c
AddModule mod_env.c
AddModule mod_define.c
AddModule mod_log_config.c
AddModule mod_log_agent.c
AddModule mod_log_referer.c
AddModule mod_mime_magic.c
AddModule mod_mime.c
AddModule mod_negotiation.c
AddModule mod_status.c
AddModule mod_info.c
AddModule mod_include.c
AddModule mod_autoindex.c
AddModule mod_dir.c
AddModule mod_cgi.c
AddModule mod_asis.c
AddModule mod_imap.c
AddModule mod_actions.c
AddModule mod_speling.c
AddModule mod_userdir.c
AddModule mod_alias.c
AddModule mod_rewrite.c
AddModule mod_access.c
AddModule mod_auth.c
AddModule mod_auth_anon.c
AddModule mod_auth_dbm.c
AddModule mod_auth_db.c
AddModule mod_digest.c
AddModule mod_proxy.c
AddModule mod_cern_meta.c
AddModule mod_expires.c
AddModule mod_headers.c
AddModule mod_usertrack.c
```

```
AddModule mod_unique_id.c
AddModule mod_so.c
AddModule mod_setenvif.c
<IfDefine DUMMYSSL>
AddModule mod_ssl.c
</IfDefine>
#
# Nochmals: die folgenden Module werden nur geladen, wenn die zugehörigen Pakete
# auf Ihrem Rechner installiert wurden.
#
<IfDefine PHP>
AddModule mod_php3.c
</IfDefine>
<IfDefine PHP4>
AddModule mod_php4.c
</IfDefine>
<IfDefine PERL>
AddModule mod_perl.c
</IfDefine>
<IfDefine DAV>
AddModule mod_dav.c
</IfDefine>
<IfDefine BACKHAND>
AddModule mod_backhand.c
</IfDefine>
<IfDefine SSL>
AddModule mod_ssl.c
AddModule mod_sxnet.c
</IfDefine>
<IfDefine SAP_CGI>
AddModule mod_fastcgi.c
</IfDefine>
<IfDefine MODULES>
# AddModule mod_allowdev.c
AddModule mod_auth_cookie.c
AddModule mod_auth_cookie_file.c
AddModule mod_auth_external.c
AddModule mod_auth_inst.c
AddModule mod_auth_system.c
AddModule mod_eaccess.c
AddModule mod_bandwidth.c
AddModule mod_cache.c
AddModule mod_urlcount.c
AddModule mod_disallow_id.c
AddModule mod_lock.c
AddModule mod_peephole.c
AddModule mod_put.c
AddModule mod_qs2ssi.c
AddModule mod_session.c
AddModule mod_fastcgi.c
AddModule mod_cvs.c
AddModule mod_roaming.c
AddModule mod_ip_forwarding.c
AddModule mod_macro.c
AddModule mod_auth_mysql.c
AddModule mod_layout.c
AddModule mod_cgisock.c
AddModule mod_ticket.c
AddModule mod_random.c
AddModule mod_dynvhost.c
AddModule mod_gzip.c
AddModule mod_throttle.c
</IfDefine>
<IfDefine LDAP>
```

```
AddModule auth_ldap.c
</IfDefine>
<IfDefine PYTHON>
AddModule mod_python.c
</IfDefine>
#
# "ExtendedStatus" legt fest, ob Apache eine vollständige Status-Information (On)
# oder nur die Basis-Informationen ausgeben soll, wenn der "server-status" im
# Browser abgerufen wird. Der Standard ist Off.
#
ExtendedStatus On
#
# Wenn Sie mod_dav zulassen wollen, fügen Sie die folgende Anweisung in
# den/die zugehörigen Container der httpd.conf ein:
#
<IfModule mod_dav.c>
DavLockDB /var/lock/DAVLock
</IfModule>
#
# Wenn Sie Server Side Includes verwenden möchten:
#
<IfModule mod_include.c>
XBitHack on
</IfModule>
#
######################################################################
######################################################################
#
# 2. Abschnitt: Generelle Server-Konfiguration
# --------------------------------------------
#
# Die Anweisungen in diesem Abschnitt legen die generell vom Server benötigten
# Werte fest, mit denen auf alle Anfragen reagiert wird, die nicht von virtuellen
# Hosts abgearbeitet werden.
# Alle diese Anweisungen können auch innerhalb von <VirtualHost>-Containern
# stehen, womit dann die hier vorgenommenen Standard-Einstellungen für den
# jeweiligen virtuellen Host überschrieben werden.
#
#
# Falls Ihre Festlegung zu "ServerType" (weiter oben im Abschnitt
# "Globale Umgebung") "inetd" lautet, haben die nächsten paar Anweisungen
# hier keinerlei Auswirkungen, da ihre Wirksamkeit bzw. Nichtwirksamket
# mit der Konfiguration für inetd definiert wird. übergehen Sie in diesem
# Fall die folgenden Schritte bis zu den Festlegungen für "ServerAdmin".
#
#---------------------------------------------------------------
#
# "Port": der port, den der Server benutzt. Manche Firewalls müssen korrekt
# konfiguriert sein, wenn Apache auf einen bestimmten Port zugreifen soll.
#
# Für niedrigere ports als 1023 müssen Sie Apache zunächst als "root" starten.
#
Port 80
#
# SSL-Unterstützung (Secure Socket Layer)
#
# Wenn auch für SSL gesorgt werden soll, muß der Standard-HTTP-Port ebenso
# genutzt werden können wie der Standard-HTTPS-Port.
#
<IfDefine SSL>
Listen 80
Listen 443
</IfDefine>
#
```

```
# Wenn ein bestimmter Benutzer bzw. eine bestimmte Gruppe den Apache
# Webserver starten soll, müssen Sie zunächst als "root" beginnen; Apache
# schaltet dann selbst um.
#
# "User/Group": Der Name (oder #Nummer) des Benutzers bzw. der Gruppe,
# der/die den Webserver starten kann.
# Auf SCO (ODT 3) tragen Sie "User nouser" und "Group nogroup" ein.
# Auf HPUX sind Sie möglicherweise nicht in der Lage, gemeinsame
# Speicherbereiche als "nobody" zu nutzen. Als Ausweg wird vorgeschlagen,
# einen Benutzer "www" zu erstellen und diesen Benutzer zu verwenden.
#
# Beachten Sie, daß manche Kernels setgid (Group) oder semctl (IPC_SET)
# verweigern, wenn die Gruppennummer über 60000 liegt; auf solchen Systemen
# können Sie keine Gruppe "nogroup" verwenden.
#
User wwwrun
Group nogroup
#
#-------------------------------------------------------------
#
# "ServerAdmin" gibt die Adresse an, an die eventuelle Probleme mit dem Server
# gemeldet werden können. Diese Adresse erscheint auf allen Dokumenten,
# die vom Server generiert werden, zum Beispiel Fehlermeldungen.
#
ServerAdmin root@localhost
#
# "ServerName" erlaubt Ihnen, einen Hostnamen festzulegen, der an Clients
# zurückgegeben wird - falls er sich von dem unterscheiden soll, den das
# Programm ausgeben würde (zum Beispiel können Sie "www" statt des realen
# Hostnamens festlegen).
#
# ACHTUNG: Sie können nicht einfach irgendeinen Hostnamen erfinden und erwarten,
# daß er funktioniert. Ein Name, den Sie hier angeben, muß ein gültiger DNS-Name
# sein. Wenn Ihnen nicht klar ist, was damit gemeint ist, fragen Sie Ihren
# Netzwerkadministrator.
# Wenn Ihr Host nicht über einen DNS-Namen verfügt, geben Sie hier seine IP-Adresse
# an. Sie können dann auf jeden Fall über diese IP-Adresse Zugriff bekommen (z.B.
# http://192.168.0.1).
#
# 127.0.0.1 ist die lokale Loopback-Adresse für das TCP/IP-Protokoll, meist
# als "localhost" bezeichnet. Ihre Maschine müßte sich mit dieser Adresse immer
# selbst erkennen. Wenn Sie Apache ausschließlich dazu einsetzen, lokale Tests
# und Entwicklungsarbeiten durchzuführen, können Sie hier 127.0.0.1 als
# Servernamen eintragen.
#
ServerName www.ihr_name.de
#
# "DocumentRoot": das Verzeichnis, von dem aus die zur Publikation
# vorgesehenen Dokumente erreicht werden können. Im Standardfall werden
# sämtliche Server-Anfragen von diesem Verzeichnis aus beantwortet, und
# symbolische links sowie Aliase können auf andere Verzeichnisse verweisen.
#
DocumentRoot "/usr/local/httpd/htdocs"
#
# Jedes Verzeichnis, auf das Apache zugreifen kann, kann unter
# Berücksichtigung dessen, welche Dienste und Features hier (und in
# Unterverzeichnissen) erlaubt und/oder nicht zugelassen sind,
# konfiguriert werden.
#
# Zuerst wird ein "Standard" so konfiguriert, daß sich ein sehr restriktiver
# Satz an Berechtigungen ergibt.
#
<Directory />
AuthUserFile /etc/httpd/passwd
```

```
AuthGroupFile /etc/httpd/group
Options -FollowSymLinks +Multiviews
AllowOverride None
</Directory>
#
# ACHTUNG !!
# Von hier an muß ZWINGEND für jedes einzelne Verzeichnis festgeschrieben
# werden, welche Berechtigungen ihm zugestanden werden sollen. Falls irgendetwas
# nicht wie erwartet funktioniert, muß sichergestellt werden, daß die
# dafür gültigen Bedingungen in den folgenden Abschnitten korrekt
# festgelegt wurden. Wenn das nicht erfolgt, gelten die soeben bestimmten
# restriktiven Standardbedingungen.
#
# Zunächst folgen die Festlegungen für das "DocumentRoot"-Verzeichnis.
# (Der Pfad muß mit dem weiter oben angegebenen Pfad übereinstimmen).
#
<Directory "/usr/local/httpd/htdocs">
#
# die folgenden Werte können auch "None" oder "All" sein, oder irgendeine
# Kombination aus "Indexes", "Includes", "FollowSymLinks", "ExecCGI" und
# "MultiViews".
#
# Denken Sie daran, daß "MultiViews" explizit angegeben werden muß - "Options All"
# enthält diese Funktion nicht.
#
Options Indexes -FollowSymLinks +Includes MultiViews
#
# Jetzt werden die Optionen festgelegt, die von der Datei .htaccess überschrieben
# werden dürfen. Auch hier kann "All" oder irgendeine Kombination von "Options",
# "FileInfo", "AuthConfig" und "Limit" eingetragen werden.
#
AllowOverride None
#
# Festlegung, wer auf den Server zugreifen darf:
#
Allow from all
#
# unterbinden Sie aus Sicherheitsgründen bitte im Standardfall
# WebDAV
#
<IfModule mod_dav.c>
DAV Off
</IfModule>
#
# Sie sollten Server Side Includes (SSI) für Indexseiten zulassen, da der Inhalt
# solcher Seiten häufig dynamisch erstellt wird. Selbstverständlich muß das
# abgestellt bleiben, wenn Sie einen produktiven Server konfigurieren.
<Files /usr/local/httpd/htdocs/index.htm*>
Options -FollowSymLinks +Includes +MultiViews
</Files>
#
# Schützen Sie eine php3 Testseite (wenn vorhanden), damit sie von einem von
# außen zugreifenden System nicht eingesehen werden kann.
#
<Files test.php3>
Order deny,allow
deny from all
allow from localhost
</Files>
</Directory>
#
# "UserDir": Das Verzeichnis, das aufgerufen wird, wenn eine "~user"-Anforderung
# eintrifft.
#
```

```
<IfModule mod_userdir.c>
UserDir public_html
</IfModule>
#
# Zugriffsberechtigungen für UserDir-Verzeichnisse. Das Folgende ist ein
# Beispiel, mit dem diese Verzeichnisse nur gelesen werden dürfen (read-only).
#
#<Directory /home/*/public_html>
# AllowOverride FileInfo AuthConfig Limit
# Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
# <Limit GET POST OPTIONS PROPFIND>
# Order allow,deny
# Allow from all
# </Limit>
# <LimitExcept GET POST OPTIONS PROPFIND>
# Order deny,allow
# Deny from all
# </LimitExcept>
#</Directory>
#
# "DirectoryIndex": der Name/die Namen der Datei(en), der/die als Index-Datei(en)
# akzeptiert wird/werden. Verschiedene Eintragungen bitte durch Leerzeichen
# voneinander trennen.
#
<IfModule mod_dir.c>
DirectoryIndex index.htm index.html index.shtml index.asp index.php
</IfModule>
#
# "AccessFileName": Name der Datei, die in jedem Verzeichnis Informationen
# über Zugriffsberechtigungen zur Verfügung stellen kann.
#
AccessFileName .htaccess
#
# Die folgenden Zeilen schützen ".htaccess"-Dateien davor, daß Clients deren
# Inhalt auslesen können. Aus Sicherheitsgründen ist der Zugriff normalerweise
# nicht gestattet, da ".htaccess" Informationen zur Autorisierung enthält.
# Entfernen Sie die Kommentarzeichen, wenn Sie Web-Besuchern gestatten
# möchten, die Informationen in ".htaccess" zu lesen. Wenn Sie weiter oben
# die Anweisung "AccessFileName" mit irgendeinem Wert versehen haben, müssen
# auch hier entsprechende Veränderungen notiert werden.
#
# Analog sollten Sie Dateinamen wie ".htpasswd" für eine Datei mit dem aktuellen
# Paßwort verwenden, die dann ebenfalls vor Zugriffen geschützt ist.
#
<Files ~ "^\.ht">
Order allow,deny
Deny from all
</Files>
#
# "CacheNegotiatedDocs": In der Regel sendet Apache ein "Pragma: no-cache" mit jedem
# Dokument. Damit werden Proxy-Server veranlaßt, das Dokument nicht
# zwischenzuspeichern. Wenn Sie die folgende Zeile auskommentieren, wird dieses
# Verhalten deaktiviert und Proxyserver können dann Dokumente zwischenspeichern.
#
#CacheNegotiatedDocs
#
# "UseCanonicalName": (seit Version 1.3.X) Wenn Sie hier "On" angeben, wird immer
# dann das, was Sie weiter oben als "ServerName" und "port" eingetragen haben, auf
# Server-generierten Dokumenten auftauchen, wenn Apache veranlaßt wird, eine sich
# selbst referenzierende URL (ein Dokument, das auf den Server, von dem die Antwort
# kommt, zurückverweist) zu erstellen. Wenn Sie "Off" angeben, wird Apache versuchen,
# die für den Client gültige "hostname:port"-Angabe in solche Dokumente aufzunehmen.
# Diese Angaben beeinflussen auch die Variablen SERVER_NAME und SERVER_PORT
# in CGI-Scripts.
```

```
#
UseCanonicalName On
#
# "TypesConfig" zeigt an, ob/wo die Datei mime.types (oder ihr äquivalent)
# gefunden werden kann.
#
<IfModule mod_mime.c>
TypesConfig /etc/httpd/mime.types
</IfModule>
#
# "DefaultType" ist der Standard-MIME-Typ, den der Server einem Dokument zuweist,
# falls es nicht von einer Extension her genauer bestimmt wird. Wenn Ihr Server
# überwiegend Text- oder HTML-Dokumente enthält ist "text/plain" ein guter
# Wert. Wenn die meisten Dateien Binärdateien sind (z. B. Applikationen oder
# Bildformate wie .BMP) sollten Sie "application/octet-stream" angeben -
# statt den Browsern zu erlauben, diese Binärdateien so darzustellen, als ob
# sie Text wären.
#
DefaultType text/plain
#
# Das Modul mod_mime_magic gestattet dem Server, verschiedene Hinweise aus einem
# Dateiinhalt direkt zu nutzen, um den Dateityp zu erkennen. Die Anweisung
# "MIMEMagicFile" teilt dem Modul mit, wo diese Hinweise zur Typ-Definition zu
# finden sind. Das Modul mod_mime_magic ist allerdings kein Bestandteil des
# Standardservers. Sie müssen es selbst mit "LoadModule" einbinden [siehe der
# Absatz über dynamisch verbundene Objekte im Abschnitt "globale Umgebung" oben]
# oder den Server neu kompilieren und dabei mod_mime_magic als Teil der Konfiguration
# einbeziehen. Da es nicht zur Standard-Konfiguration gehört, wird es hier in einem
# eigenen <IfModule>-Container notiert. Das bedeutet, daß die Anweisung "MIMEMagicFile"
# nur dann befolgt werden kann, wenn das Modul in den Server eingebunden worden ist.
#
<IfModule mod_mime_magic.c>
MIMEMagicFile /etc/httpd/magic
</IfModule>
#
# "HostnameLookups": Zeichnet die Namen von Clients auf oder auch nur deren
# IP-Adresse; z.B. www.apache.org (on) oder 204.62.129.132 (off).
# Der Standard ist off, weil es insgesamt für das Netz besser ist, wenn die
# Leute dieses Feature bewußt nutzen. Es bedeutet nämlich, daß jede Anfrage
# eines Clients mindestens eine lookup-Rückfrage an den Nameserver weitergibt.
#
HostnameLookups Off
#
# "ErrorLog": die Ablagedatei für Fehlermeldungen.
# Wenn Sie innerhalb Ihrer <VirtualHost>-Container keine speziellen
# ErrorLog-Anweisungen niederlegen, werden Fehlermeldungen dieser virtuellen
# Hosts hier eingetragen. Wenn Sie allerdings eine eigene Protokolldatei für
# einen virtuellen Host festgelegt haben, werden Fehler auch dort protokolliert
# und nicht hier.
#
ErrorLog /var/log/httpd/error_log
#
# "LogLevel": Legt die Art und Menge der Eintragungen fest, die in der
# Protokolldatei abgelegt werden sollen. Mögliche Werte sind:
# debug, info, notice, warn, error, crit, alert, emerg.
#
LogLevel warn
#
# Die nächsten Anweisungen bestimmen, was bei der Verwendung einer
# CustomLog-Anweisung in die Protokolldatei eingetragen wird (siehe weiter unten).
#
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
```

```
LogFormat "%{User-agent}i" agent
#
# Ablageort und Format des Zugriffsprotokolls (Common Logfile Format).
# Falls Sie nicht innerhalb von <VirtualHost>-Containern eigene Zugriffs-Protokolldateien
# definiert haben, werden Zugriffe hier aufgezeichnet. Wenn es aber in Ihren
# <VirtualHost>-Containern definierte Zugriffs-Protokolldateien gibt, werden alle
# Berichte dort niedergelegt und nicht hier.
#
CustomLog /var/log/httpd/access_log common
#
# Wenn Sie getrennte "agent"- und "referer"-Protokolldateien haben möchten,
# entfernen Sie die Kommentarzeichen vor den folgenden Anweisungen.
#
#CustomLog /var/log/httpd/referer_log referer
#CustomLog /var/log/httpd/agent_log agent
#
# Wenn eine einzige Protokolldatei gewünscht wird mit allen Informationen
# (kombiniertes Protokolldateiformat), kann die folgende Anweisung
# genutzt werden.
#
#CustomLog /var/log/httpd/access_log combined
#
# In serverseitig generierte Dokumente (Fehlermeldungen, FTP-Verzeichnislisten
# usw., jedoch keine von CGI-Scripts erstellten Dokumente) können Sie eine Zeile
# einfügen, die die Server-Version und den virtuellen Hostnamen enthält. Wenn
# Sie hier "EMail" angeben, erscheint außerdem ein link mit der mail-Adresse
# des Server-Administrators.
# Angegeben werden kann einer dieser Werte: On | Off | EMail
#
ServerSignature On
# Aliases: Sie können hier so viele Aliasnamen angeben wie Sie wünschen
# (Anzahl ist nicht beschränkt).
#
<IfModule mod_alias.c>
#
# Beachten Sie: wenn Sie den Aliasnamen mit einem Slash ("/") beenden,
# verlangt Apache diesen Schrägstrich auch in einer URL. So würde hier
# "/icons" nicht als Alias akzeptiert, nur "/icons/" ist ein gültiger Aliasname.
#
Alias /icons/ "/usr/local/httpd/icons/"
<Directory "/usr/local/httpd/icons">
Options Indexes MultiViews
AllowOverride None
Order allow,deny
Allow from all
</Directory>
#
# ScriptAlias: damit wird festgelegt, welche Verzeichnisse Server-Scripts
# enthalten.
# ScriptAliases sind nahezu dasselbe wie Aliases, mit dem Unterschied, daß
# Dokumente des mit dem Alias benannten Verzeichnisses als Applikationen
# behandelt werden und immer dann über den Server laufen, wenn sie angefordert
# werden.
# Es gelten für den nachgestellten Schrägstrich dieselben Bestimmungen wie oben
# für Aliases angegeben.
#
ScriptAlias /cgi-bin/ "/usr/local/httpd/cgi-bin/"
<IfModule mod_perl.c>
# Für dasselbe cgi-bin-Verzeichnis sind zwei Aliase vorgesehen, um die Wirkung
# von zwei verschiedenen mod_perl-Betriebsarten zu sehen.
#
# für den Modus Apache::Registry
ScriptAlias /perl/ "/usr/local/httpd/cgi-bin/"
#
```

```
# für den Modus Apache::Perlrun
ScriptAlias /cgi-perl/ "/usr/local/httpd/cgi-bin/"
</IfModule>
#
# "/usr/local/httpd/cgi-bin" ist das Standard-Verzeichnis für CGI-Scripts.
# ändern Sie den Pfad entsprechend Ihren Festlegungen für ScriptAliases;
# Sie können mehrere Verzeichnisse angeben.
#
<Directory "/usr/local/httpd/cgi-bin">
AllowOverride None
Options None
Order allow,deny
Allow from all
</Directory>
</IfModule>
#
# Konfigurieren des Verzeichnisses /cgi-bin zur Ausführung von CGI-Programmen
#
<Location /cgi-bin>
AllowOverride None
Options +ExecCGI -Includes
SetHandler cgi-script
</Location>
#
# Wenn mod_perl aktiviert ist, laden Sie jetzt die Informationen der Konfiguration
#
<IfModule mod_perl.c>
Perlrequire /usr/include/apache/modules/perl/startup.perl
PerlModule Apache::Registry
#
# Setzen des Modus Apache::Registry für den Alias /perl
#
<Location /perl>
SetHandler perl-script
PerlHandler Apache::Registry
Options ExecCGI
PerlSendHeader On
</Location>
#
# Setzen des Modus Apache::PerlRun für den Alias /cgi-perl
#
<Location /cgi-perl>
SetHandler perl-script
PerlHandler Apache::PerlRun
Options ExecCGI
PerlSendHeader On
</Location>
</IfModule>
#
# "Redirect" (Weiterleitung) gibt Ihnen die Möglichkeit, Clients bekanntzugeben,
# welche Dokumente lediglich im Namensraum des Servers angewendet werden
# dürfen und sonst nichts weiter bewirken können. Damit kann Ihr Client
# erfahren, wo er nach einem weitergeleiteten Dokument suchen kann.
# Format: Redirect (alt)URI (neu)URL
#
#
# Jetzt folgen Anweisungen, die über die Art und Weise der Bildschirmausgabe
# von servergenerierten Verzeichnisanzeigen entscheiden.
#
<IfModule mod_autoindex.c>
#
# "FancyIndexing" bestimmt, ob Sie eine Standardausgabe sehen oder eine
# grafisch etwas aufgelockerte Anzeige auf Ihrem Monitor.
#
```

```
IndexOptions FancyIndexing
#
# "AddIcon*" teilt dem Server mit, welches Icon für welche Datei-Erweiterung
# verwendet werden soll. Diese Icons sehen Sie natürlich nur, wenn oben
# "FancyIndexing" angegeben wurde. Sie können die Liste beliebig bearbeiten.
#
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core
AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^
#
# "DefaultIcon" ist das Icon, das immer dann angezeigt wird, wenn für den
# Dateityp in der Liste oben kein eigenes Icon definiert wurde.
#
DefaultIcon /icons/unknown.gif
#
# "AddDescription" erlaubt Ihnen, eine Kurzbeschreibung der verschiedenen
# Dateitypen anzeigen zu lassen. Diese Kurzbeschreibungen werden ebenfalls
# nur dann angezeigt, wenn oben "FancyIndexing" gewählt wurde.
#
# Format: AddDescription "Beschreibungstext" Dateityp (Extension)
#
AddDescription "GZIP compressed document" .gz
AddDescription "tar archive" .tar
AddDescription "GZIP compressed tar archive" .tgz
#
# "HeaderName" ist der Name einer Datei, die einem Verzeichnis-Index vorangestellt
# werden soll und individuelle Informationen über dieses Verzeichnis enthält.
#
# "ReadmeName" ist der Name der README-Datei, in der der Server im Normalfall
# nach Zusatzinformationen sucht, die an Verzeichnislisten angehängt werden
# sollen.
#
# Wenn Sie in den Optionen "MultiViews" vorgegeben haben, wird der Server
# zuerst nach einer Datei "name.html" suchen und sie einbinden, falls sie
# existiert. Wenn es keine solche Datei gibt, sucht er nach "name.txt" und bindet
# sie als einfache Textdatei ein.
#
HeaderName HEADER
ReadmeName README
#
# "IndexIgnore" ist ein Satz an Dateinamen, die bei der Auflistung von
# Verzeichnisinhalten ignoriert und nicht angezeigt werden sollen. Wildcards
```

```
# sind zulässig
#
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
</IfModule>
#
# Dokument-Typen
#
<IfModule mod_mime.c>
#
# "AddEncoding" ist eine Anweisung, auf deren Grundlage manche Browser
# (Mosaic/X 2.1+) Dateiinhalte "on the fly" aus komprimierten Archiven verarbeiten
# können. Beachten Sie aber, daß nicht alle Browser dieses Feature zur Ver-
# fügung stellen.
# Trotz der Namensähnlichkeit haben die folgenden Add*-Anweisungen nichts mit
# dem oben angegebenen "FancyIndexing" zu tun.
#
AddEncoding x-compress Z
AddEncoding x-gzip gz tgz
#
# "AddLanguage": damit können Sie die Sprache eines Dokuments festlegen.
# Bei Server-Abfragen können dann Dateiinhalte vereinbart werden, so daß
# ein abfragender Browser Dateien in der Sprache erhält, die er am besten
# versteht.
#
# Anmerkung 1: Das Suffix muß nicht dasselbe sein wie das in Dokumenten
# bzw. META-tags verwendete Schlüsselwort für die Sprache. Beispielsweise
# können polnische Dokumente, deren Standard-Code im WWW "pl" ist,
# "AddLanguage pl .po" verlangen - um Konflikte mit dem für Perl-Scripts
# gebräuchlichen Suffix zu vermeiden.
#
# Anmerkung 2: Die Beispiele weiter unten zeigen, daß in sehr vielen Fällen
# die beiden Buchstaben für "Language" nicht mit den beiden Buchstaben
# für das zugehörige Land übereinstimmen - siehe z.B. "Dänemark/dk"
# und "Dänisch/da". Dagegen gibt es bei "Deutschland/de" und
# "Deutsch/de" durchaus eine übereinstimmung.
#
# Anmerkung 3: Bei "ltz" haben die Apache-Entwickler bewußt gegen die RFC
# verstoßen und ein aus drei Buchstaben bestehendes Kürzel definiert. Aber da
# Apache insgesamt ein "work in progress" ist, wird dieser Verstoß mit einer
# der nächsten Versionen korrigiert.
#
# Danish (da) - Dutch (nl) - English (en) - Estonian (ee)
# French (fr) - German (de) - Greek-Modern (el)
# Italian (it) - Korean (kr) - Norwegian (no)
# Portugese (pt) - Luxembourgeois* (ltz)
# Spanish (es) - Swedish (sv) - Catalan (ca) - Czech(cz)
# Polish (pl) - Brazilian Portuguese (pt-br) - Japanese (ja)
# Russian (ru)
#
AddLanguage da .dk
AddLanguage nl .nl
AddLanguage en .en
AddLanguage et .ee
AddLanguage fr .fr
AddLanguage de .de
AddLanguage el .el
AddLanguage he .he
AddCharset ISO-8859-8 .iso8859-8
AddLanguage it .it
AddLanguage ja .ja
AddCharset ISO-2022-JP .jis
AddLanguage kr .kr
AddCharset ISO-2022-KR .iso-kr
AddLanguage no .no
```

```
AddLanguage pl .po
AddCharset ISO-8859-2 .iso-pl
AddLanguage pt .pt
AddLanguage pt-br .pt-br
AddLanguage ltz .lu
AddLanguage ca .ca
AddLanguage es .es
AddLanguage sv .se
AddLanguage cz .cz
AddLanguage ru .ru
AddLanguage tw .tw
AddCharset Big5 .Big5 .big5
AddCharset WINDOWS-1251 .cp-1251
AddCharset CP866 .cp866
AddCharset ISO-8859-5 .iso-ru
AddCharset KOI8-R .koi8-r
AddCharset UCS-2 .ucs2
AddCharset UCS-4 .ucs4
AddCharset UTF-8 .utf8
# "LanguagePriority": damit können Sie Prioritäten setzen, wenn Server und
# anfragender Browser über Dateiinhalte zu verhandeln beginnen.
# Geben Sie die Sprachen in der Reihenfolge der für Sie gültigen Prioritäten
# an.
#
<IfModule mod_negotiation.c>
LanguagePriority en da nl et fr de el it ja kr no pl pt pt-br ru ltz ca es sv tw
</IfModule>
#
# "AddType" gibt Ihnen die Möglichkeit, die Informationen der Datei "mime.types"
# einzuarbeiten, ohne diese Informationsdatei ausgeben oder bestimmten Dateien
# bestimmte Typen zuordnen zu müssen.
#
# Beispielsweise verwendet das PHP3.X-Modul (das kein Teil der Apache-
# Distribution ist - siehe http://www.php.net) die folgenden charakteristischen
# Zeilen:
#
<IfModule mod_php3.c>
AddType application/x-httpd-php3 .php3
AddType application/x-httpd-php3-source .phps
AddType application/x-httpd-php3 .phtml
</IfModule>
#
# und für PHP 4.x nehmen Sie:
#
<IfModule mod_php4.c>
AddType application/x-httpd-php .php
AddType application/x-httpd-php .php4
AddType application/x-httpd-php-source .phps
</IfModule>
AddType application/x-tar .tgz
AddType text/vnd.wap.wml wml
AddType text/vnd.wap.wmlscript wmls
AddType application/vnd.wap.wmlc wmlc
AddType application/vnd.wap.wmlscriptc wmlsc
Addtype image/vnd.wap.wbmp wbmp
#
# "AddHandler" erlaubt es, ausgewählte Dateitypen an "Handler" zu übergeben,
# unabhängig vom Dateityp selbst. Diese "Handler" (Behandlungsroutinen) können
# entweder im Server enthalten sein oder mit dem Befehl "action" eingebunden
# werden (siehe unten).
#
# Wenn Sie SSI (Server Side Includes) oder CGI außerhalb der mit ScriptAlias
# festgelegten Verzeichnisse zulassen möchten, entfernen Sie die
# Kommentarzeichen vor den nachfolgenden Angaben.
```

```
#
AddHandler cgi-script .cgi
#
# für HTML-Dateien, die auf dem Server "geparst" werden.
#
AddType text/html .shtml
AddHandler server-parsed .shtml
#
# Entfernen Sie das Kommentarzeichen, wenn Sie Apache's "send-asis"
# HTTP-Datei zulassen möchten.
#
#AddHandler send-as-is asis
#
# Wenn Sie Imagemaps zulassen möchten:
#
#AddHandler imap-file map
#
# Um type-maps einzusetzen, können Sie den folgenden Eintrag nutzen
#
#AddHandler type-map var
</IfModule>
#
# "Action" definiert einen Medien(Datei)-Typ, der immer dann zugeordnet wird, wenn eine
# entsprechende Datei aufgerufen wird. Dadurch entfällt der Zwang, Pfadnamen immer wieder
# notieren zu müssen.
# Format: Action media/type /cgi-script/location
# Format: Action handler-name /cgi-script/location
#
#
# "MetaDir" ist der Name eines Verzeichnisses, in dem Apache Dateien mit
# "META-Informationen" finden kann. Diese Dateien enthalten zusätzliche
# HTTP-Header, die der Server grundsätzlich jedem auszuliefernden Dokument
# mitgeben soll.
#
#MetaDir .web
#
# "MetaSuffix" bestimmt den Dateityp (das zugehörige Suffix) für die Datei, die
# diese META-Informationen enthält.
#
#MetaSuffix .meta
#
# Konfigurierbare Fehlermeldungen (Apache-Stil)
# das kann auf drei verschiedene Arten geschehen:
#
# 1) plain text
ErrorDocument 500 "Der Server sagt nur pfffffffffftt"
# Nachbemerkung: die vorangestellten Anführungszeichen (") markieren die
# Meldung selbst als Ausgabetext, der sonst keine weiteren Ausgaben
# erzeugt
#
# 2) lokale Weiterleitung
# Damit wird sofort zu einer lokalen URL "/missing.html" weitergeleitet.
#ErrorDocument 404 /missing.html
#
# Sie können natürlich auch zu einem Script oder einem Dokument mit SSI
# (server-side-includes) weiterleiten.
#ErrorDocument 404 /cgi-bin/missing_handler.pl
#
# 3) externe Weiterleitung
#ErrorDocument 402 http://some.other_server.com/subscription_info.html
#
# Nachbemerkung: Viele mit der originalen Anfrage verbundene Umgebungs-
# Variablen werden einem auf diese Weise aufgerufenen Script nicht zur
# Verfügung stehen.
```

```
#
# Anpassung des Browserverhaltens
#
<IfModule mod_setenvif.c>
#
# Die folgenden Anweisungen modifizieren das normale Verhalten bei
# Reaktionen auf HTTP-Anfragen.
# Die erste Anweisung verhindert persistente Verbindungen für Netscape 2.x
# und andere Browser, die solche Verbindungen nur vortäuschen. Es gibt ein
# paar bekannte Probleme bei den Implementationen dieser Browser.
# Die zweite Anweisung gilt für Microsoft Internet Explorer 4.0b2, in den
# HTTP/1.1 nicht korrekt implementiert wurde und der deshalb "keepalive" nicht
# richtig unterstützt, sobald über persistente Verbindungen 301- oder 302
# (Weiterleitungs)-Reaktionen des Servers angefordert werden.
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
#
# Die folgende Anweisung verhindert HTTP/1.1-Ausgaben an Browser, die
# gegen die HTTP/1.0- Spezifikationen verstoßen und dadurch nicht fähig
# sind, eine Standard-Reaktion auf HTTP/1.1 zu erzeugen.
#
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
</IfModule>
#
# Wenn Sie sich Informationen über den Serverstatus mit der URL
# http://servername/server-status ausgeben lassen möchten, geben Sie
# hier statt ".your_domain.com" den passenden Domainnamen an.
#
<IfDefine STATUS>
<Location /server-status>
SetHandler server-status
Order deny,allow
Deny from all
# Allow from http://your.domain.com
Allow from localhost
</Location>
#
# Wenn Sie sich mit der URL http://servername/server-info die Konfiguration
# von "Remote"-Servern anzeigen lassen wollen (das setzt voraus, daß das
# Modul mod_info.c geladen ist), verwenden Sie die folgenden Zeilen.
# Geben Sie statt ".your_domain.com" die gewünschte Domain an.
#
<Location /server-info>
SetHandler server-info
Order deny,allow
Deny from all
Allow from localhost
</Location>
#
# für mod_perl sollten Sie "perl-status"-Abfragen ermöglichen
#
<IfModule mod_perl.c>
<Location /perl-status>
SetHandler perl-script
PerlHandler Apache::Status
order deny,allow
deny from all
allow from localhost
</Location>
</IfModule>
</IfDefine>
```

```
#
# Es gab Hinweise von Leuten, die versucht haben, einen alten Fehler aus den
# Tagen der Version "pre-1.1" auszunutzen. Dieser Fehler betraf die Ausgabe
# von CGI-Scripts auf eine Weise, als seien sie Bestandteile von Apache.
# Wenn Sie die folgenden Zeilen auskommentieren, werden diese Zugriffe zu
# einem Protokollscript auf phf.apache.org umgeleitet. Sie können sie auch
# zunächst selbst aufzeichnen, indem Sie ein script "/support/phf_abuse_log.cgi"
# benutzen.
#
#<Location /cgi-bin/phf*>
# Deny from all
# ErrorDocument 403 http://phf.apache.org/phf_abuse_log.cgi
#</Location>
#
# Anweisungen zur Konfiguration eines Proxy-Servers. Entfernen Sie die
# Kommentarzeichen, wenn Sie Proxies einrichten möchten.
#
#<IfModule mod_proxy.c>
# ProxyRequests On
#
# <Directory proxy:*>
# Order deny,allow
# Deny from all
# Allow from .your_domain.com
# </Directory>
#
# Zulassen/Verhindern der Behandlung von HTTP/1.1 "Via:"-Headern.
# ("Full" schreibt die Serverversion dazu, "Block" entfernt alle "Via:"-Header)
# Mögliche Angaben: Off | On | Full | Block
#
#ProxyVia On
#
#
# Wenn Sie möchten, daß ein Proxy Dokumente in seinem Cache
# zwischenspeichert, entfernen Sie die Kommentarzeichen vor den
# folgenden Zeilen (ohne "CacheRoot" gibt es keine Zwischenspeicherung).
#
# CacheRoot "/var/cache/http"
# CacheSize 5
# CacheGcInterval 4
# CacheMaxExpire 24
# CacheLastModifiedFactor 0.1
# CacheDefaultExpire 1
# NoCache a_domain.com another_domain.edu joes.garage_sale.com
#</IfModule>
#
################################################################
################################################################
#
# 3. Abschnitt: Virtuelle Hosts
# -----------------------------
#
# "VirtualHost": Wenn mit mehreren Domain- bzw. Host-Namen auf diesem Server
# gearbeitet werden soll, können virtuelle "Container" für jeden einzelnen
# Hostnamen festgelegt werden.
# Bitte unbedingt die Dokumentation auf <http://www.apache.org/docs/vhosts/>
# nachlesen bzw. das mit der Distribution ausgelieferte Manual.
# Um die Konfiguration der virtuellen Hosts zu überprüfen, steht der
# Befehlszeilen-Parameter "-S" zur Verfügung.
#
#
# Konfiguration für DNS-Namen-gestützte virtuelle Hosts
#
#NameVirtualHost *
```

```
#
# Beispiel für einen virtuellen Host:
#
# Fast jede Apache-Anweisung kann in einem "VirtualHost"-
# Container verwendet werden.
# Der erste Container in der (beliebig langen) Liste wird dabei immer genutzt,
# wenn eine Webadresse ohne bekannten Servernamen verlangt wird.
#
#<VirtualHost *>
# ServerAdmin webmaster@dummy-host.example.com
# DocumentRoot /www/docs/dummy-host.example.com
# ServerName dummy-host.example.com
# ErrorLog logs/dummy-host.example.com-error_log
# CustomLog logs/dummy-host.example.com-access_log common
#</VirtualHost>
#
######################################################################
######################################################################
#
# 4. zusätzliche Module und anderes
# ---------------------------------
#
######################################################################
```