Apache Web server Complete Guide

Dedoimedo

Foreword

Since I cannot be sure you have read my introductory article on my website (http://www.dedoimedo.com/), here's an abstract of what you should expect from this document.

The Web server - Apache - Complete Guide is one of the many topics covered in the series of books that I'm writing on Linux, the goal of which is to help any enthusiastic Windows user or a Linux newbie become a powerful, confident Linux professional. As a preview of what you should expect when these books become published, I have decided to post a single Part on my website.

I am truly convinced that you will thoroughly enjoy this document, for it has been written with care and attention to tiniest details. Every procedure is explained step by step, accompanied by numerous examples and screenshots.

I hope this will be the best guide on the Apache Web server you will have ever read.

The only thing that you will miss is the fact that links to other Parts, covering other material, are not available in this standalone release. However, every procedure required to setup the Web

server is fully self-contained. You will be able to fully configure the Apache server by just using this document as your guide.

Lastly, let's get one thing straight: you will not become Apache gurus by reading this document. For that matter, I'm not an Apache guru, either. There are so many aspects to the usability and security of the Apache Web server, it is practically impossible to put them all in a single book.

However, by reading this document, you WILL learn how to use the Apache Web server on the basic and intermediate level. And there's no guide that will explain things as simply and as beautifully as mine, I guarantee that.

From here on, the sky's your limit.

About

Dedoimedo is a website specializing in step-by-step tutorials intended for human beings. Everything posted on my website is written in plain, down-to-Earth English, with plenty of screenshot examples and no steps ever skipped. You won't easily find tutorials simpler or friendlier than mine.

Myself, I'm a former physicist, currently living the dream and working as a Linux Systems Expert, hacking the living daylight out of the Linux kernel. Few people have the privilege to work in what is essentially their hobby and passion and truly love it, so I'm most grateful for the beauty, freedom and infinite possibilities of the open-source world. I also hold a bunch of certifications of all kinds, but you can read more about those on my website.

Have fun!

Copyright

Disclaimer

I am not very fond of disclaimers, but they are a necessary part of our world. So here we go:

I must emphasize the purpose of this document is for educational purposes. It is not an official document and should not be treated as such. Furthermore, I cannot take any responsibilities for errors, inaccuracies or damages resulting from the use of this article (and its contents).

All of the material in this document has been carefully worded and prepared by me. However, if for some reason you may feel this document infringes on copyright or intellectual property of another work, please contact me with a detailed explanation pointing to a troublesome part and I will try to sort the problem in the best way possible.

This tutorial has also been posted as a web article on my website. For any news, changes or updates, you should always refer first to www.dedoimedo.com.

Contents

1 Introduction **11**

2 Basic Setup **13**

2.1 Verify installation 13

2.2 Package files 14

2.3 Main configuration file(s) 15

2.4 Backup 16

2.5 Edit the httpd.conf configuration file 16

2.5.1 ServerRoot 17

2.5.2 PidFile 18

2.5.3 ServerName 18

2.5.4 /etc/hosts file 19

2.5.5 DocumentRoot 23

2.5.6 ErrorLog 23

2.5.7 Listen 25

2.6 Create your HTML documents 26

2.7 Start the Web Server 28

2.8 Access the web site 30

2.8.1 Local access 30

2.8.2 External access 32

2.9 Summary of basic setup 36

3 Advanced setup 37
3.1 Directory tags . . . 37
3.1.1 Order (allow, deny) . . . 40
3.1.2 Indexes . . . 43
3.1.3 DirectoryMatch . . . 48
3.2 Files tags . . . 48
3.3 Location tags . . . 49
3.4 Directory, Files and Location . . . 51
3.5 Redirect . . . 52
3.6 Virtual Hosts . . . 54
3.6.1 Single IP, two websites . . . 57
3.6.2 Two IPs, two websites . . . 63
3.6.3 Other scenarios . . . 69
3.6.3.1 Different content for intranet and Internet . . 69
3.6.3.2 Different websites on different ports . . . 72
3.7 Modules . . . 74
3.7.1 Module types . . . 74
3.8 View installed modules . . . 74
3.8.1 LoadModule . . . 76
3.8.2 mod_access . . . 78
3.8.3 mod_dir . . . 78
3.8.4 mod_perl . . . 79
3.8.5 mod_python . . . 79
3.8.6 mod_ssl . . . 79

4 .htaccess 80
4.1 Create .htaccess file . . . 83
4.2 Create .htpasswd file . . . 83
4.3 Copy .htaccess to restricted directory . . . 85
4.4 Configure httpd.conf to allow authentication via .htaccess . . . 85
4.5 Restart server . . . 86

4.6 Test setup . . . 87
4.7 Other configurations . . . 90
4.7.1 Inheritance & performance loss . . . 90
4.7.2 Disable web access to .htaccess . . . 91

5 Secure Web server 93
5.1 Encrypted session . . . 94
5.2 Requirements . . . 95
5.3 Limitations . . . 96
5.4 Setup . . . 97
5.4.1 Main configuration file(s) . . . 97
5.4.2 Backup . . . 98
5.4.3 Edit the ssl.conf configuration file - part 1 . . . 98
5.4.3.1 LoadModule . . . 98
5.4.3.2 Listen . . . 99
5.4.3.3 VirtualHost . . . 99
5.4.4 Create SSL certificate . . . 101
5.4.4.1 Create Certificate Authority (CA) . . . 102
5.4.4.2 Create server key . . . 107
5.4.4.3 Create Certificate Signing Request (CSR) . . . 108
5.4.4.4 Sign Certificate Signing Request (CSR) with Certificate Authority (CA) . . . 110
5.4.4.5 Verify certificates . . . 112
5.4.5 Edit ssl.conf configuration file - part 2 . . . 115
5.4.5.1 Server Certificate . . . 115
5.4.5.2 Server Private Key . . . 116
5.4.5.3 Certificate Authority . . . 116
5.4.6 Test setup . . . 117
5.4.7 Mini-summary . . . 122
5.4.7.1 Names . . . 122
5.4.7.2 Commands . . . 123

5.4.7.3 Difference between self-signed and CA-signed certificates . . . 124
5.4.7.4 Verification . . . 125
5.4.7.5 File names and locations . . . 125
5.5 Extras . . . 126
5.5.1 Do not use password-protected server keys . . . 126
5.5.1.1 Create server key without password . . . 126
5.5.2 Submission of CSR to CA . . . 128
5.5.2.1 Create CSR . . . 128
5.5.2.2 Send CSR to CA . . . 129
5.5.2.3 Verify certificate . . . 129
5.6 General considerations . . . 130
5.6.1 Use secure server only . . . 130
5.6.2 Use only IP-based virtual hosts . . . 130
5.6.3 Use server.key as file name for the server key . . . 131

6 Other configurations 132
6.1 Firewall rules . . . 132
6.1.1 Advanced firewall rules . . . 133
6.1.1.1 Port forwarding . . . 134
6.1.1.2 Destination NAT . . . 135
6.1.1.3 Static NAT . . . 136
6.2 Enable Web server on startup . . . 139

7 Security 140
7.1 Updates . . . 141
7.2 Hide your server version . . . 141
7.3 Logs . . . 143
7.4 Permissions . . . 143
7.5 Access to root (/) . . . 145
7.6 AllowOverride . . . 145

7.7 Disable public access to .ht files . . . 146
7.8 Dynamic content . . . 146
7.8.1 Disable CGI . . . 146
7.8.2 Disable Server Side Includes (SSI) . . . 147
7.9 Disable unnecessary modules . . . 147
7.10 Use ModSecurity (mod_security) module . . . 147
7.11 Chroot Jail . . . 148
7.12 Secure web server only . . . 149
7.12.1 Different DocumentRoot . . . 149
7.12.2 Permissions . . . 149
7.12.3 Duration of certificates . . . 149
7.13 Word of caution . . . 150

8 Additional resources 151

9 Exercises 152
9.1 Questions . . . 153
9.1.1 Secure Web server & VirtualHost . . . 153
9.1.2 Directory, Files and Locations . . . 154
9.1.3 Server functionality, 1 . . . 154
9.1.4 Server functionality, 2 . . . 155
9.1.5 .htaccess . . . 156
9.2 Answers . . . 157
9.2.1 Secure Web server & VirtualHost . . . 157
9.2.2 Directory, Files and Locations . . . 157
9.2.3 Server functionality, 1 . . . 157
9.2.4 Server functionality, 2 . . . 158
9.2.5 .htaccess . . . 159

Chapter 1

Introduction

A Web server is a server that is responsible for accepting HTTP requests from web clients and serving them HTTP responses, usually in the form of web pages containing static (text, images etc) and dynamic (scripts) content.

The Apache Web server has been the most popular and widely used Web server for the last decade. It is used by approximately 50% of all websites. Apache is cross-platform, lightweight, robust, and used in small companies as well as large corporations. Apache is also free and open-source.

The Apache Web server has almost endless possibilities, due to its great modularity, which allows it to be integrated with numerous other applications. One of the most popular bundles is the LAMP Web server application stack, which includes the Apache Web server alongside MySQL, PHP, Perl, and Python.

The Apache Web server is developed by the Apache Software Foundation. You can read more about Apache on Wikipedia.

Being able to configure and secure the Apache Web server is one of the most important tasks for a (Linux) system administrator. Almost every company has some sort of a website that advertises it, including intranet pages that are used by the company's workers. The Web interface is used for many tasks beside pure browsing, including tasks as simple as meal orders and shift rosters, but also important tasks like administration of databases. In most cases, a local web server is setup to accommodate these needs.

If you are working for a company that hosts public websites, the task becomes even more complicated. Web sites are used to serve content to billions of users daily. Whoever controls this content - controls the World Wide Web, from news and blogs to financial transactions. Web servers are hubs of information and power. Misconfigured or compromised servers can expose a large number of people to undesired content and potentially incur huge damages to involved parties.

Running a Web site is much more than opening a port and serving a few HTML pages. There are tremendous network usability and security considerations that must continuously be met, evaluated and improved in order to maintain a safe and effective Web server.

In this Part of the Book, we will learn how to properly setup and run the Apache Web server, including the secure (HTTPS) server.

Chapter 2

Basic Setup

In this chapter, we will setup a Web server that will serve pages on our internal network. In this chapter, we will perform the most basic setup with the minimum number of steps required to get the server running. Later, we will slowly expand, introducing new features and options.

2.1 Verify installation

First, we'll verify that Apache is indeed installed:

```
rpm -q httpd
```

If you get an empty prompt or a message saying the package is not installed, you will need to download and install it. If the shell displays the package name and version, you're good to go.

2.2 Package files

Rule no. 1: don't panic! The list before you might seem intimidating at the moment, but that is simply because you are not yet familiar with Apache. But don't worry. For now, treat the list as a reference only. At this stage, you don't need to know anything or remember anything. We will slowly cover everything, step by step.

Now, let us overview the location and purpose of the files used by the Apache server. Please note that the list is partial and includes only the most important entries. We will slowly expand this list as we go through the Part.

File name	**Description**
/usr/sbin/httpd	server binary
/etc/httpd	directory containing server configuration files
/etc/httpd/conf	directory containing main configuration files
/etc/httpd/conf.d	directory containing configuration files for individually packaged modules, like ssl, php, perl etc
/etc/httpd/logs	symbolic link to /var/log/httpd
/etc/httpd/modules	symbolic link to /usr/lib/httpd/modules
/etc/httpd/run	symbolic link to /var/run
/usr/lib/httpd/modules	server modules
/var/log/httpd	server log
/var/run	runtime variables
/var/run/httpd.pid	server process ID
/var/www/html	public html files

We will discuss the main configuration file shortly.

2.3 Main configuration file(s)

The main configuration file for the Apache Web server is:

```
/etc/httpd/conf/httpd.conf
```

This file is well commented and self explanatory. It contains quite a large number of settings, but we'll concentrate on just the few necessary to setup the server.

2.4 Backup

This is one of the most important things to remember. Always retain the copy of the original file so you can easily revert to the default. At the very least, do NOT delete default lines; instead, just comment them out so you'll be able to see what the original settings read and refer to them.

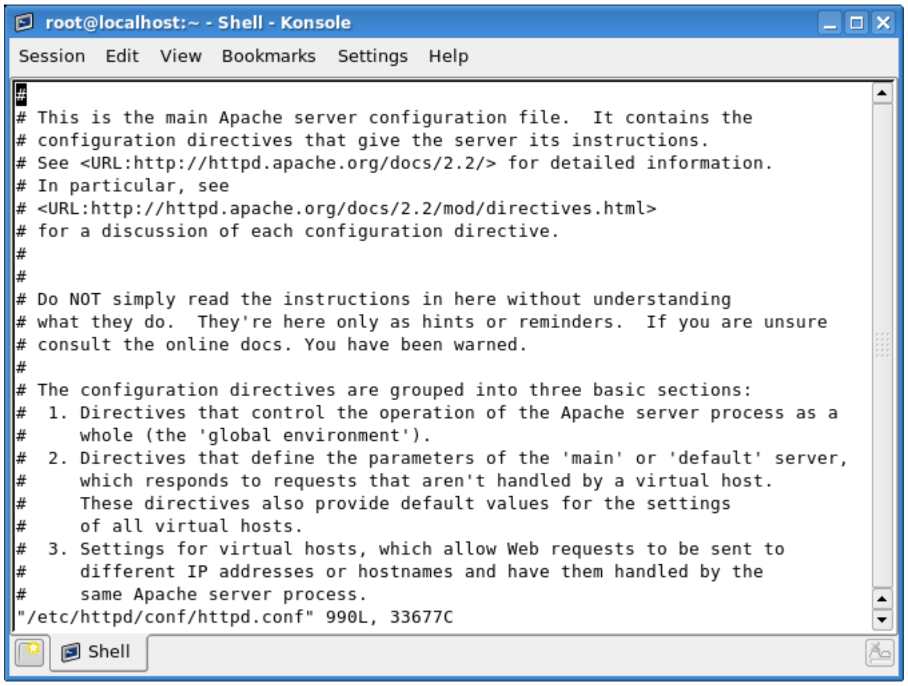
```
cp /etc/httpd/conf/httpd.conf→
    /etc/httpd/conf/httpd.conf-default
```

2.5 Edit the httpd.conf configuration file

Let's open this file in *vi* text editor and review the most important entries. The file has many options - but we need only a few. In fact, you will need to change just a *single* line to create your server and get it running. However, you should be familiar with some other settings before launching the server.

```
vi /etc/httpd/conf/httpd.conf
```

This is what the file looks like - at least the beginning of it:


root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

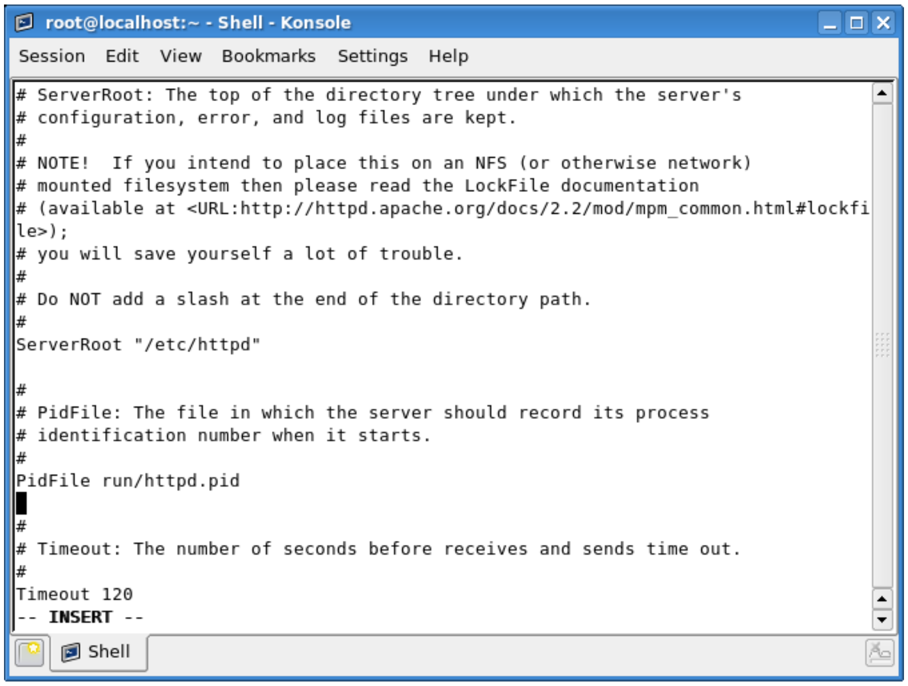
```
#
# This is the main Apache server configuration file.  It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.2/> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.2/mod/directives.html>
# for a discussion of each configuration directive.
#
#
# Do NOT simply read the instructions in here without understanding
# what they do.  They're here only as hints or reminders.  If you are unsure
# consult the online docs. You have been warned.
#
# The configuration directives are grouped into three basic sections:
#  1. Directives that control the operation of the Apache server process as a
#     whole (the 'global environment').
#  2. Directives that define the parameters of the 'main' or 'default' server,
#     which responds to requests that aren't handled by a virtual host.
#     These directives also provide default values for the settings
#     of all virtual hosts.
#  3. Settings for virtual hosts, which allow Web requests to be sent to
#     different IP addresses or hostnames and have them handled by the
#     same Apache server process.
"/etc/httpd/conf/httpd.conf" 990L, 33677C
```

Let's go over the most important entries you should remember for now.

2.5.1 ServerRoot

ServerRoot is the path to the server's configuration, error and log files. It is possible to change this path, provided all the necessary files are copied to the new location accordingly. We will later review this concept as a part of the security measure known as the Chroot Jail, but more about that later (). The default location is */etc/httpd*.

As you can see, the file is rich with easy-to-understand comments. Apropos the comments, please note that you should not place a trailing slash at the end of the specified path.


root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# NOTE!  If you intend to place this on an NFS (or otherwise network)
# mounted filesystem then please read the LockFile documentation
# (available at <URL:http://httpd.apache.org/docs/2.2/mod/mpm_common.html#lockfi
le>);
# you will save yourself a lot of trouble.
#
# Do NOT add a slash at the end of the directory path.
#
ServerRoot "/etc/httpd"

#
# PidFile: The file in which the server should record its process
# identification number when it starts.
#
PidFile run/httpd.pid

#
# Timeout: The number of seconds before receives and sends time out.
#
Timeout 120
-- INSERT --
```

2.5.2 PidFile

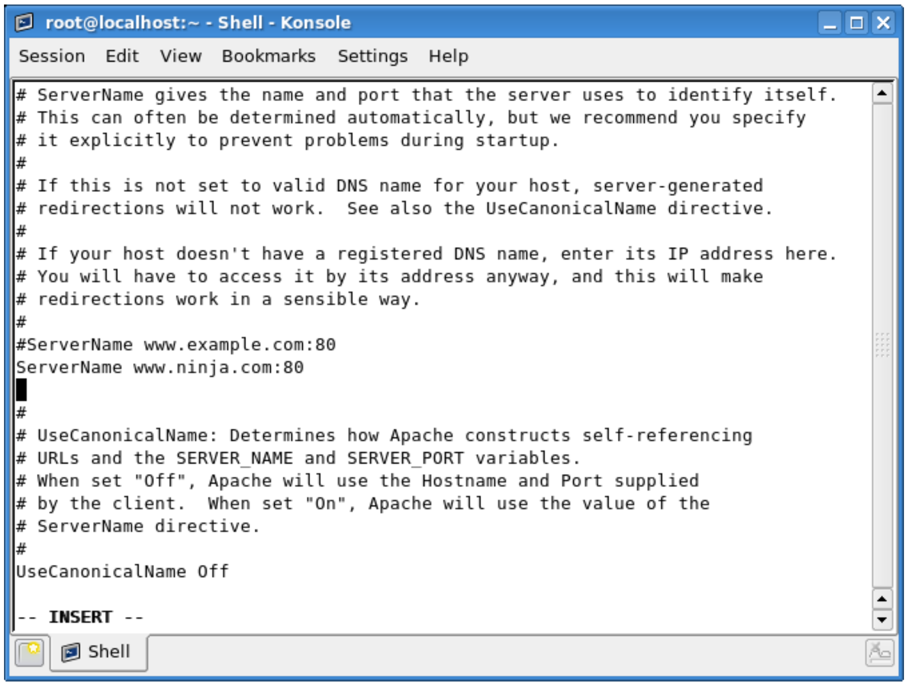
PidFile **is the process identification number for the** ***httpd*****. This process number is important, because Apache spawns numerous child processes when running to accommodate the web traffic. It allows you to monitor and manipulate your server processes. See image above.**

2.5.3 ServerName

This is the one setting you will have to change to get your server running. This is where you declare the name of your website.

I will use *www.ninja.com* - just a random name with no association whatsoever to the real site bearing this name.

The generous comments in the file remind us that if we do not have a registered DNS name, we should use an IP address. One, we'll discuss registered DNS names later. Two, we're going to use the *hosts* file to demonstrate the address-to-name translation.

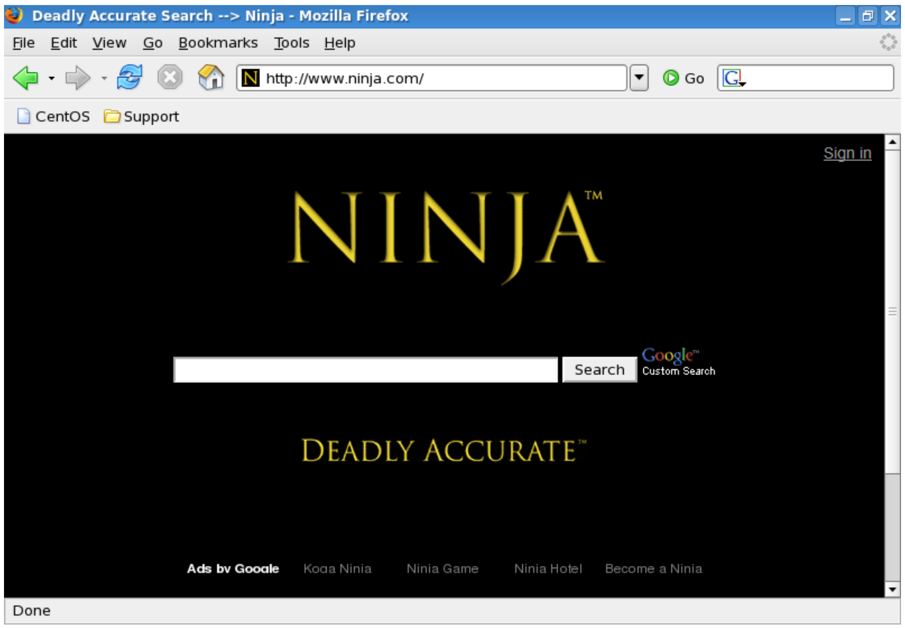

root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# ServerName gives the name and port that the server uses to identify itself.
# This can often be determined automatically, but we recommend you specify
# it explicitly to prevent problems during startup.
#
# If this is not set to valid DNS name for your host, server-generated
# redirections will not work.  See also the UseCanonicalName directive.
#
# If your host doesn't have a registered DNS name, enter its IP address here.
# You will have to access it by its address anyway, and this will make
# redirections work in a sensible way.
#
#ServerName www.example.com:80
ServerName www.ninja.com:80

#
# UseCanonicalName: Determines how Apache constructs self-referencing
# URLs and the SERVER_NAME and SERVER_PORT variables.
# When set "Off", Apache will use the Hostname and Port supplied
# by the client.  When set "On", Apache will use the value of the
# ServerName directive.
#
UseCanonicalName Off

-- INSERT --
```

2.5.4 /etc/hosts file

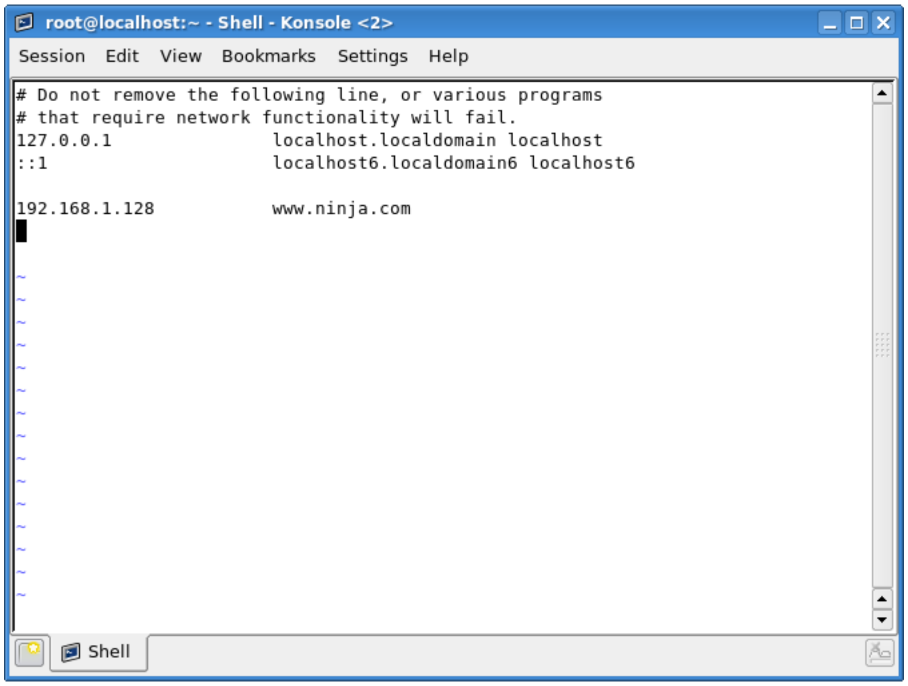
As you already know, the *hosts* file allows easy matching of names to IP addresses, . In general, using the *hosts* file is a good way of testing your IP-to-name (or vice versa) configurations before committing these changes into a production environment.

First, with no new entries added to the *hosts* file, typing *www.ninja.com* in the address bar of a web browser takes us to the site itself (on the Internet).



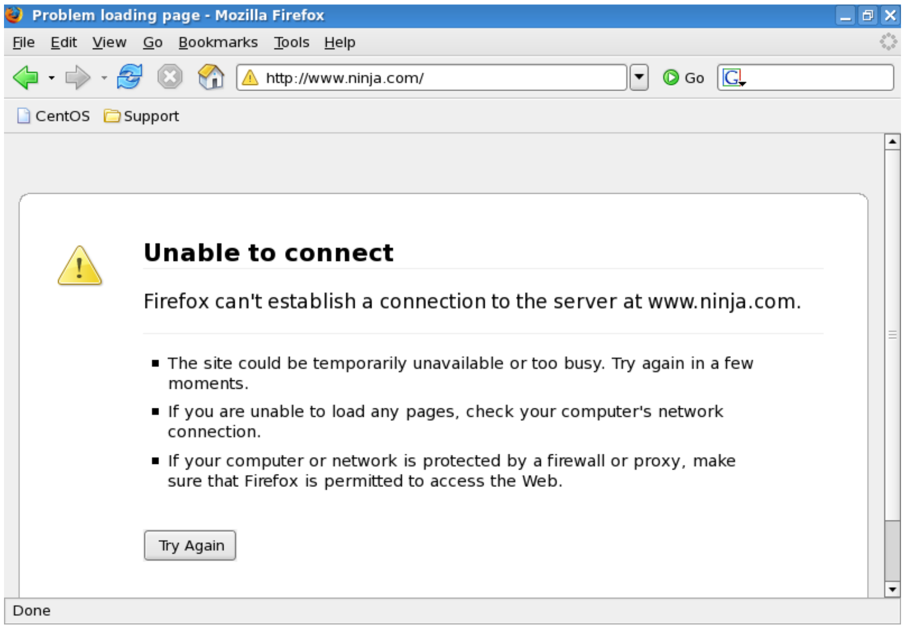
Now, we shall edit the file and add an entry, pointing *www.ninja.com* to a local IP address.

```
vi /etc/hosts
```


root@localhost:~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help
Shell

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain localhost
::1                     localhost6.localdomain6 localhost6

192.168.1.128           www.ninja.com

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

After saving the *hosts* file, we can no longer see the Internet site. Furthermore, we don't get any fancy results from our own Web server, because it is not running yet.


Problem loading page - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja.com/
Go
CentOS
Support
Try Again
Done

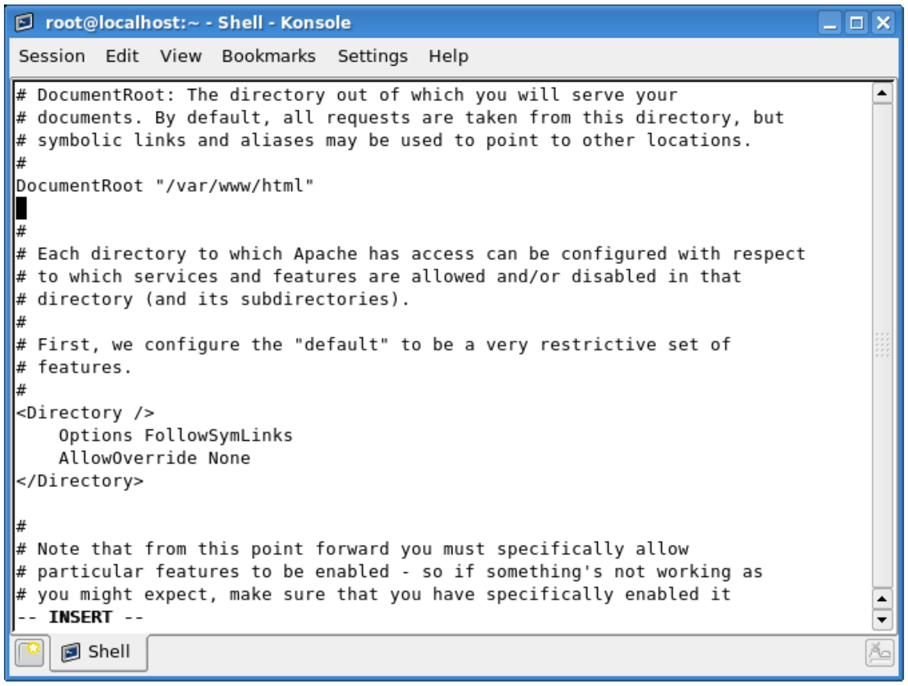
Unable to connect

Firefox can't establish a connection to the server at www.ninja.com.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

2.5.5 DocumentRoot

DocumentRoot **tells you where your web documents (html files, images etc) should be located. It is possible to reference files in other directories using aliases and symbolic links. The default directory is** ***/var/www/html.***

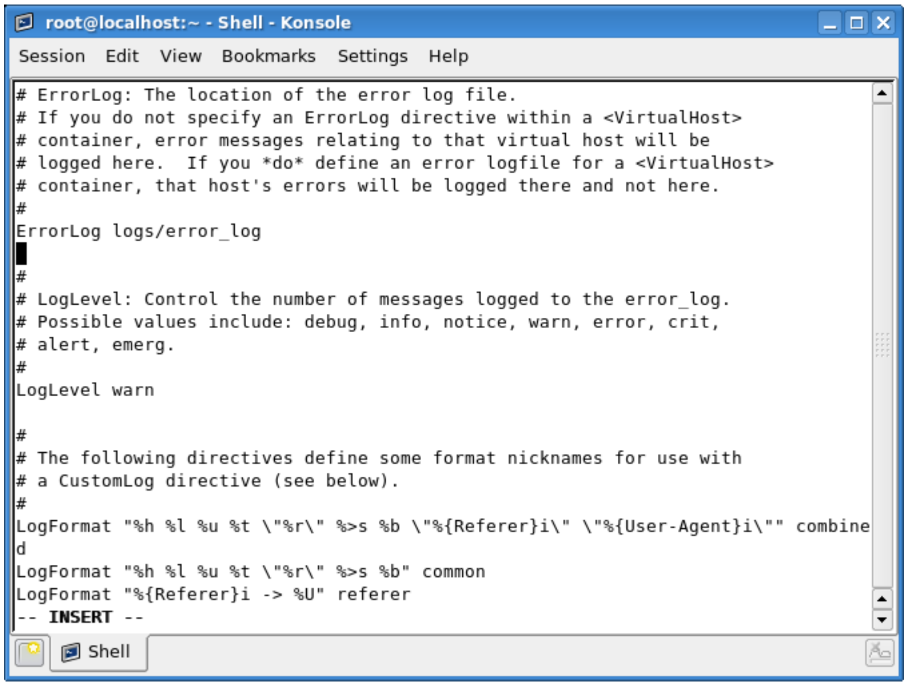

root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "/var/www/html"

#
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
-- INSERT --
```

2.5.6 ErrorLog

ErrorLog **tells you where the log containing all server errors is located. This file is critical for debugging and solving server misconfiguration problems and for proper traffic shaping. By default, all messages with the value of warning (*****warn*****) and higher will be logged. This is described in the** ***LogLevel*** **directive just below.**

The default location is *logs/error_log*. Please note that this is relative to the *ServerRoot*. Therefore, our log file is */etc/httpd/logs/error_log*. However, let us not forget that */etc/httpd/logs* is a symbolic link to */var/log/httpd*. Thus, finally, the actual error log is */var/log/httpd/error_log*.

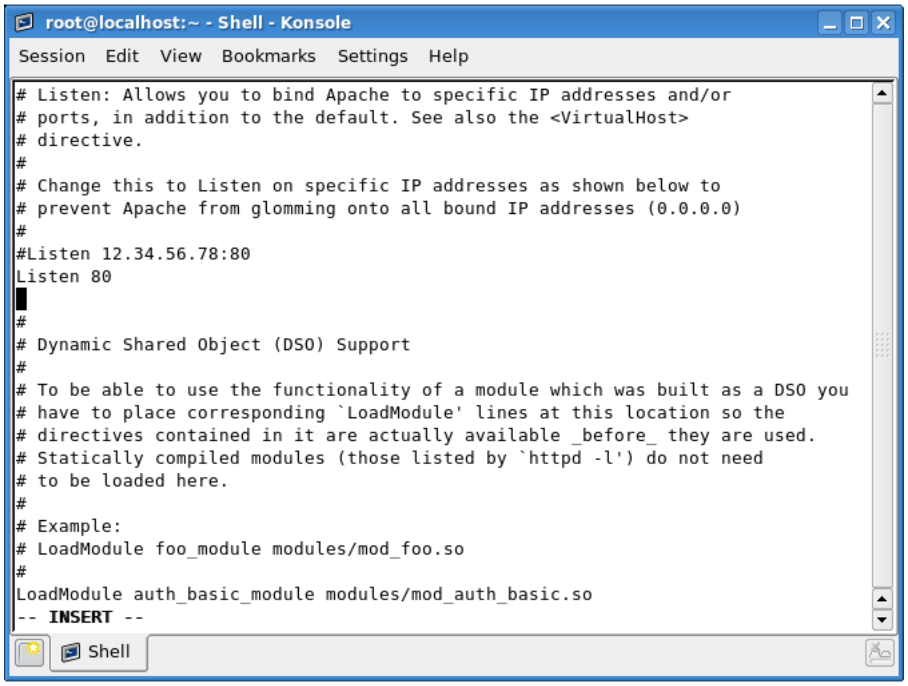

root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.
#
ErrorLog logs/error_log

#
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel warn

#
# The following directives define some format nicknames for use with
# a CustomLog directive (see below).
#
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combine
d
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
-- INSERT --
```

2.5.7 Listen

The ***Listen*** command tells the Web server what ports to use for incoming connections. By default, port 80 is used, although any one or several can be used. The accepted conventions calls for using port 80 for non-secure web communications (without any encryption of traffic). Secure web communications are normally handled on port 443.

We'll talk about the Secure Web server later.

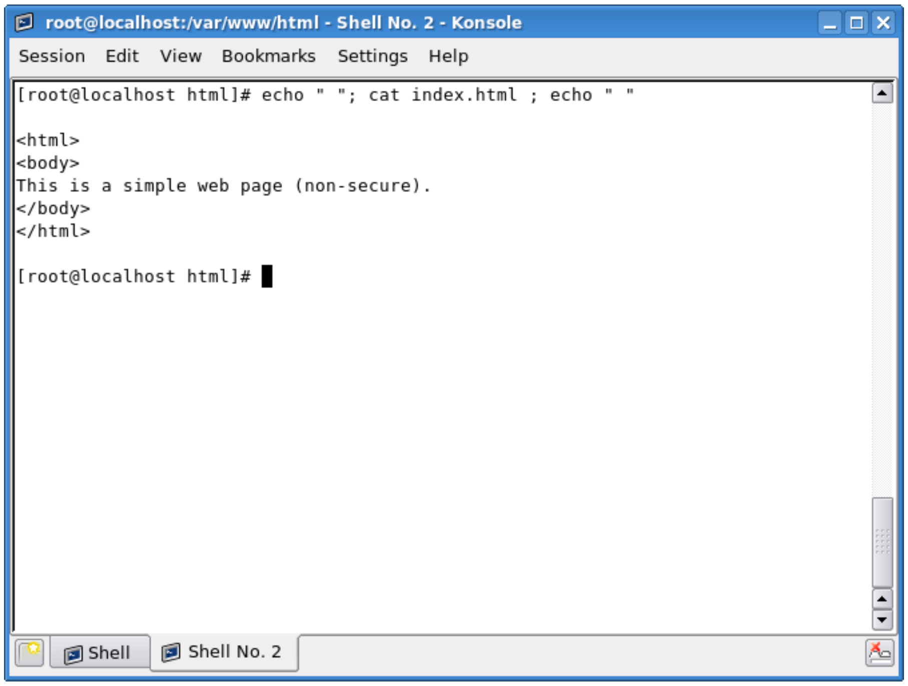

root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, in addition to the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses (0.0.0.0)
#
#Listen 12.34.56.78:80
Listen 80

#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding `LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Statically compiled modules (those listed by `httpd -l') do not need
# to be loaded here.
#
# Example:
# LoadModule foo_module modules/mod_foo.so
#
LoadModule auth_basic_module modules/mod_auth_basic.so
-- INSERT --
```

That's it. These are all the settings you need to know for now and tamper with in order to successfully launch the Web server. Save the configuration file (***Esc*** then ***:x*** in ***vi*** text editor).

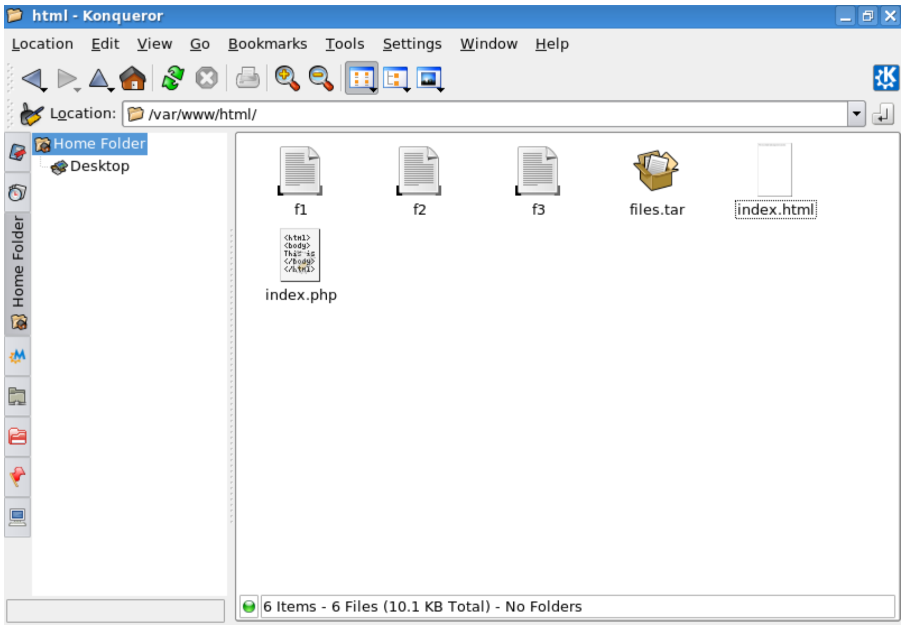
2.6 Create your HTML documents

Now, just to make things more interesting, we shall create a number of files and place them in the *DocumentRoot* directory (*/var/www/html*), including a simple index.html file. Here's the source of our *index.html* file (the two echoes are used to make the output easier to read):


root@localhost:/var/www/html - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
[root@localhost html]# echo " "; cat index.html ; echo " "

<html>
<body>
This is a simple web page (non-secure).
</body>
</html>

[root@localhost html]#
```

And here is the preview of files we have in the *html* directory:


html - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: /var/www/html/
Home Folder
Desktop
Home Folder
f1
f2
f3
files.tar
index.html
index.php
6 Items - 6 Files (10.1 KB Total) - No Folders

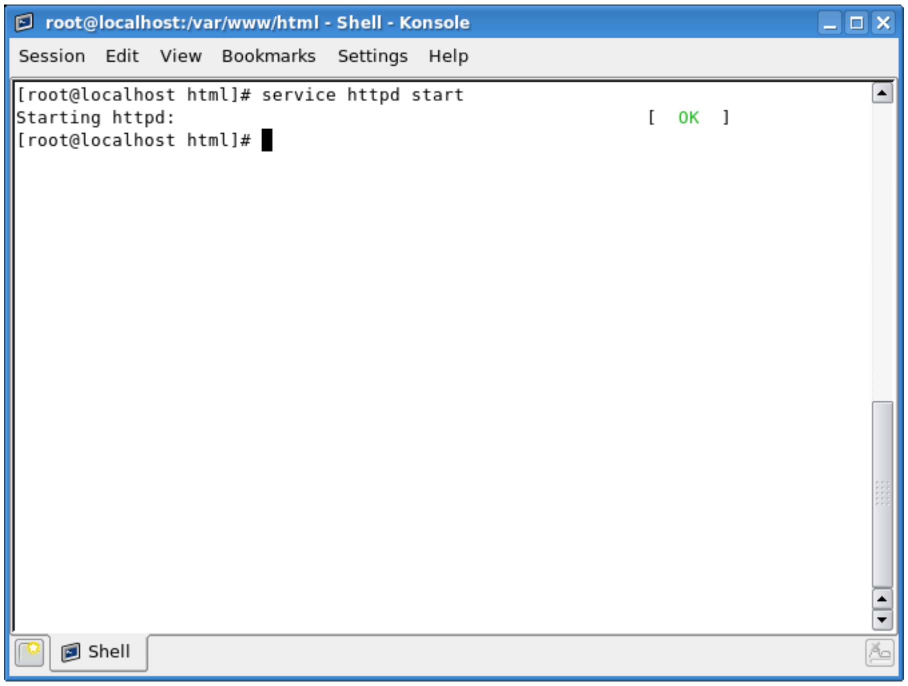
Now that we know what we have, it's time to power up the server.

2.7 Start the Web Server

Start the *httpd* service:

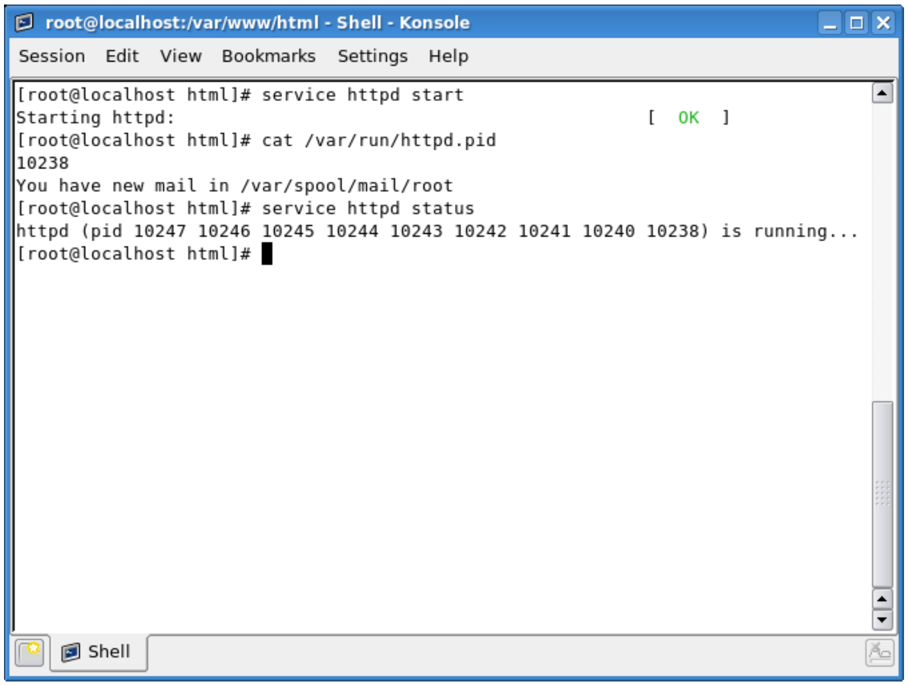
```
service httpd start
```

If everything worked out fine, the web server should start without any errors and you should see the following image:


root@localhost:/var/www/html - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@localhost html]# service httpd start
Starting httpd:                                            [  OK  ]
[root@localhost html]# 
```

Still, it does not hurt to check the status of the service or verify its process ID:

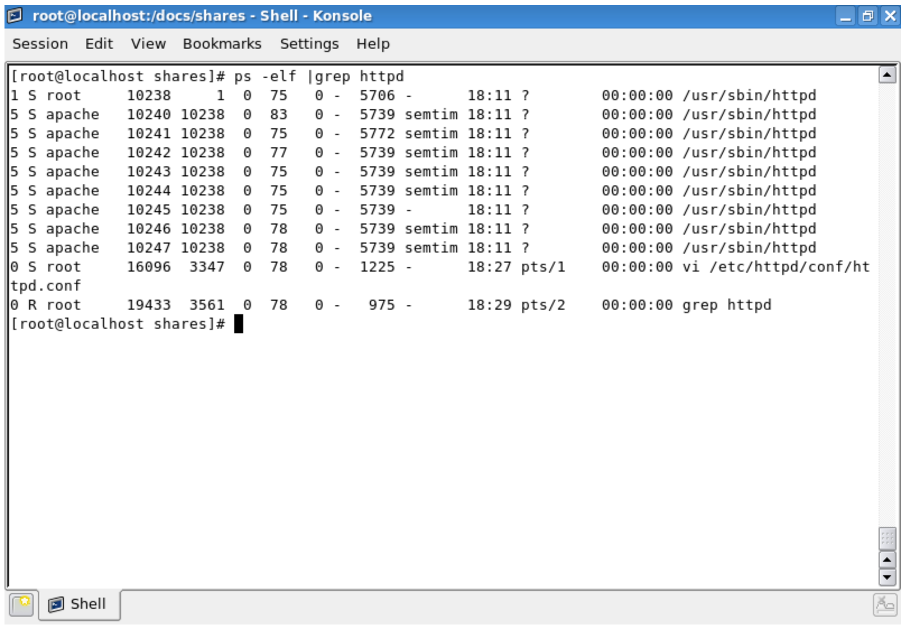

root@localhost:/var/www/html - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@localhost html]# service httpd start
Starting httpd:                                            [  OK  ]
[root@localhost html]# cat /var/run/httpd.pid
10238
You have new mail in /var/spool/mail/root
[root@localhost html]# service httpd status
httpd (pid 10247 10246 10245 10244 10243 10242 10241 10240 10238) is running...
[root@localhost html]#
```

There are 9 processes running for Apache. This may be confusing, but there's a very simple explanation for this. In the *httpd.conf* file, you will find a directive called *StartServers*. This directive tells the Web server how many server processes to launch on startup. The default setting is 8 server processes.

Once started, the Web server dynamically kills and creates processes based on the traffic load, with the number of server processes fluctuating between *MinSpareServers* and *MaxSpareServer*s. So far, everything figures out just nicely. Now, let's make another check.

The Apache Web server, if configured to listen on port 80 (or any "secure" port below 1024) must be started as root. Otherwise, it can also be started by regular (non-root) users. As a security precaution, the server processes spawned by Apache run as user *apache*, which belongs to the group *apache*. Indeed, we can easily verify that:


root@localhost:/docs/shares - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@localhost shares]# ps -elf |grep httpd
1 S root     10238     1  0  75   0 -  5706 -      18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10240 10238  0  83   0 -  5739 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10241 10238  0  75   0 -  5772 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10242 10238  0  77   0 -  5739 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10243 10238  0  75   0 -  5739 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10244 10238  0  75   0 -  5739 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10245 10238  0  75   0 -  5739 -      18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10246 10238  0  78   0 -  5739 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
5 S apache   10247 10238  0  78   0 -  5739 semtim 18:11 ?        00:00:00 /usr/sbin/httpd
0 S root     16096  3347  0  78   0 -  1225 -      18:27 pts/1    00:00:00 vi /etc/httpd/conf/ht
tpd.conf
0 R root     19433  3561  0  78   0 -   975 -      18:29 pts/2    00:00:00 grep httpd
[root@localhost shares]#
```

You can change these settings in the *httpd.conf* file, as well.

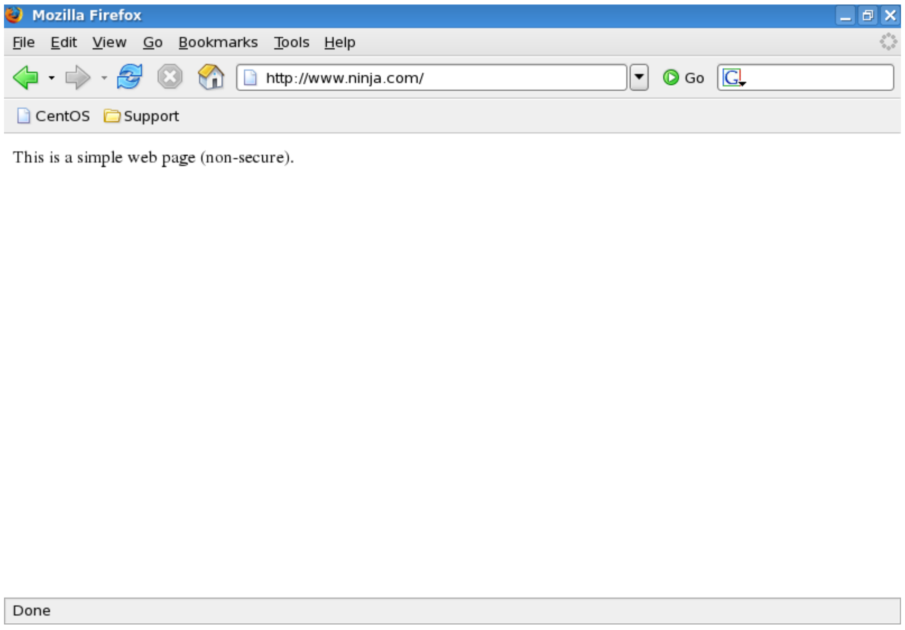
2.8 Access the web site

2.8.1 Local access

Now, let's access our homepage.

Open a web browser and type *www.ninja.com* in the address bar. Earlier, we were unable to access it, even though we have specified the entry for our website in the *hosts* file. This was because the

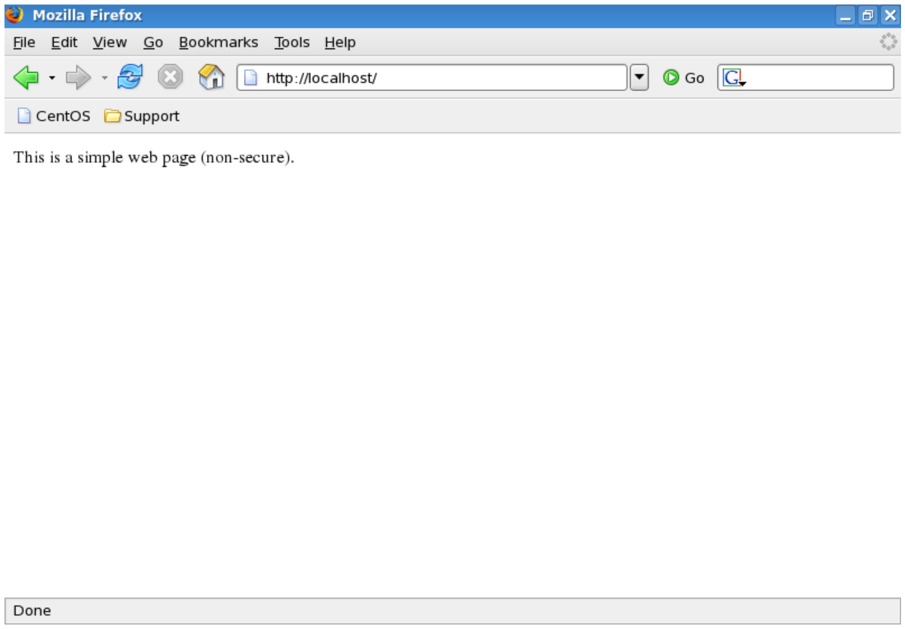
server was not running. But now, *www.ninja.com* resolves to our custom webpage.



This is a simple web page (non-secure).

Our server works.

Alternatively, we could have simply accessed it by typing *localhost* in the web browser address bar.

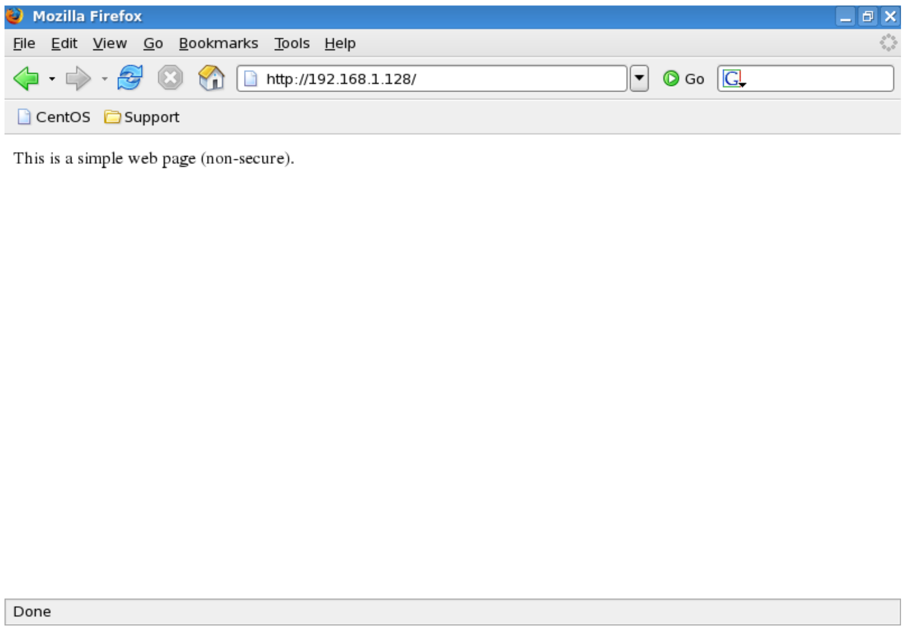


This is a simple web page (non-secure).

2.8.2 External access

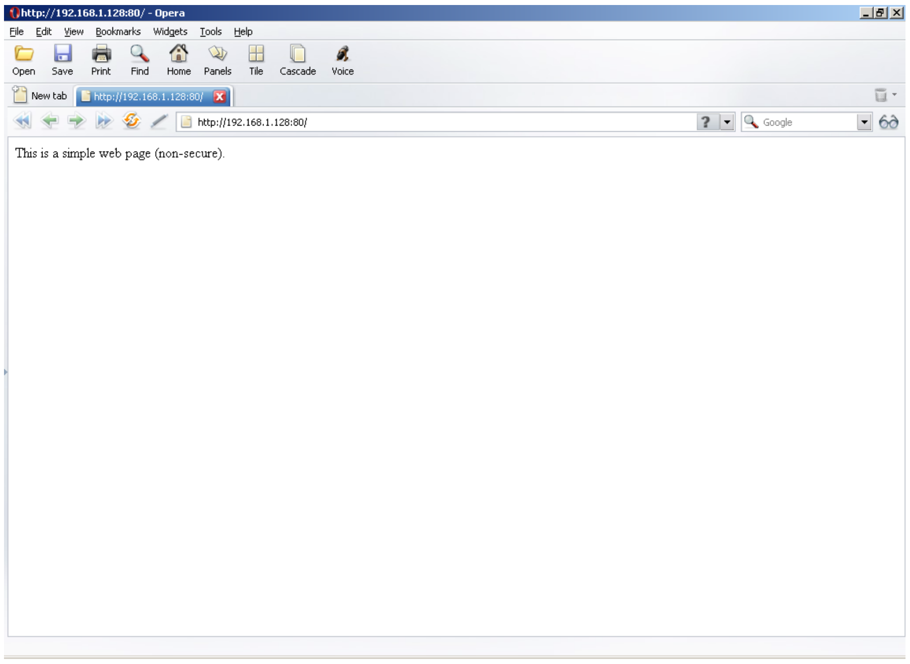
Accessing the web server locally is not the most challenging thing to do. Let's access it from other machines.

Here's our webpage, seen in another CentOS machine, belonging to the same subnet:



This is a simple web page (non-secure).

Here's our webpage, seen in the Opera browser running on a Windows machine:

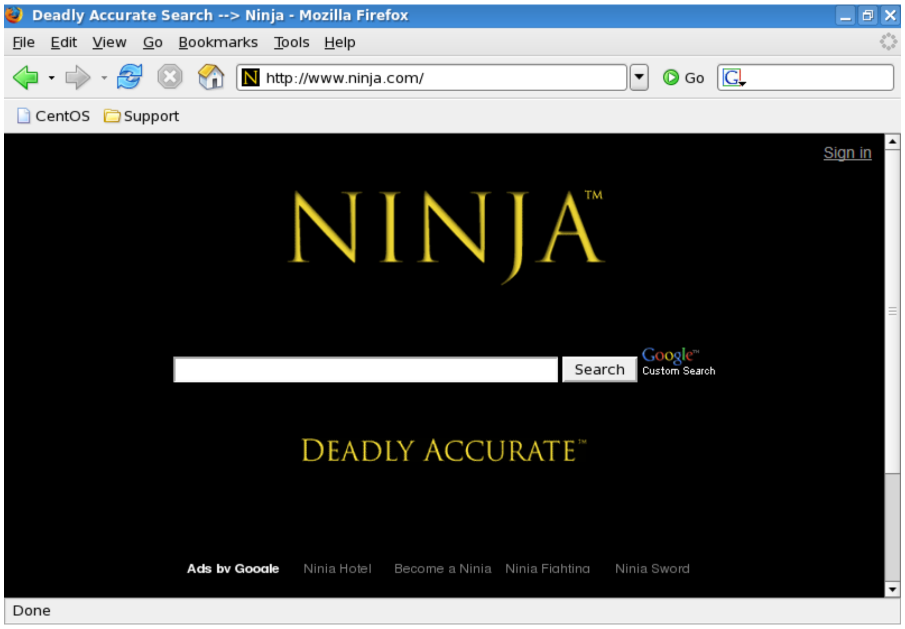

http://192.168.1.128:80/ - Opera
File Edit View Bookmarks Widgets Tools Help
Open Save Print Find Home Panels Tile Cascade Voice
New tab
http://192.168.1.128:80/
Google

This is a simple web page (non-secure).

As you might expect, using ***www.ninja.com*** on the other machines does not yield the desired result. This is because there is nothing telling these machines to match the name to the IP address of our server (192.168.1.128). On the server itself, we overcame the problem using the ***hosts*** file.

Theoretically, we could do the same thing on every host on our network, but this is slightly impractical and cumbersome. However, this will not solve the problem of accessibility from hosts that we have no control of, outside our local network. To overcome this monumental problem, we'll use name resolution by configuring and

running a Domain Name System (DNS) server. This material is covered in great detail in the next Part.



Meanwhile, everything works as we've expected. Soon, we will go over some advanced configurations.

2.9 Summary of basic setup

To make things simple and clear, here's an overview of the steps you will have to take to setup and launch Apache:

- Verify installation of the Apache RPM.
- Backup the */etc/httpd/conf/httpd.conf* main configuration file.
- Open it in the *vi* text editor and review the options listed therein.
- Setup the *DocumentRoot* directive (default */var/www/html*).
- Setup the *ServerName* directive (for example, *www.ninja.com*).
- Optionally setup other directives (like *ServerRoot*, *ErrorLog*, *Listen* etc).
- Configure the */etc/hosts* file so that you can access the website by name.
- Create a sample HTML file and place it in the *DocumentRoot* directory.
- Start the *httpd* server.
- Test the setup by accessing the web site.

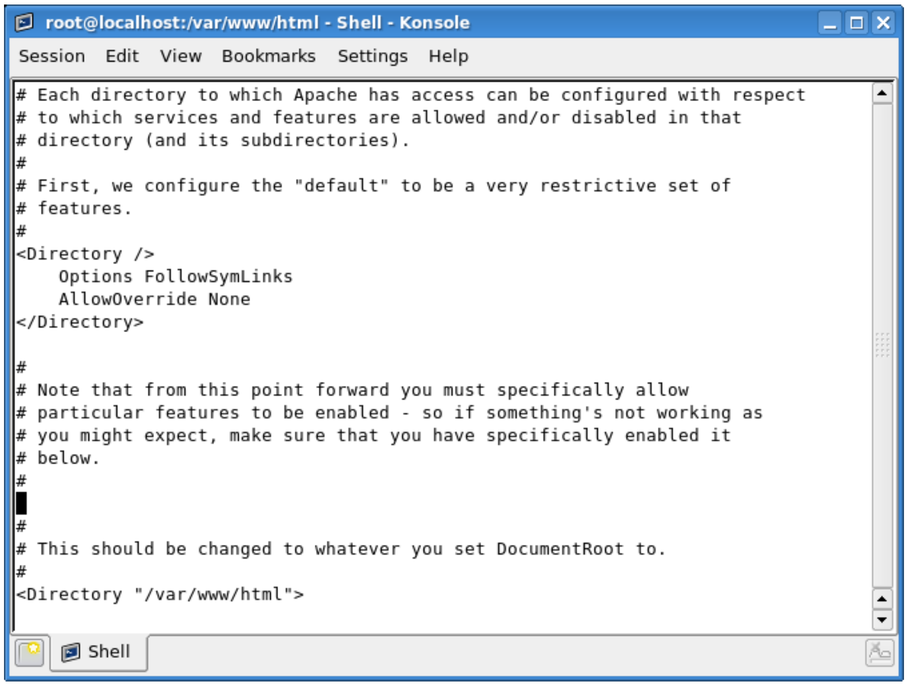
Chapter 3

Advanced setup

The ***httpd.conf*** file can be extensively customized using a range of directives. We have studied a few and will now review several more. Please note that it is impossible to list every single directive here. Nevertheless, we will go over some of the more useful and practical directives, which will greatly enhance the usability (and also the security) of your web server.

3.1 Directory tags

Directory tags allow you to specify the configurations separately for each directory serving the web pages. If you are familiar with HTML and CSS, then using <div> containers might be the simplest analogy. This allows you to serve content to specific IP ranges while denying other ranges, limit access to certain files, set the behavior of pages contained in these directories, and more.


root@localhost:/var/www/html - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
#
# First, we configure the "default" to be a very restrictive set of
# features.
#
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

#
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
#

#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory "/var/www/html">
```

Just about any directory can be listed, although it is not necessary. The most sensible solution is to setup very restrictive parameters to the root (/) directory and custom, desired parameters to directories inside *DocumentRoot*.

***Directory* tags take the following form (again this is very analogous to HTML <div> tags):**

- *<Directory directory_path>* tag begins a block.
- Next, follows a series of options defining what users accessing web pages located in this directory can do.
- *</Directory>* tag closes the block.

Here's a sample block, showing the default settings applied to the root (/) directory:

```
<Directory />
    Options FollowSymLinks
    AllowOverride none
</Directory>
```

Let's try to understand what we have here:

```
<Directory />
```

This declares the block for the root (/) directory and all sub-directories.

```
Options FollowSymLinks
```

The *Options* directive declares which server features are valid for the specified directory; *FollowSymLinks* is one of the possible options - it allows webpages to use symbolic links to point to files located anywhere on the root (/) directory. Please note this is not the best configuration from the security point of view; however, it does demonstrate the functionality of the *Directory* tags. We will discuss the server security measures later in the Part.

```
AllowOverride none
```

The *AllowOverride* directive governs the behavior of *.htaccess* files (more about them later). It tells whether the restrictions imposed by the *Options* can be overridden by specific settings inside the *.htaccess* files. The default behavior is set to *none* and should

remain that way. This will prevent security breaches or nuisances due to misconfiguration.

```
</Directory>
```

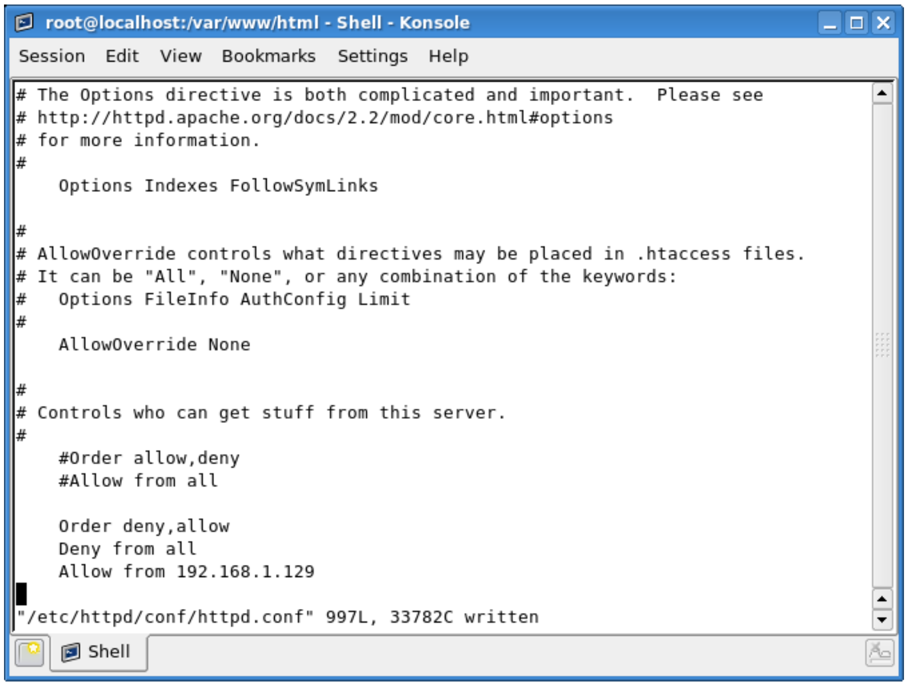
This tag closes the block.

3.1.1 Order (allow, deny)

Allow **and** ***Deny*** **directives govern the access to the directory declared (via the** ***Directory*** **tags). The** ***Order*** **directive specifies how the** ***allow*** **and** ***deny*** **directives are treated. The** ***Order*** **of** ***allow*****,** ***deny*** **can be looked upon as default-allow or blacklist; only "bad" hosts or IPs are disallowed. The** ***Order*** **of** ***deny*****,** ***allow*** **can be looked upon as default-deny or whitelist; only "good" hosts or IPs are allowed.**

Possible declaration of allowed or denied clients can be via host name, domain name, IP address, partial IP address, and more.

Here, we'll restrict access to the directory (or rather, the server) by denying access from all - and only permitting access from a single IP address, that of another machine on the LAN (in this case, 192.168.1.129).


root@localhost:/var/www/html - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# The Options directive is both complicated and important.  Please see
# http://httpd.apache.org/docs/2.2/mod/core.html#options
# for more information.
#
    Options Indexes FollowSymLinks

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
    AllowOverride None

#
# Controls who can get stuff from this server.
#
    #Order allow,deny
    #Allow from all

    Order deny,allow
    Deny from all
    Allow from 192.168.1.129

"/etc/httpd/conf/httpd.conf" 997L, 33782C written
```

Let's review the changes to the *httpd.conf* file:

- I have commented out the original parameters, which allowed access from all hosts (or IPs).

- I have changed the order of *allow*, *deny* directives. Again, this is important, because the order defines the precedence of the rules. Thus, first, we'll deny everyone (this can be called default deny policy, so to speak) and then permit only specific hosts (or IPs). If the *Order* were reversed (*allow*, *deny* rather than *deny*, *allow*), no one would be able to access the server. This is critically important to remember when implementing *allow*, *deny* policies.

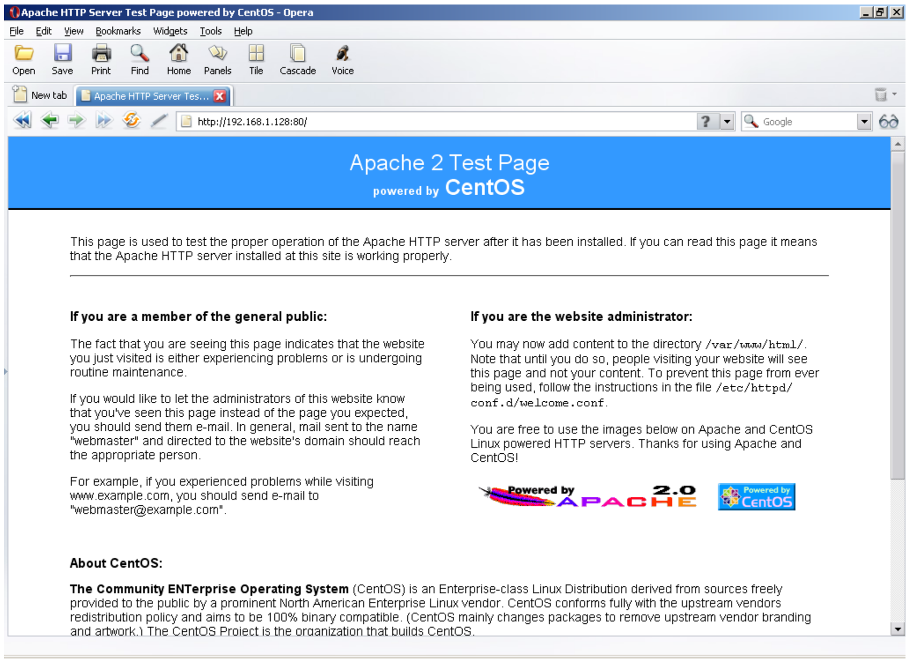
The changes will only take effect after the Web server is restarted or the configuration file reloaded. This can be achieved by running either:

```
service httpd restart
```

Or:

```
service httpd reload
```

After *httpd* reads the new configuration file, the changes will take effect. Now, let's try to access the server from the Windows machine.


Apache HTTP Server Test Page powered by CentOS - Opera
File Edit View Bookmarks Widgets Tools Help
Open Save Print Find Home Panels Tile Cascade Voice
New tab
Apache HTTP Server Tes...
http://192.168.1.128:80/
Google
Powered by APACHE 2.0
Powered by CentOS

Apache 2 Test Page
powered by CentOS

This page is used to test the proper operation of the Apache HTTP server after it has been installed. If you can read this page it means that the Apache HTTP server installed at this site is working properly.

If you are a member of the general public:

The fact that you are seeing this page indicates that the website you just visited is either experiencing problems or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

If you are the website administrator:

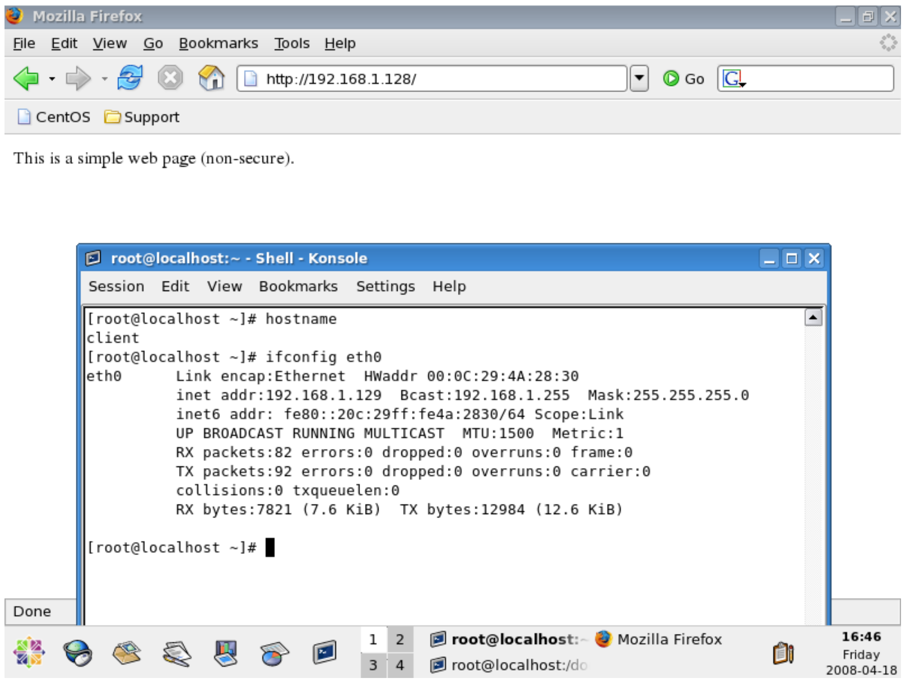
You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

You are free to use the images below on Apache and CentOS Linux powered HTTP servers. Thanks for using Apache and CentOS!

About CentOS:

The Community ENTerprise Operating System (CentOS) is an Enterprise-class Linux Distribution derived from sources freely provided to the public by a prominent North American Enterprise Linux vendor. CentOS conforms fully with the upstream vendors redistribution policy and aims to be 100% binary compatible. (CentOS mainly changes packages to remove upstream vendor branding and artwork.) The CentOS Project is the organization that builds CentOS.

As you can see, we are denied access. But accessing from the CentOS client with the IP of 192.168.1.129 works fine.



This is a simple web page (non-secure).

```
[root@localhost ~]# hostname
client
[root@localhost ~]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0C:29:4A:28:30
          inet addr:192.168.1.129  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe4a:2830/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:82 errors:0 dropped:0 overruns:0 frame:0
          TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:7821 (7.6 KiB)  TX bytes:12984 (12.6 KiB)

[root@localhost ~]#
```

3.1.2 Indexes

The *Indexes* directive tells the server whether to display the directory listing when asked. The behavior of this directive depends on another directive - the *DirectoryIndex*. The *DirectoryIndex* directive tells the server the name of the default page that it should serve when a user requests the listing of a directory.

This is the typical everyday scenario. Users are trying to access webpages by simply typing their names, without typing the exact homepage (like *index.html*, *index.php* etc). Various file names

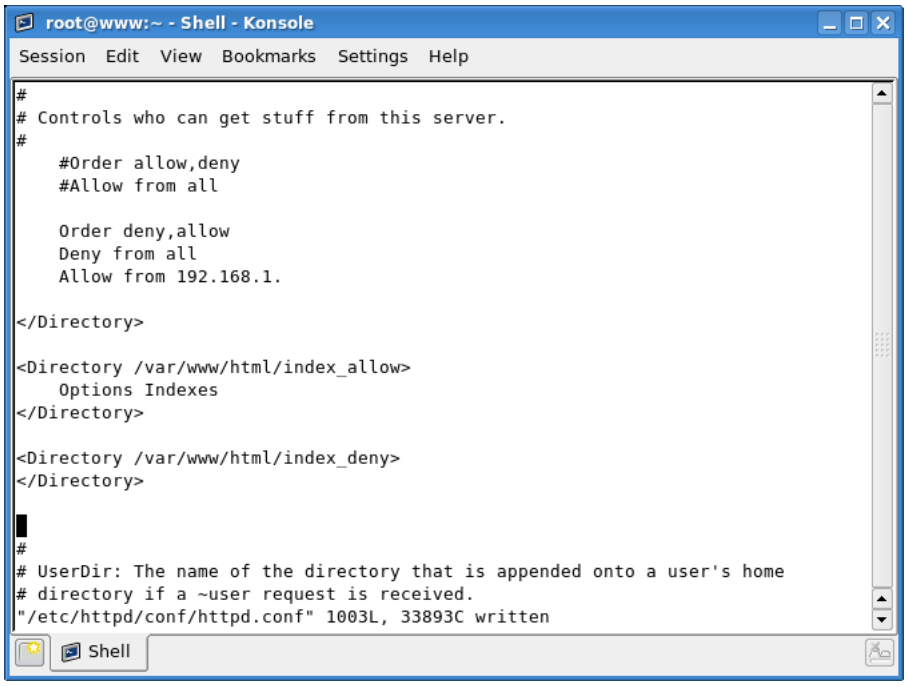
specified under the *DirectoryIndex* are looked for and the first one found is presented to the user. If no file is found, the listing of the directory is then generated by the server.

This is something you may want to avoid, especially if there are files you do not wish your users to see. However, if the *Options Indexes* directives are used, then directory listings will be generated.

One solution is to place a dummy *index.html* file in every directory, but this is cumbersome. The more elegant approach is to disable the listing globally (remove Indexes from the *Options* directive under *DocumentRoot*) and then allow per-directory listing when you see fit.

The default configuration in the *httpd.conf* file specifies *Options Indexes* for the *Directory* tags of the default *DocumentRoot* (*/var/www/html*). We will change that.

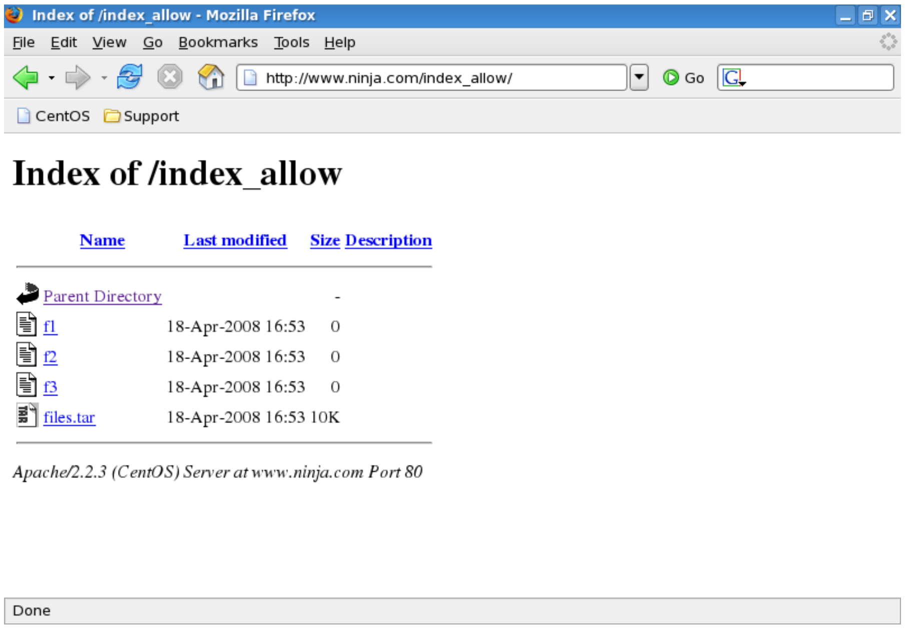
First, we will remove *Indexes* from the *Options* line for our *DocumentRoot.* Then, we will create two directories, called *index_allow* and *index_deny*, where only the first will have the *Options Indexes* specified. Both of these directories will contain some random files.


root@www:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
#
# Controls who can get stuff from this server.
#
    #Order allow,deny
    #Allow from all

    Order deny,allow
    Deny from all
    Allow from 192.168.1.

</Directory>

<Directory /var/www/html/index_allow>
    Options Indexes
</Directory>

<Directory /var/www/html/index_deny>
</Directory>


#
# UserDir: The name of the directory that is appended onto a user's home
# directory if a ~user request is received.
"/etc/httpd/conf/httpd.conf" 1003L, 33893C written
```

This is the new configuration file. Save it, then restart *httpd*. Now, if we request the directory listing for each one from our clients, we'll get the following results:

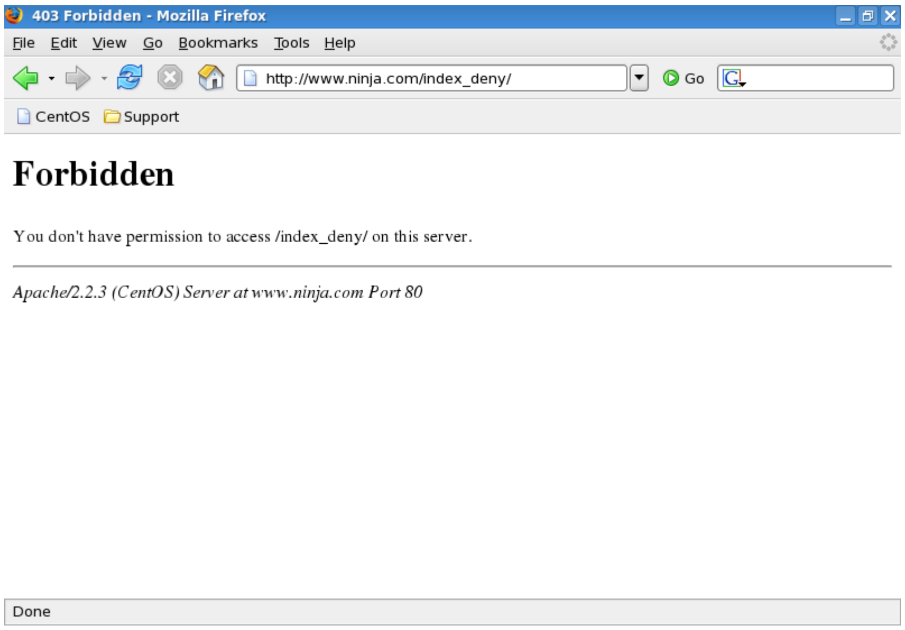
index_allow


Index of /index_allow - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja.com/index_allow/
Go
CentOS Support
Done

Index of /index_allow

```
      Name          Last modified      Size Description
----------------------------------------------------
Parent Directory                          -
f1               18-Apr-2008 16:53    0
f2               18-Apr-2008 16:53    0
f3               18-Apr-2008 16:53    0
files.tar        18-Apr-2008 16:53 10K
----------------------------------------------------
Apache/2.2.3 (CentOS) Server at www.ninja.com Port 80
```

index_deny


403 Forbidden - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja.com/index_deny/
Go
CentOS
Support
Done

Forbidden

You don't have permission to access /index_deny/ on this server.

Apache/2.2.3 (CentOS) Server at www.ninja.com Port 80

3.1.3 DirectoryMatch

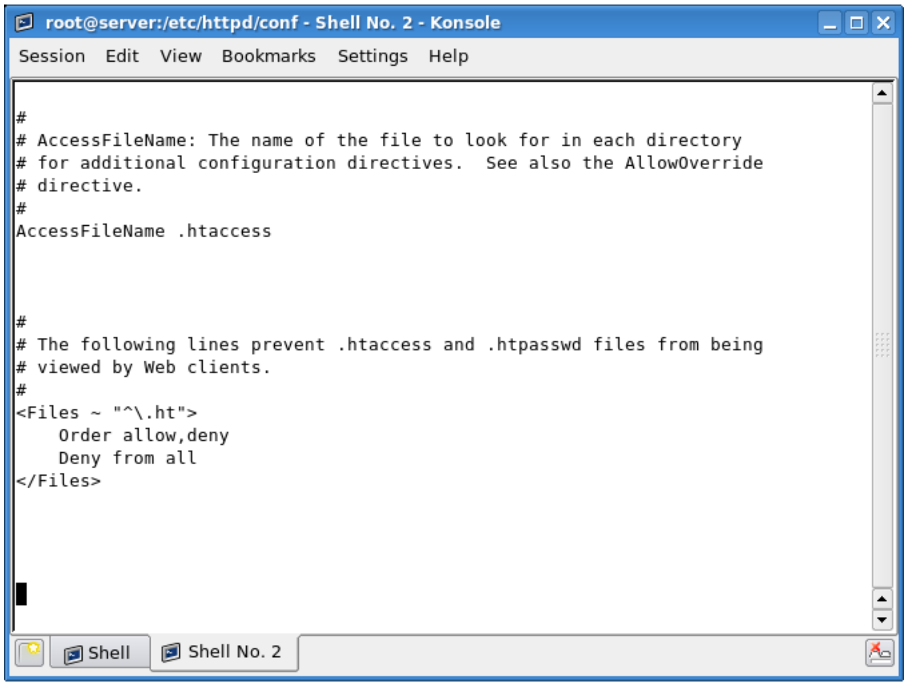
The directives enclosed in the *Directory* tags will be indiscriminately applied to all sub-directories. If you require a more fine-tuned approach for several similar sub-directories, you will have to use the DirectoryMatch tags. The main difference is that the DirectoryMatch tags allow the use of regular expressions, allowing you to match several sub-directories inside a single rule.

Again, for those familiar with HTML / CSS and the use of classes and ids, the idea is very much similar.

3.2 Files tags

The *Files* tags are very similar to the *Directory* tags. The major difference is that while the *Directory* tags govern the scope of permissions (or restrictions) of the enclosed directives by directory name, the *Files* tags do the same on the file name level. In other words, the *Files* tags can be used to configure the behavior of a single file - or a set of files that match a regular expression.

Here's an example, showing the restrictions applied to *.htaccess* and *.htpasswd* files, the files usually used in restricting access to certain directories (and/or files) by requiring users to authenticate before viewing the content:


root@server:/etc/httpd/conf - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
#
# AccessFileName: The name of the file to look for in each directory
# for additional configuration directives.  See also the AllowOverride
# directive.
#
AccessFileName .htaccess



#
# The following lines prevent .htaccess and .htpasswd files from being
# viewed by Web clients.
#
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

We will review this particular example later in this Part.

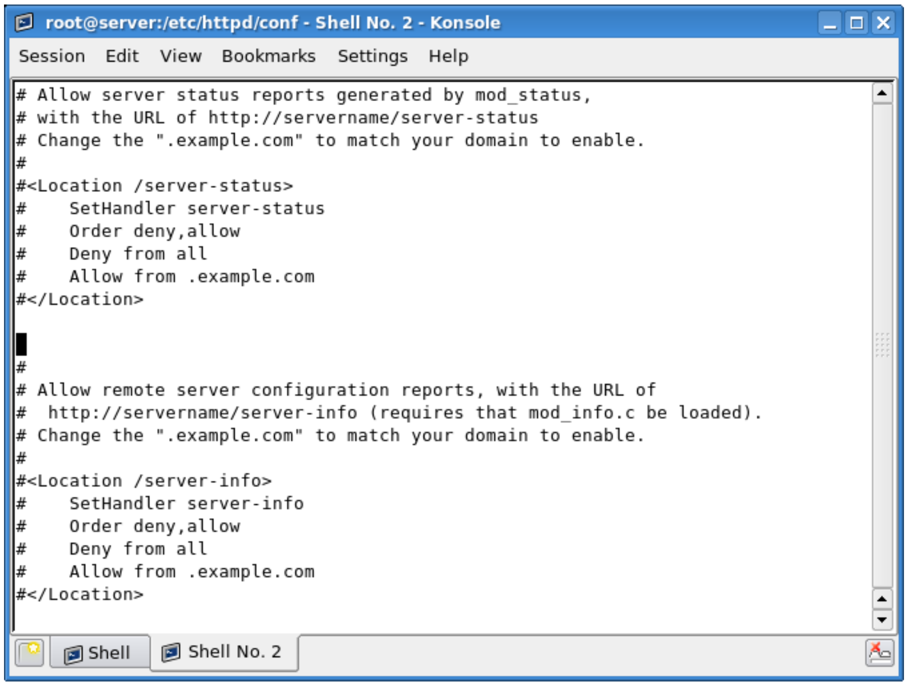
In the above example, we've seen the use of regular expressions to allow multiple files to be covered by a single rule. However, a comparable directive, more suitable for handling multiple files and complex regular expressions is the ***FilesMatch*** directive.

3.3 Location tags

Again, the ***Location*** tags are quite similar to the two mentioned above. The major difference is that the ***Location*** tags are used to limit the scope of enclosed directives by URLs.

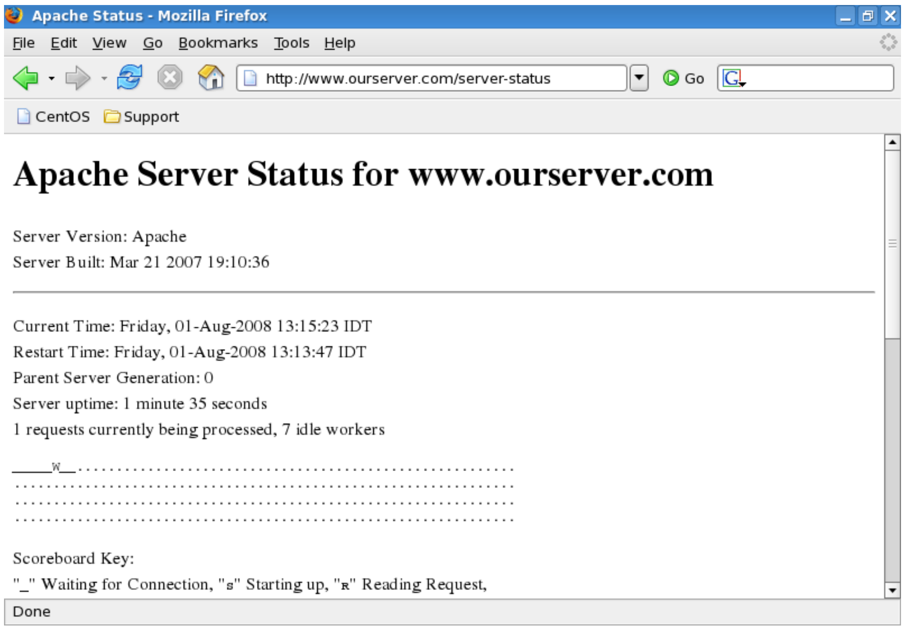
In other words, the ***Directory*** and ***Files*** tags should be used to control content that resides on the system (like various files and images, within their sub-directories), while the ***Location*** tags should be used to control content that is located ***outside*** the system, like databases, for instance.

Below, we can see a commented example included in the ***httpd.conf*** file. If enabled, this block would allow you to access your server statistics, but only if you connected from the server itself.


root@server:/etc/httpd/conf - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
# Allow server status reports generated by mod_status,
# with the URL of http://servername/server-status
# Change the ".example.com" to match your domain to enable.
#
#<Location /server-status>
#    SetHandler server-status
#    Order deny,allow
#    Deny from all
#    Allow from .example.com
#</Location>


#
# Allow remote server configuration reports, with the URL of
#  http://servername/server-info (requires that mod_info.c be loaded).
# Change the ".example.com" to match your domain to enable.
#
#<Location /server-info>
#    SetHandler server-info
#    Order deny,allow
#    Deny from all
#    Allow from .example.com
#</Location>
```

Here's an example (please disregard the actual URL):


Apache Status - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/server-status
Go
CentOS
Support
Done

```
Apache Server Status for www.ourserver.com

Server Version: Apache
Server Built: Mar 21 2007 19:10:36

Current Time: Friday, 01-Aug-2008 13:15:23 IDT
Restart Time: Friday, 01-Aug-2008 13:13:47 IDT
Parent Server Generation: 0
Server uptime: 1 minute 35 seconds
1 requests currently being processed, 7 idle workers

_____W__.........................................................
.................................................................
.................................................................
.................................................................

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
```

Again, for complex regular expressions, you should use the ***LocationMatch*** directive.

3.4 Directory, Files and Location

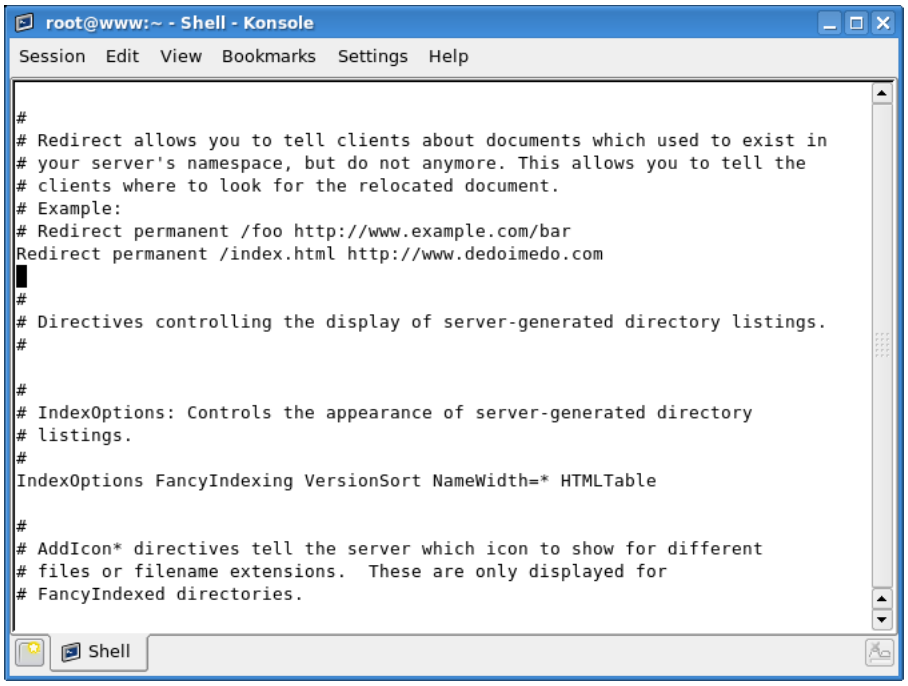
The ***Directory***, ***Files*** and ***Location*** tags all perform a similar function: they categorize what restrictions are placed on content enclosed by each one. At first glance, there seems to be very little difference between them. However, just like the order of ***allow*** and ***deny*** directives is critical, so is the correct use of these tags.

The configuration sections must be placed in a very particular order to make sure they behave as intended. The order of precedence of their execution by the server means that a misplaced section could compromise the security of the server - or not get executed at all.

For more details, please refer to the following Apache documentation page: Configuration Sections - Apache HTTP Server.

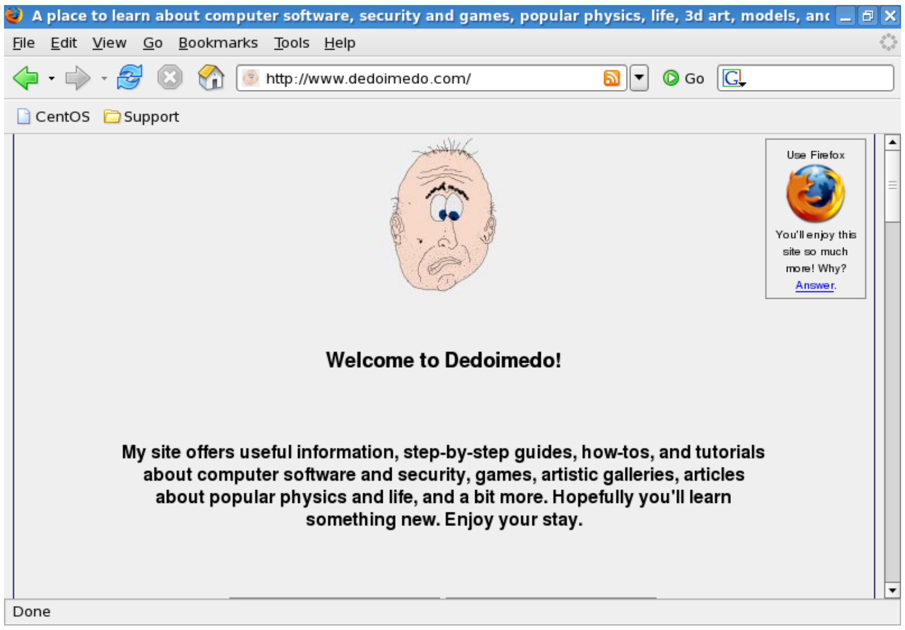
3.5 Redirect

The Redirect setting allows you to map an old webpage to a new URL. This could be the case if you changed domain, for example, or moved around a lot of files, renaming and deleting them. To demonstrate the directive, we'll map our server to point to my own site.


root@www:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
#
# Redirect allows you to tell clients about documents which used to exist in
# your server's namespace, but do not anymore. This allows you to tell the
# clients where to look for the relocated document.
# Example:
# Redirect permanent /foo http://www.example.com/bar
Redirect permanent /index.html http://www.dedoimedo.com

#
# Directives controlling the display of server-generated directory listings.
#


#
# IndexOptions: Controls the appearance of server-generated directory
# listings.
#
IndexOptions FancyIndexing VersionSort NameWidth=* HTMLTable


#
# AddIcon* directives tell the server which icon to show for different
# files or filename extensions.  These are only displayed for
# FancyIndexed directories.
```

Save the file, restart the server.

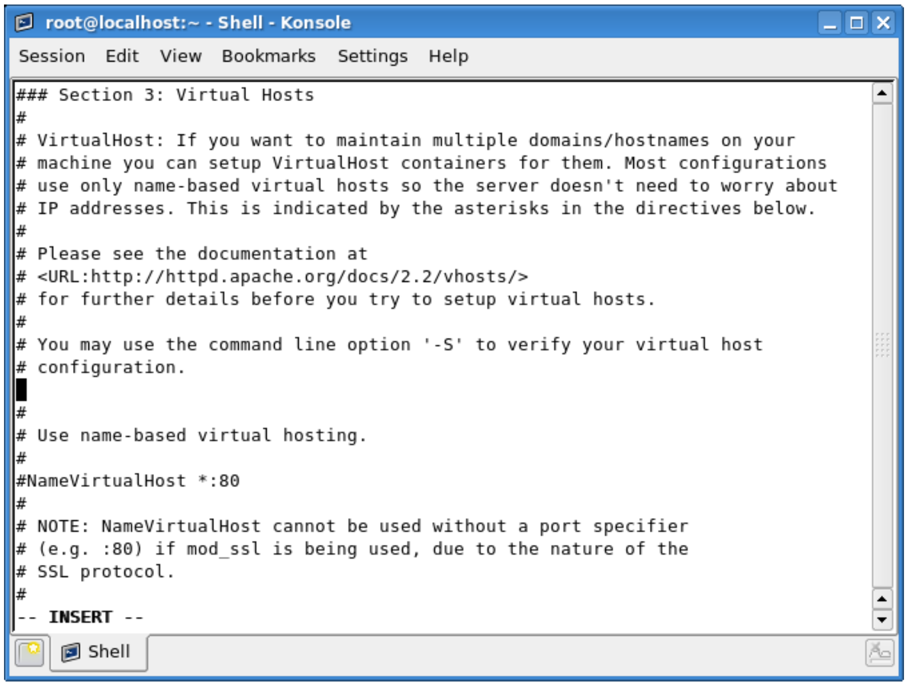

A place to learn about computer software, security and games, popular physics, life, 3d art, models, and
File Edit View Go Bookmarks Tools Help
http://www.dedoimedo.com/
Go
CentOS
Support
Use Firefox
You'll enjoy this site so much more! Why? Answer.
Done

Welcome to Dedoimedo!

My site offers useful information, step-by-step guides, how-tos, and tutorials about computer software and security, games, artistic galleries, articles about popular physics and life, and a bit more. Hopefully you'll learn something new. Enjoy your stay.

3.6 Virtual Hosts

Virtual Hosts **is an important, powerful feature that allows you to run several websites from a single computer.** ***Virtual Hosts*** **can be IP-based or named-based, offering a high level of customization (and flexibility).**


root@localhost:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
### Section 3: Virtual Hosts
#
# VirtualHost: If you want to maintain multiple domains/hostnames on your
# machine you can setup VirtualHost containers for them. Most configurations
# use only name-based virtual hosts so the server doesn't need to worry about
# IP addresses. This is indicated by the asterisks in the directives below.
#
# Please see the documentation at
# <URL:http://httpd.apache.org/docs/2.2/vhosts/>
# for further details before you try to setup virtual hosts.
#
# You may use the command line option '-S' to verify your virtual host
# configuration.

#
# Use name-based virtual hosting.
#
#NameVirtualHost *:80
#
# NOTE: NameVirtualHost cannot be used without a port specifier
# (e.g. :80) if mod_ssl is being used, due to the nature of the
# SSL protocol.
#
-- INSERT --
```

Virtual Hosts **can use almost any option normally used in the** ***httpd.conf*** **file. To make you better understand this, you can treat** ***Virtual Hosts*** **as individual customized** ***httpd.conf*** **files nested inside the main** ***httpd.conf*** **file.**

Let's review a sample *Virtual Host*:

```
<VirtualHost *:80>
    DocumentRoot /var/www/html/ninja-father
    ServerName www.ninja-father.com
    # other directives
</VirtualHost>
```

What do we have here?

```
<VirtualHost *:80>
```

This declares the name or the IP address of the site (server) that should be served using the directives inside the *VirtualHost* block on port 80. If no port number is used, the default one specified under the *Listen* option is used. The default port is 80 (standard convention). Asterisk (*) can be replaced with any name (for example, *www.ninja.com*) or IP address (for example, 192.168.1.128), depending on your needs and requirements. Let see several simple examples:

- *<VirtualHost 192.168.1.128:80>* will apply the directives listed in the block below to all incoming connections aimed at 192.168.1.128 on port 80.
- *<VirtualHost 192.168.1.128>* will apply the directives listed in the block below to all incoming connections aimed at 192.168.1.128 on the default port (as specified in the *Listen* directive). If this port is 80, then this option is identical to the one above.
- *<VirtualHost planck.matter.com>* will apply the directives listed in the block below to all incoming connections aimed at *planck.matter.com* on

the default port (which can be 80, 8080 or any other).

- *<VirtualHost ninja.com:8777>* will apply the directives listed in the block below to all incoming connections aimed at our site *ninja.com* on port 8777. This port must be specified under the *Listen* directive.

```
DocumentRoot /var/www/html/ninja-father
```

This declares the directory where you should place all files that you wish served when the *VirtualHost* is invoked (matching names or IPs and the port).

```
ServerName www.ninja-father.com
```

This is the name of the server. In other words, this is the address people will type in the web browser address name in order to get to your site. In order to successfully resolve this name to the IP address of the Web Server, we will need to use */etc/hosts* file like before or setup a DNS Server (later).

```
# other directives
```

This is just a comment specifying many other options can be used, including those we have not yet reviewed here.

OK, now that we know what we're dealing with, let's create and test several scenarios.

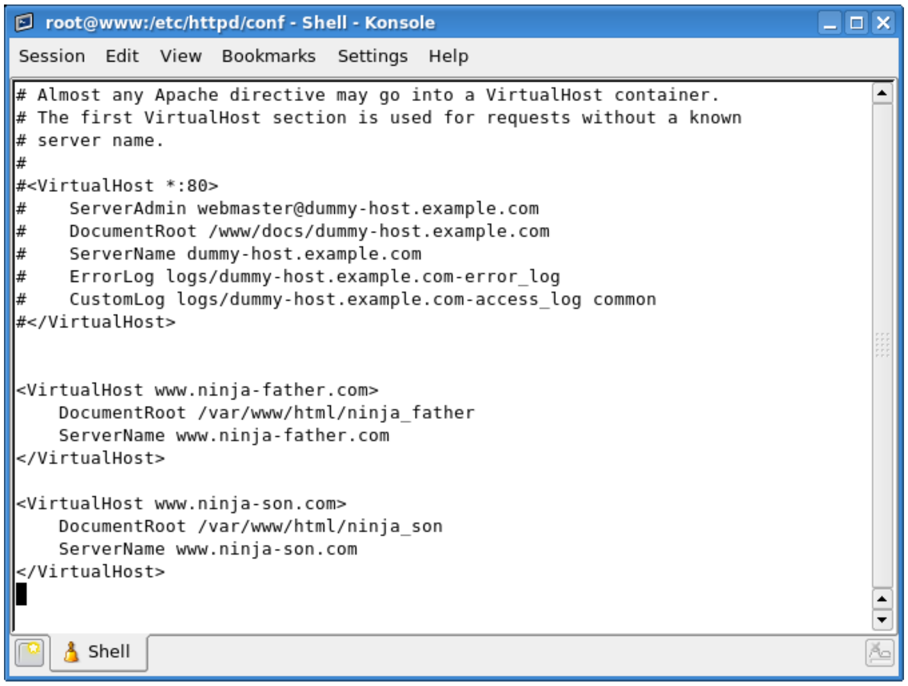
3.6.1 Single IP, two websites

This is one of the most common setups. We will create two websites - *www.ninja-father.com* and *www.ninja-son.com*. Both will reside on our server, which has the IP address 192.168.1.128. In order to make them both accessible to the world, we will create two *VirtualHost* blocks and declare their *DocumentRoot* and *ServerName* separately.

Here's what we need to do:

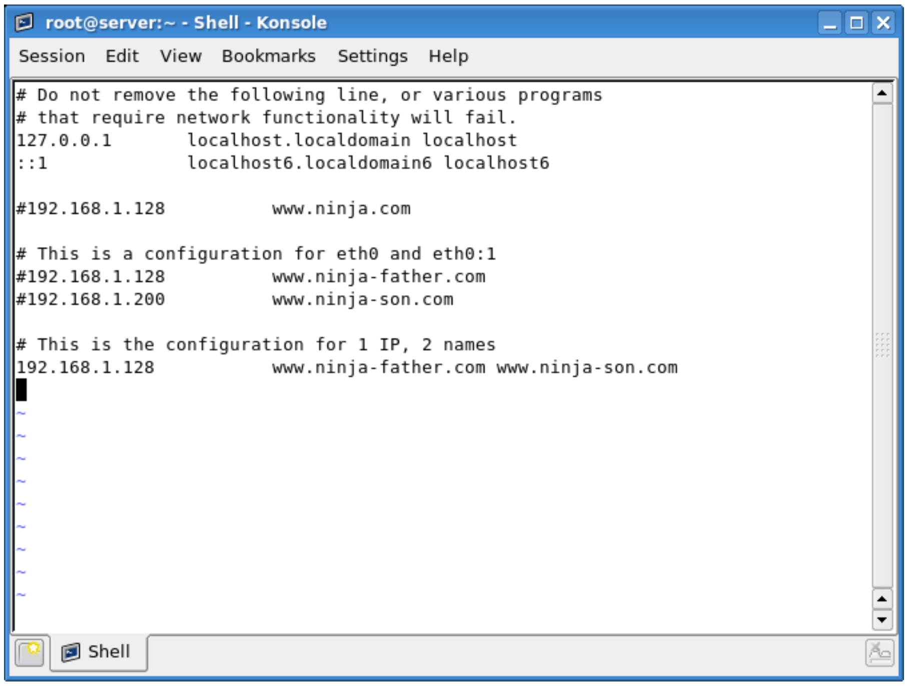
- Create directories inside */var/www/html* called *ninja-father* and *ninja-son.*
- Create simple *index.html* files for each.
- Edit *httpd.conf* and create our two *VirtualHost* blocks.

Here's what the *httpd.conf* looks like:


root@www:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# Almost any Apache directive may go into a VirtualHost container.
# The first VirtualHost section is used for requests without a known
# server name.
#
#<VirtualHost *:80>
#    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>


<VirtualHost www.ninja-father.com>
    DocumentRoot /var/www/html/ninja_father
    ServerName www.ninja-father.com
</VirtualHost>

<VirtualHost www.ninja-son.com>
    DocumentRoot /var/www/html/ninja_son
    ServerName www.ninja-son.com
</VirtualHost>
```

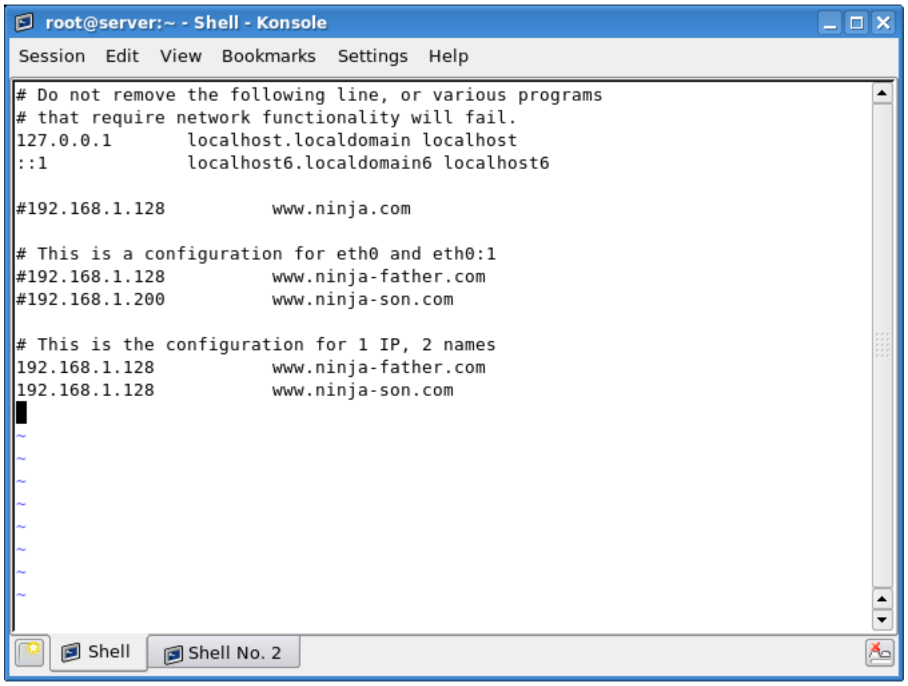
Now, for the sake of convenience, we will also use the */etc/hosts* file to allow name resolution to work. It is also imperative in our case, because using the IP address would always point to the first *VirtualHost* listed in the *httpd.conf* file.


root@server:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6

#192.168.1.128          www.ninja.com

# This is a configuration for eth0 and eth0:1
#192.168.1.128          www.ninja-father.com
#192.168.1.200          www.ninja-son.com

# This is the configuration for 1 IP, 2 names
192.168.1.128           www.ninja-father.com www.ninja-son.com

~
~
~
~
~
~
~
~
~
```

Please note that specifying an IP address in two different lines is wrong. The *hosts* file will always use only the first entry. You should list all names for a specific IP in a single line.

For example, this is incorrect (although it would work in our case):

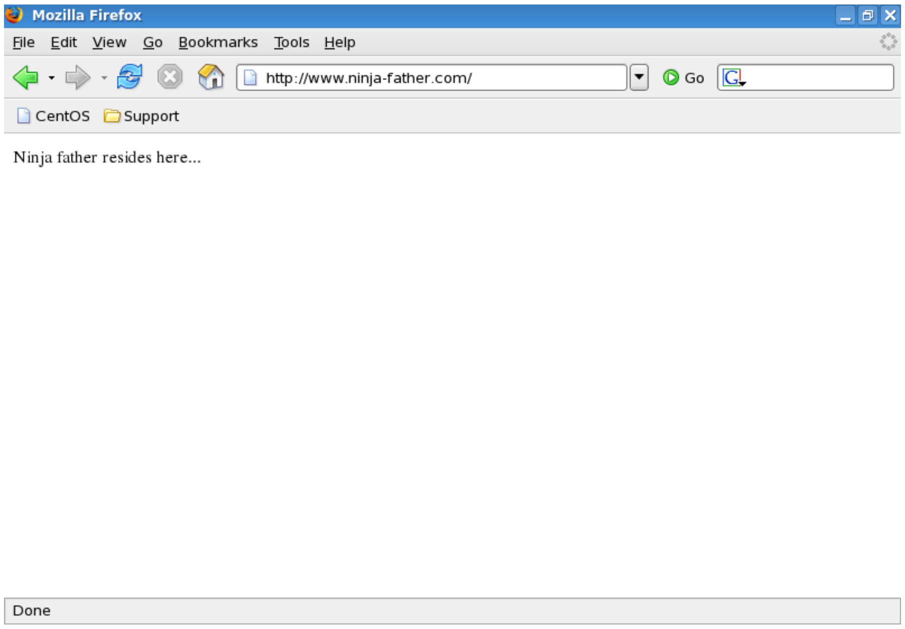

root@server:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6

#192.168.1.128          www.ninja.com

# This is a configuration for eth0 and eth0:1
#192.168.1.128          www.ninja-father.com
#192.168.1.200          www.ninja-son.com

# This is the configuration for 1 IP, 2 names
192.168.1.128           www.ninja-father.com
192.168.1.128           www.ninja-son.com

~
~
~
~
~
~
~
~
```

Don't mind the commented lines, they are used for other configurations: the first, our standard website; the second, for yet another *VirtualHost* scenario, which we will discuss soon.

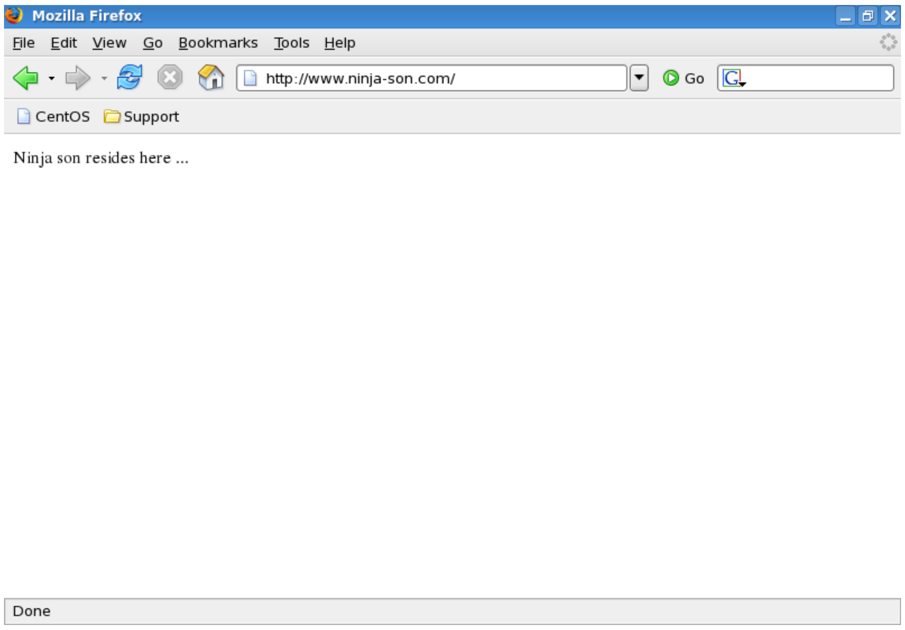
Now, we shall save the files (both *httpd.conf* and */etc/hosts*) and restart *httpd*. Then, using Firefox, we will try to access each one.

www.ninja-father.com


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja-father.com/
Go
CentOS
Support

Ninja father resides here...

www.ninja-son.com


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja-son.com/
Go
CentOS
Support
Done

Ninja son resides here ...

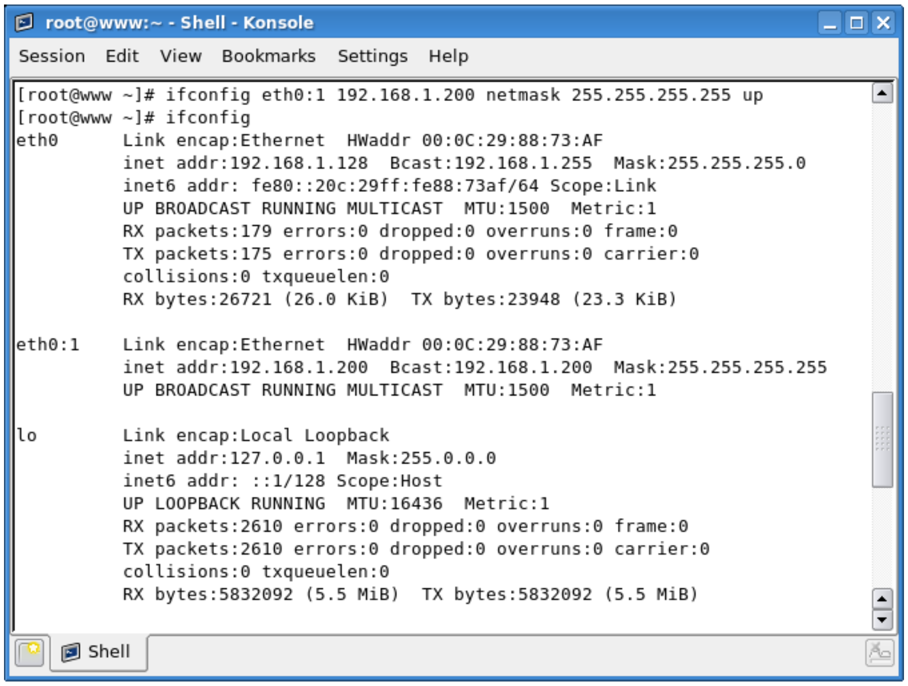
It works like magic. Best of all, the user has no idea that these two sites reside on the same machine.

3.6.2 Two IPs, two websites

This is another common scenario. You can assign a different IP to each website, avoid possible resolution mixups and simplifying your setup. However, this requires that you either use more than a single network adapter or create virtual adapters. If you have, let's say 14 websites, having 14 physical network devices plugged into your machines is not the best idea. Using virtual adapters is the most sensible choice here.

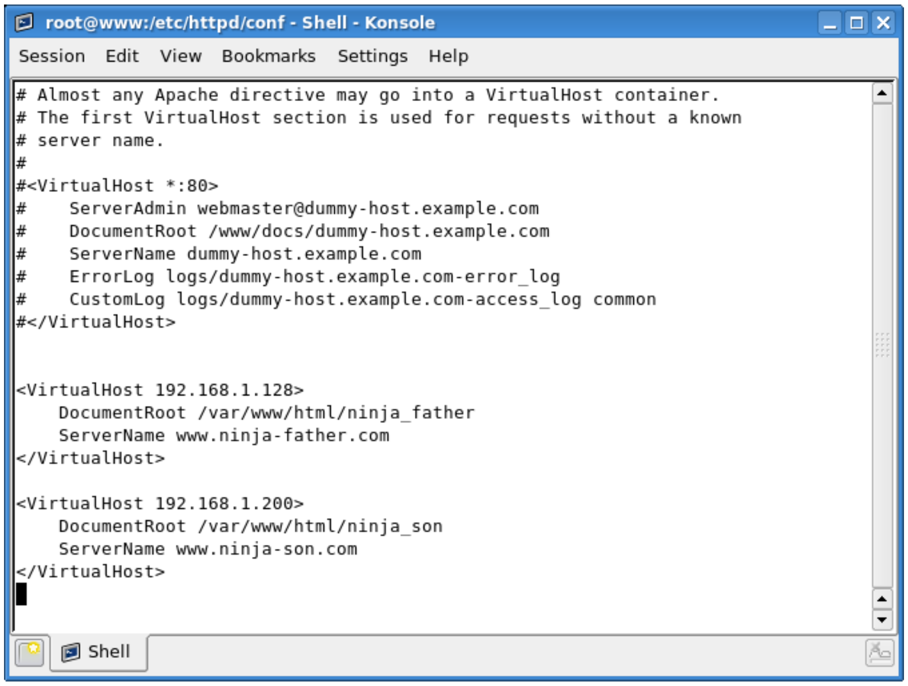
We already have our two websites ready. We just need to create a virtual network card and then change the *httpd.conf* file to reflect the changes.

First, we will create a virtual adapter (eth0:1) with the IP address of 192.168.1.200.


root@www:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

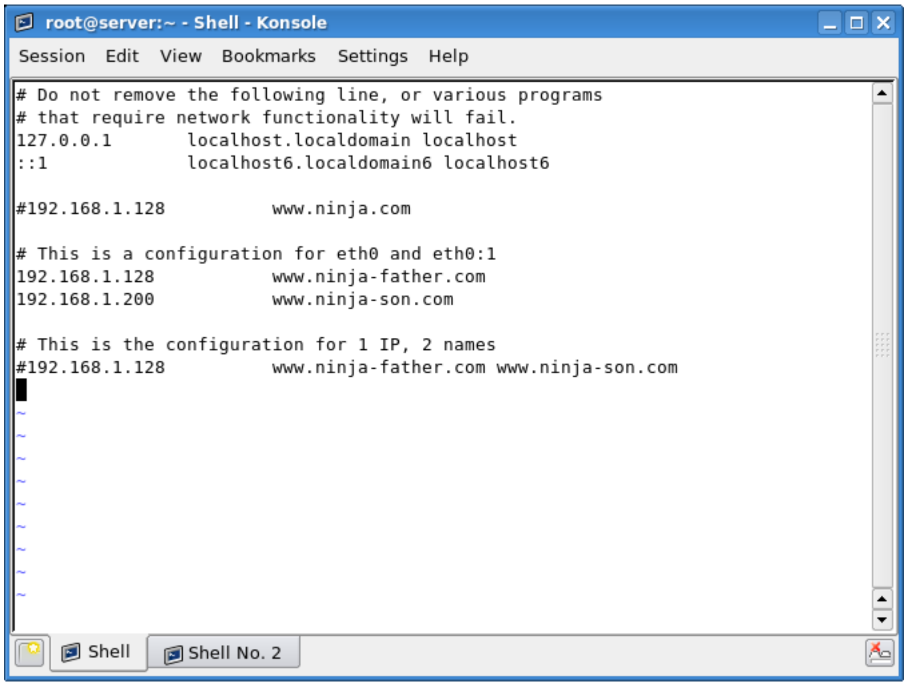
```
[root@www ~]# ifconfig eth0:1 192.168.1.200 netmask 255.255.255.255 up
[root@www ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:88:73:AF
          inet addr:192.168.1.128  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe88:73af/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:179 errors:0 dropped:0 overruns:0 frame:0
          TX packets:175 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:26721 (26.0 KiB)  TX bytes:23948 (23.3 KiB)

eth0:1    Link encap:Ethernet  HWaddr 00:0C:29:88:73:AF
          inet addr:192.168.1.200  Bcast:192.168.1.200  Mask:255.255.255.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:2610 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2610 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5832092 (5.5 MiB)  TX bytes:5832092 (5.5 MiB)
```

Then, we'll edit the *httpd.conf* file:


root@www:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# Almost any Apache directive may go into a VirtualHost container.
# The first VirtualHost section is used for requests without a known
# server name.
#
#<VirtualHost *:80>
#    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>


<VirtualHost 192.168.1.128>
    DocumentRoot /var/www/html/ninja_father
    ServerName www.ninja-father.com
</VirtualHost>

<VirtualHost 192.168.1.200>
    DocumentRoot /var/www/html/ninja_son
    ServerName www.ninja-son.com
</VirtualHost>
```

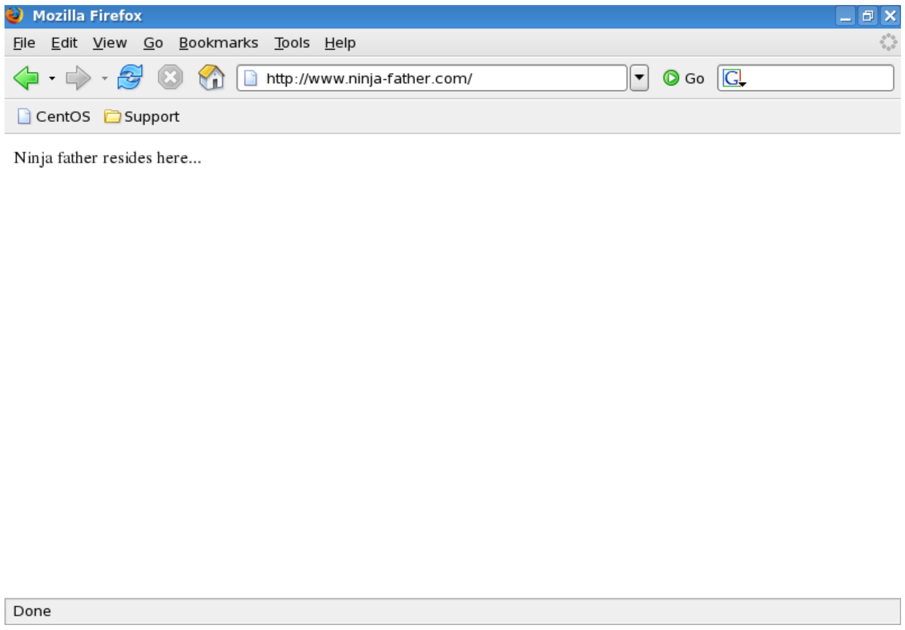
Lastly, we'll edit the */etc/hosts* file:


root@server:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6

#192.168.1.128          www.ninja.com

# This is a configuration for eth0 and eth0:1
192.168.1.128           www.ninja-father.com
192.168.1.200           www.ninja-son.com

# This is the configuration for 1 IP, 2 names
#192.168.1.128          www.ninja-father.com www.ninja-son.com

~
~
~
~
~
~
~
~
~
```

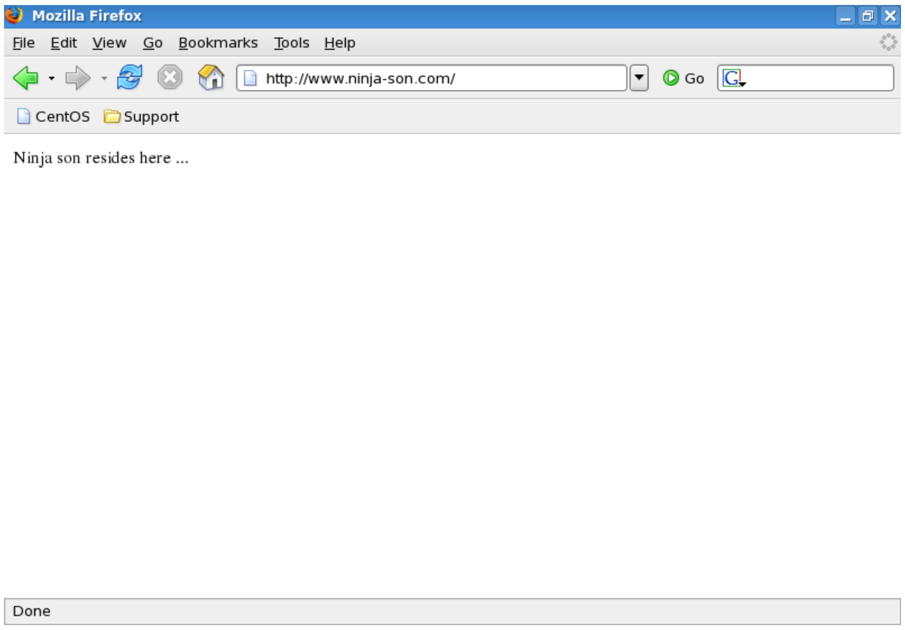
After restarting the server, we'll be able to get to our two sites easily. Again, the change is completely transparent to the user.

www.ninja-father.com


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja-father.com/
Go
CentOS
Support
Done

Ninja father resides here...

www.ninja-son.com


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ninja-son.com/
Go
CentOS Support
Done

Ninja son resides here ...

Excellent.

Please note that the configuration of the virtual network adapter is temporary. You will have to create a network script to preserve the change between reboots. This setup has covered extensively in Part ?: Networking - sub-part 2: Basic and intermediate configurations ().

3.6.3 Other scenarios

Basically, the above two scenarios cover pretty much everything. Once you get the hang of *VirtualHost* setting, creating any which setup becomes a simple matter. Nevertheless, for the sake of clarity, I will demonstrate several more typical scenarios in the examples below, including some features not mentioned yet in this Part of the Book.

3.6.3.1 Different content for intranet and Internet

In practice, this scenario is very similar to having 2 different IPs serving two different websites, except that you will use one website but serve different parts of it to different customers.

Let's assume you wish to achieve the following:

- Allow users on the local network access to all content, but deny some to users on the Internet.
- Allow users on the local network to list directory index, but deny this feature to the Internet users.
- Display a different home page to local users and external customers.
- Allow certain custom scripts to be available only to external customers.

Here's what a sample configuration in *httpd.conf* file would look like:

```
AddHandler cgi-script .cgi

NameVirtualHost 172.16.1.1:80
<VirtualHost 172.16.1.1:80>
    DocumentRoot /www/intranet
    ServerName www.our-company.com
    <Directory /www/intranet>
        Option Indexes FollowSymLinks
    </Directory>
</VirtualHost>

NameVirtualHost 211.211.211.211:80
<VirtualHost 211.211.211.211:80>
    DocumentRoot /www/web
    ServerName www.our-company.com
    <Directory /www/web>
        Options +ExecCGI FollowSymLinks
    </Directory>
</VirtualHost>
```

This examples introduces a number of concepts we have not yet seen, so let's briefly review them:

```
NameVirtualHost
```

This directory allows you to map named-based incoming connections to specific IP addresses. You might ask yourselves why you need this, when we have seen perfectly good examples before, without this feature. Well, the answer is: if somehow a named-based request gets "lost" (due to DNS configuration, firewall rules or similar), it might not match any of the *VirtualHost* blocks. In that case, the default settings configured in *httpd.conf* will be applied to this request, which could be contrary to your needs. Using *NameVirtualHost* forces all incoming connections to a certain IP address to point to a certain *VirtualHost* block. This request will also never fall back to the main configuration, allowing you a complete modularity in your setup.

Thus, in our example, all requests to the internal IP address will go the *VirtualHost* with this IP address declared. We can also see that the users will be able to view directory listings and follow symbolic links.

```
+ExecCGI
```

We see this directive listed under *Options* in the second block, which refers to the Internet customers. All incoming connections on the external IP will go to the *VirtualHost* with this IP declared. We can see the users won't be able to demand directory listings, but they will be able to follow symbolic links - and execute *.cgi* scripts located in this directory.

The ***ExecCGI*** directive tells the server to allow server-side scripting in the specified directory. The plus (+) signs signifies this ***Option*** is used in addition to all those ***Options*** already specified for the ***root*** directory. Similarly, the minus (-) sign can remove some of the privileges, compared to the Options already specified for the ***root*** directory.

However, alone, this directive is insufficient to allow scripting in this directory.

```
AddHandler cgi-script.cgi
```

In order to enable ***.cgi*** scripts to work outside the default script directory, a directive must be added to the ***httpd.conf*** configuration file. Indeed, this is the first line of our sample code - ***AddHandler cgi-script .cgi*** - it allows scripts in non-default directories to be executed, by using the ***+ExecCGI*** option, as we've done before.

The Apache Web server has many other options and features. You are welcome to try them all, using this Part of the Book as the foundation for expanding your knowledge. For more information, please refer to:

- Apache HTTP Server Version 2.0 Documentation
- RedHat Enterprise Linux 4: Reference Guide, Chapter 10: Apache HTTP Server

3.6.3.2 Different websites on different ports

We've already discussed this before. Let's say you have a single IP address with multiple websites served. Using the ***hosts*** file or

DNS resolution is a possibility, but this might not always work. Configuring the Web server to listen on several ports for incoming connections and then using *NameVirtualHost* feature to force the connections to specific *VirtualHost* blocks will force the server to behave as you desire.

Here's an example:

```
Listen 192.168.1.128:80
Listen 192.168.1.128:9021

NameVirtualHost 192.168.1.128:80
<VirtualHost 192.168.1.128:80>
    DocumentRoot /www/white-socks
    ServerName www.white-socks.com
</VirtualHost>

NameVirtualHost 192.168.1.128:9021
<VirtualHost 192.168.1.128:9021>
    DocumentRoot /www/black-socks
    ServerName www.black-socks.com
</VirtualHost>
```

For more examples, please refer to: VirtualHost Examples - Apache HTTP Server.

3.7 Modules

Modules are extensions that enhance the basic functionality of the Web server. The modules reflect the growth of the Web and the inclusion of dynamic content into the web pages. The static HTML can provide only so much functionality. In fact, many of the options we have seen and used above are provided by different modules. For example, the *Order* directive is provided by the *mod_access* module.

3.7.1 Module types

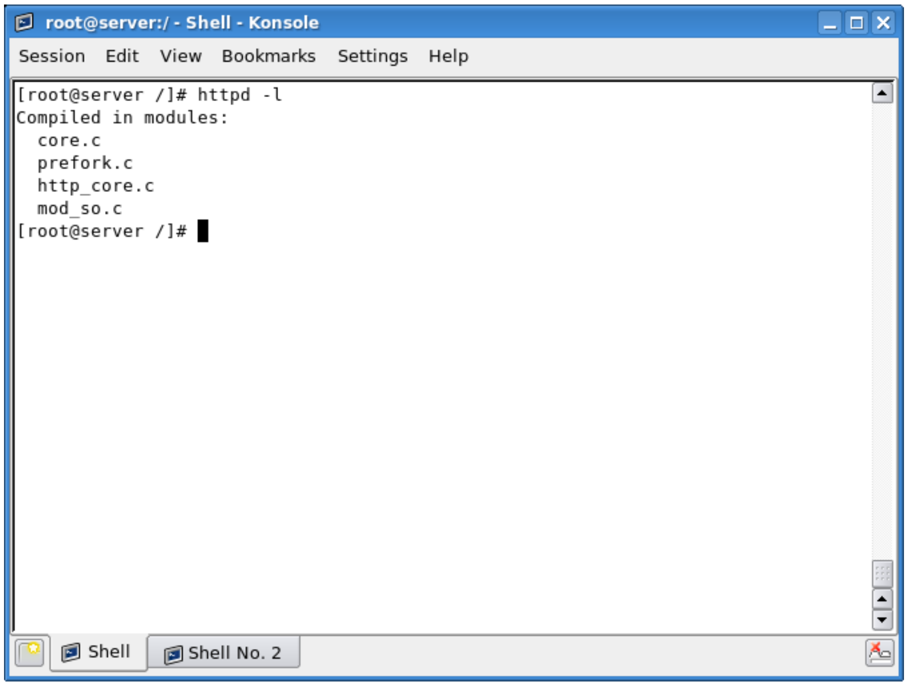
There are two types of modules:

- Built-in modules, which are compiled into Apache and will load with the server any time it is started. Their functionality cannot be removed without recompiling the package. These modules are also known as *static*.
- Loadable modules, which can be loaded on and off as required. These are the *shared* modules.

3.8 View installed modules

You can always list the modules currently used by the server. The command below will display only the modules compiled into Apache.

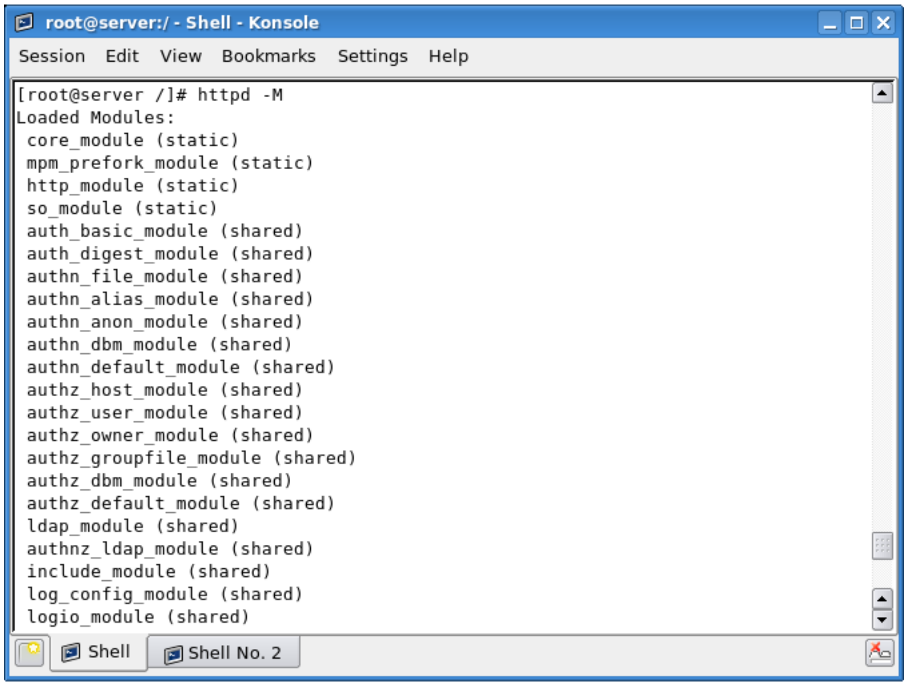
```
httpd -l
```


root@server:/ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
[root@server /]# httpd -l
Compiled in modules:
  core.c
  prefork.c
  http_core.c
  mod_so.c
[root@server /]#
```

This command will list all modules, both *static* and *shared*:

```
httpd -M
```

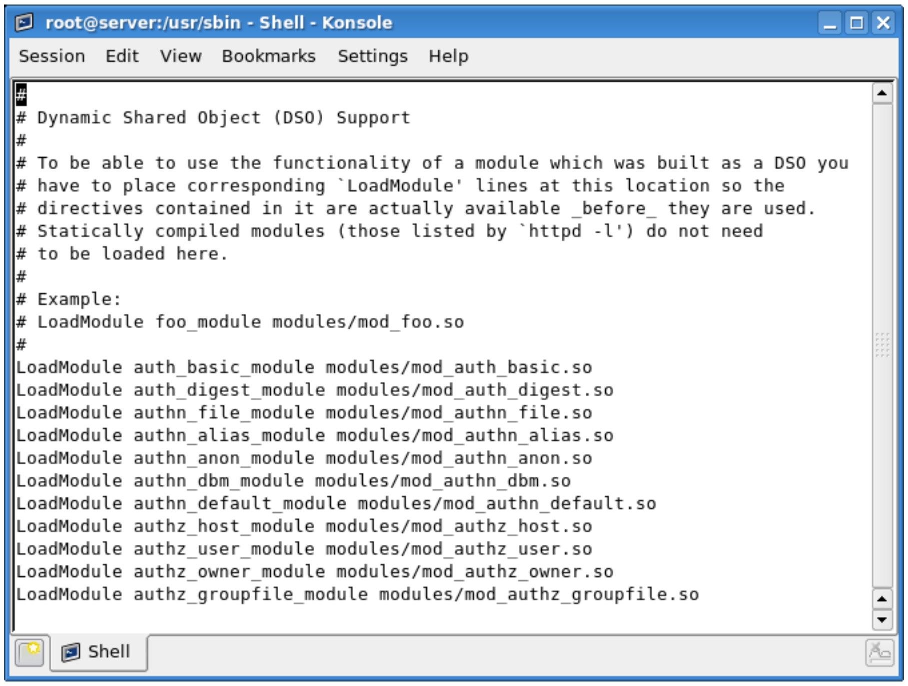

root@server:/ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
[root@server /]# httpd -M
Loaded Modules:
 core_module (static)
 mpm_prefork_module (static)
 http_module (static)
 so_module (static)
 auth_basic_module (shared)
 auth_digest_module (shared)
 authn_file_module (shared)
 authn_alias_module (shared)
 authn_anon_module (shared)
 authn_dbm_module (shared)
 authn_default_module (shared)
 authz_host_module (shared)
 authz_user_module (shared)
 authz_owner_module (shared)
 authz_groupfile_module (shared)
 authz_dbm_module (shared)
 authz_default_module (shared)
 ldap_module (shared)
 authnz_ldap_module (shared)
 include_module (shared)
 log_config_module (shared)
 logio_module (shared)
```

There is a wide range of modules available. We will review a number of more common ones. Please note that the list below is only partial and just briefly introduces the range of available modules.

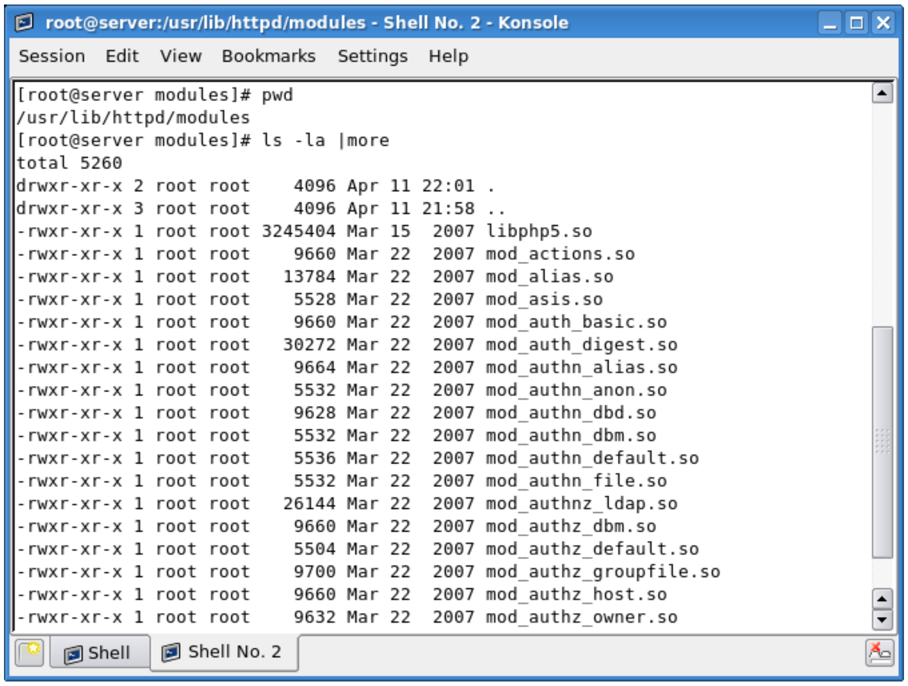
3.8.1 LoadModule

Shared modules are called by the Web server using the *LoadModule* directive in the *httpd.conf* file. If you do not wish to use a certain module, simply comment its line. However, you must remember this will remove the functionality that the module provides.


root@server:/usr/sbin - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding `LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Statically compiled modules (those listed by `httpd -l') do not need
# to be loaded here.
#
# Example:
# LoadModule foo_module modules/mod_foo.so
#
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule authn_file_module modules/mod_authn_file.so
LoadModule authn_alias_module modules/mod_authn_alias.so
LoadModule authn_anon_module modules/mod_authn_anon.so
LoadModule authn_dbm_module modules/mod_authn_dbm.so
LoadModule authn_default_module modules/mod_authn_default.so
LoadModule authz_host_module modules/mod_authz_host.so
LoadModule authz_user_module modules/mod_authz_user.so
LoadModule authz_owner_module modules/mod_authz_owner.so
LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
```

These modules are referenced by a symbolic link in the ***/etc/httpd/***** directory, pointing to *****/usr/lib/httpd/modules*****.**


root@server:/usr/lib/httpd/modules - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
[root@server modules]# pwd
/usr/lib/httpd/modules
[root@server modules]# ls -la |more
total 5260
drwxr-xr-x 2 root root    4096 Apr 11 22:01 .
drwxr-xr-x 3 root root    4096 Apr 11 21:58 ..
-rwxr-xr-x 1 root root 3245404 Mar 15  2007 libphp5.so
-rwxr-xr-x 1 root root    9660 Mar 22  2007 mod_actions.so
-rwxr-xr-x 1 root root   13784 Mar 22  2007 mod_alias.so
-rwxr-xr-x 1 root root    5528 Mar 22  2007 mod_asis.so
-rwxr-xr-x 1 root root    9660 Mar 22  2007 mod_auth_basic.so
-rwxr-xr-x 1 root root   30272 Mar 22  2007 mod_auth_digest.so
-rwxr-xr-x 1 root root    9664 Mar 22  2007 mod_authn_alias.so
-rwxr-xr-x 1 root root    5532 Mar 22  2007 mod_authn_anon.so
-rwxr-xr-x 1 root root    9628 Mar 22  2007 mod_authn_dbd.so
-rwxr-xr-x 1 root root    5532 Mar 22  2007 mod_authn_dbm.so
-rwxr-xr-x 1 root root    5536 Mar 22  2007 mod_authn_default.so
-rwxr-xr-x 1 root root    5532 Mar 22  2007 mod_authn_file.so
-rwxr-xr-x 1 root root   26144 Mar 22  2007 mod_authnz_ldap.so
-rwxr-xr-x 1 root root    9660 Mar 22  2007 mod_authz_dbm.so
-rwxr-xr-x 1 root root    5504 Mar 22  2007 mod_authz_default.so
-rwxr-xr-x 1 root root    9700 Mar 22  2007 mod_authz_groupfile.so
-rwxr-xr-x 1 root root    9660 Mar 22  2007 mod_authz_host.so
-rwxr-xr-x 1 root root    9632 Mar 22  2007 mod_authz_owner.so
```

Let us go over some of the more interesting modules, just a sampling.

3.8.2 mod_access

This module provides access control based on client host name, IP address, or other characteristics of the client request.

3.8.3 mod_dir

This modules provides interface for redirects and serving directory indexes. We have reviewed quite a bit of its functionality in the previous sections.

3.8.4 mod_perl

This module allows dynamic content produced by Perl scripts to be served to incoming requests without using the Perl interpreter every time, reducing overhead and system load. This is done by embedding a Perl interpreter into the Apache server. The module can also emulate a CGI environment, allowing the reuse of Common Gateway Interface (CGI) scripts without any changes to the setup.

3.8.5 mod_python

mod_python allows integration of the Python programming language into the Apache server. It is intended to replace CGI as a method of executing Python scripts on a web server. It offers much faster execution and allows data to be maintained over multiple sessions.

3.8.6 mod_ssl

This module provides an interface to the OpenSSL library, allowing the use of Secure Socket Layer (SSL) and Transport Layer Security (TSL) secure communication protocols. This allows you to run a Web server that will run encrypted sessions with clients, allowing a safe exchange of potentially sensitive data. We will discuss this module again when we setup a secure Web server (7.12).

For a detailed list of available modules and their functionality, please refer to: Apache HTTP Server Module Index.

Chapter 4

.htaccess

***.htaccess* stands for hypertext access. This is the default name of the Apache directory-level configuration file. This file can be used to create security restrictions for particular directories. One of the most common uses is to require user authentication in order to serve certain web pages.**

Before we setup *.htaccess*, there are some things you should remember:

- *.htaccess* is not a replacement for a carefully laid out security plan. You should use the *httpd.conf* file to place restrictions on your server. Only then should you use *.htaccess*, to further restrict the already *allowed* users.

- Do not ever use *.htaccess* to handle secure or privileged content, like user data.

- *.htaccess* file is loaded every time a webpage is requested, incurring a performance loss.

- Using this file grants individual users an ability to make security modifications to your site, creating possible risks if not properly configured.

On the other hand, using *.htaccess* is useful if you run a multi-user hosting plan. These users do not have *root* access to the main configuration file and their only way of "shaping" traffic is by using the *.htaccess* file. In general, the use of the *.htaccess* file should be limited to non-root users only.

Before we can setup access-protected pages, we need to briefly overview the layout and syntax of the *.htaccess* files. Let's examine what a typical *.htaccess* file looks like. Then, we will combine it with our web content.

```
AuthType Basic
AuthName "Restricted web page"
AuthUserFile "/etc/httpd/conf/.htpasswd
require valid-user
```

```
AuthType Basic
```

This line defines the type of authentication. *Basic* means there is no encryption and the password hash is sent as clear text. This is one of the major reasons why *.htaccess* cannot be considered for protection of confidential user data.

```
AuthName "Restricted web page"
```

When someone tries to access an *.htaccess*-protected page, a user-name & password window will pop in the web browser. This window will bear a title - this is the *AuthName*. It can be anything you like.

```
AuthUserFile /etc/httpd/conf/.htpasswd
```

This line defines the path to a file where user credentials are stored. This file does not exist, but we will create it soon.

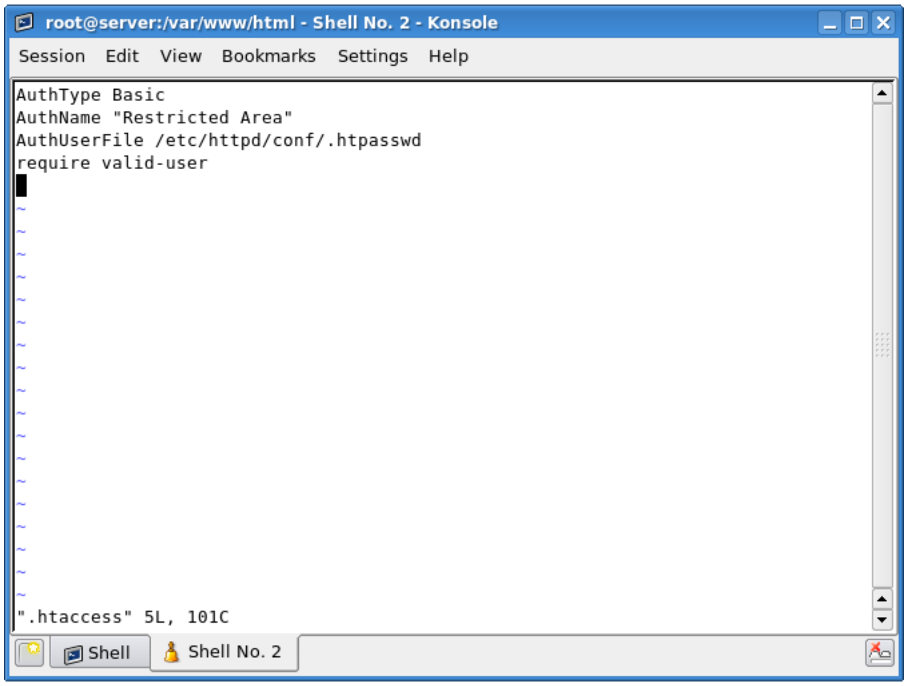
```
require valid-user
```

This line indicates only successful authentication attempts will result in the loading of the page.

Now that we know what we're about, we will:

- Create an *.htaccess* file similar to the one above.
- Create the *.htpasswd* file containing usernames & password necessary for the authentication.
- Place *.htaccess* in the directory we wish users to validate before accessing the content.
- Tell *httpd* to allow user authentication via *.htaccess* files.
- Restart the server.
- Test the results.

4.1 Create .htaccess file


root@server:/var/www/html - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
AuthType Basic
AuthName "Restricted Area"
AuthUserFile /etc/httpd/conf/.htpasswd
require valid-user

".htaccess" 5L, 101C
```

4.2 Create .htpasswd file

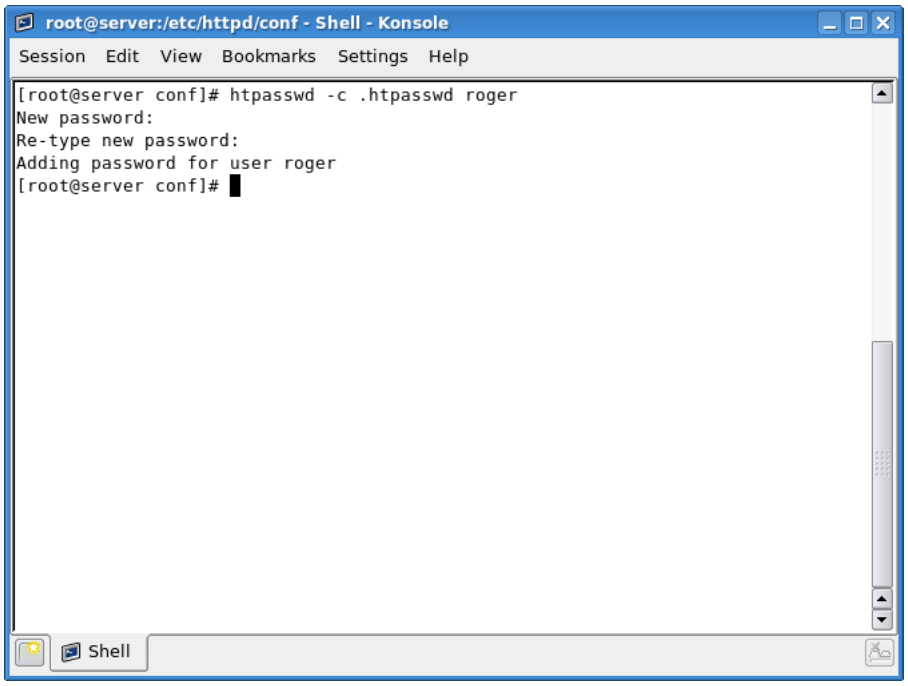
First, we will access the directory where we intend to place the file - */etc/httpd/conf*. It can be any directory, but it must be outside the *DocumentRoot*, so it so not viewable by your clients. Furthermore, only the root should be able to modify this file.

Make sure only root can modify the *.htpasswd* file! It should have permissions set to 0644.

Users and passwords are added to the file by running the ***htpasswd*** command.

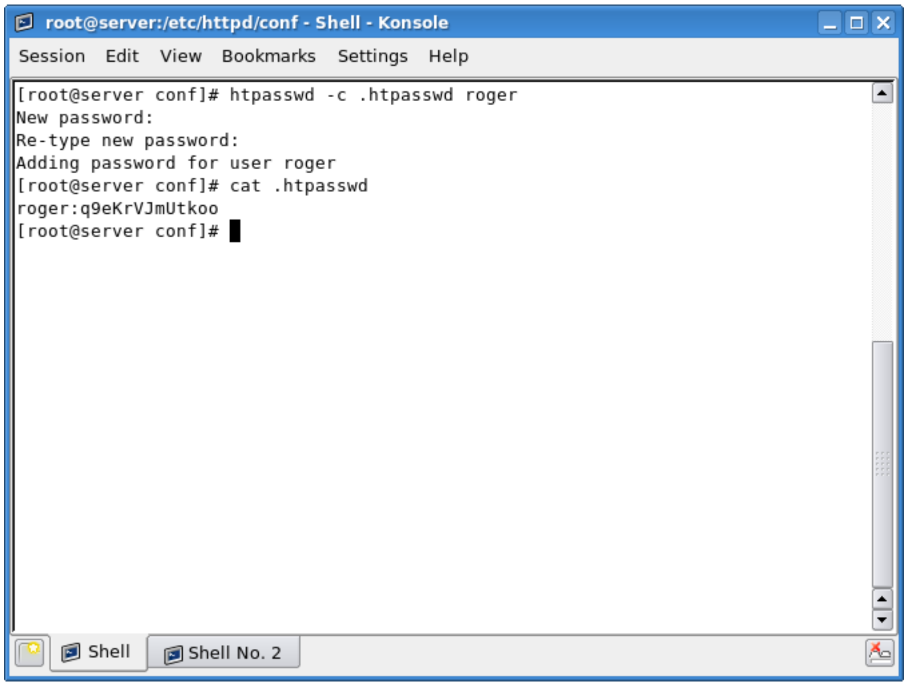
```
htpasswd -c .htpasswd username
```

The name of the authentication file can be anything. You may consider changing it to something else.


root@server:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@server conf]# htpasswd -c .htpasswd roger
New password:
Re-type new password:
Adding password for user roger
[root@server conf]#
```

After you have finished adding the usernames (there can be one or more), you can see the contents of the ***.htpasswd*** file. The passwords are encrypted.


root@server:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
[root@server conf]# htpasswd -c .htpasswd roger
New password:
Re-type new password:
Adding password for user roger
[root@server conf]# cat .htpasswd
roger:q9eKrVJmUtkoo
[root@server conf]#
```

4.3 Copy .htaccess to restricted directory

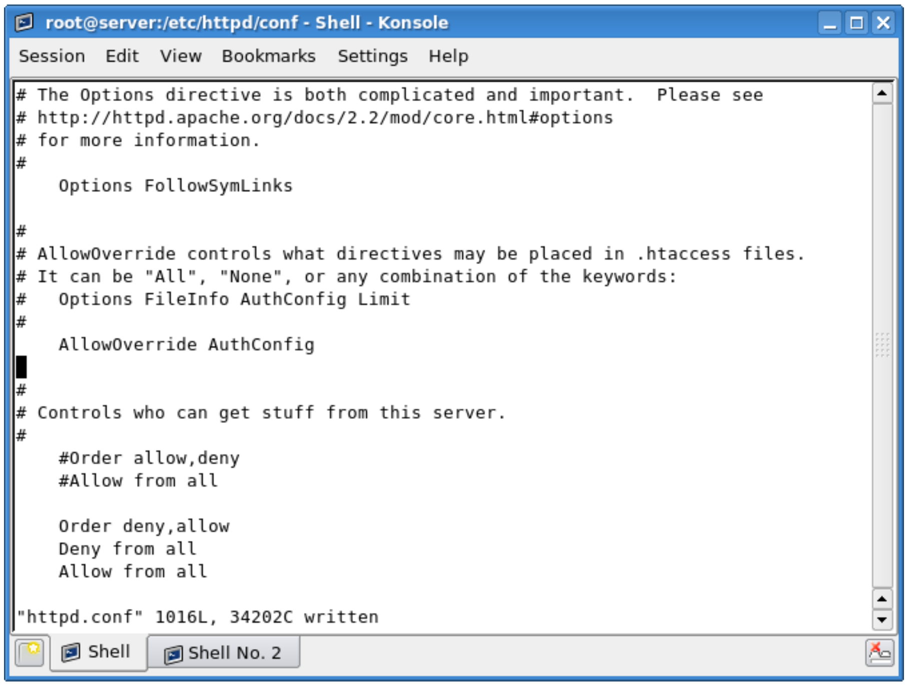
We will place the *.htaccess* file in our *DocumentRoot*. To make things interesting, we will also change the site and the homepage somewhat. Instead of *ninja.com*, we will serve *ourserver.com*. This is the site we will use to configure a DNS server in the next Part ().

4.4 Configure httpd.conf to allow authentication via .htaccess

By default, *.htaccess* files are given no control whatsoever. This is accomplished by the *AllowOverride* directive. This directive

specifies what the *.htaccess* files can do - in addition and contrary to main configuration settings. Please note that this could pose a security risk. Badly configured *.htaccess* files can compromise the security of your system.

We will allow *.htaccess* to authenticate users. We will replace the original *AllowOverride none* to *AllowOverride AuthConfig*.


root@server:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

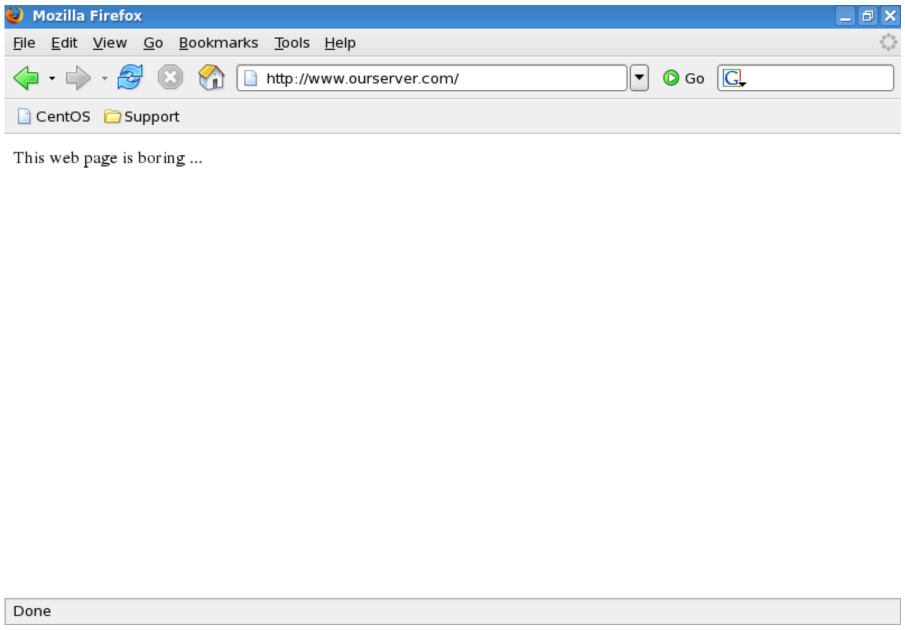
```
# The Options directive is both complicated and important.  Please see
# http://httpd.apache.org/docs/2.2/mod/core.html#options
# for more information.
#
    Options FollowSymLinks

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
    AllowOverride AuthConfig

#
# Controls who can get stuff from this server.
#
    #Order allow,deny
    #Allow from all

    Order deny,allow
    Deny from all
    Allow from all

"httpd.conf" 1016L, 34202C written
```

4.5 Restart server

```
service httpd restart
```

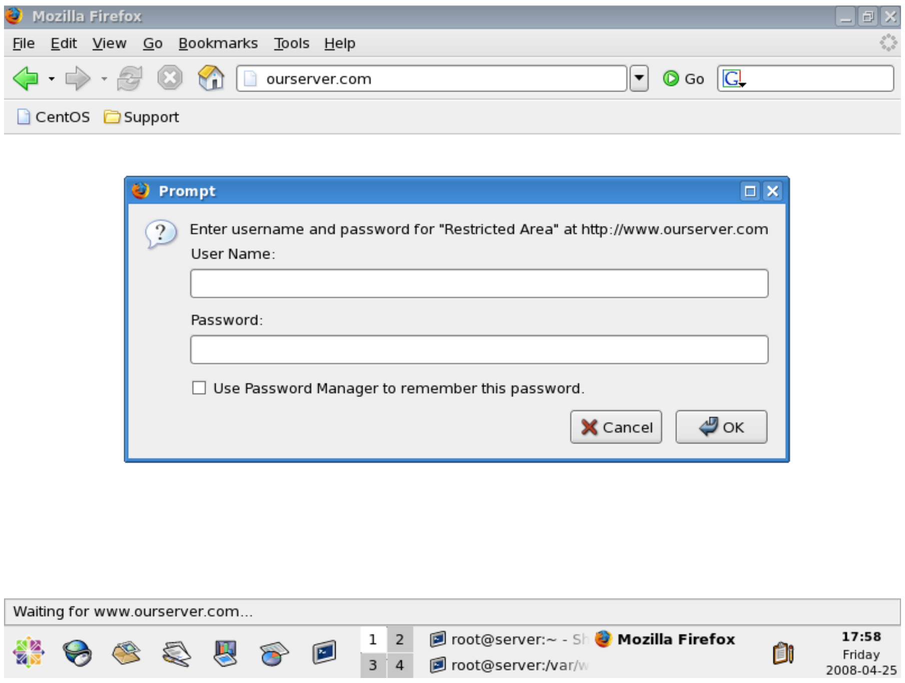
4.6 Test setup

This is the webpage, seen without any restrictions.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/
Go
CentOS
Support
Done

This web page is boring ...

Now, after restarting the server, we will be asked for authentication credentials.



Enter username and password for "Restricted Area" at http://www.ourserver.com

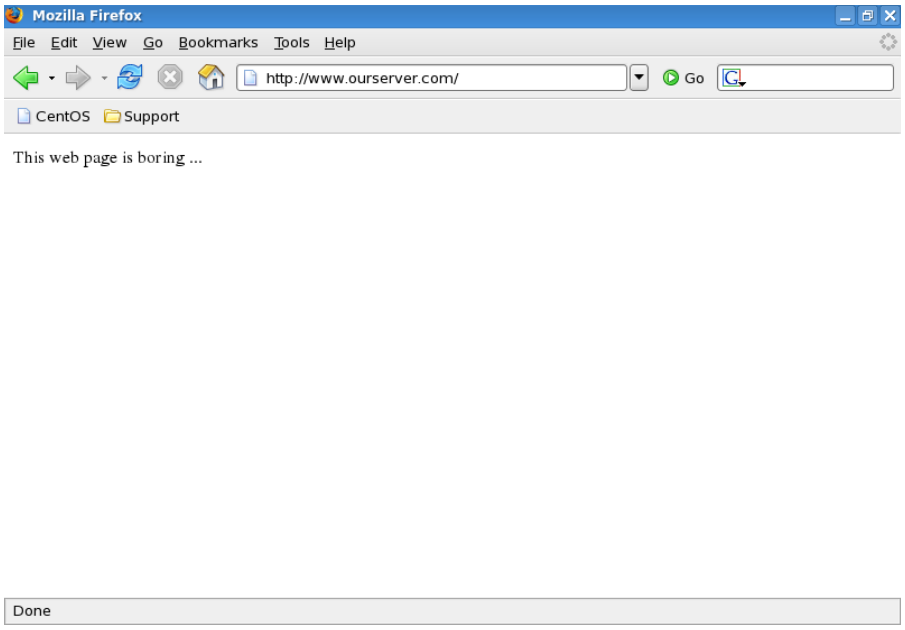
User Name:

Password:

☐ Use Password Manager to remember this password.

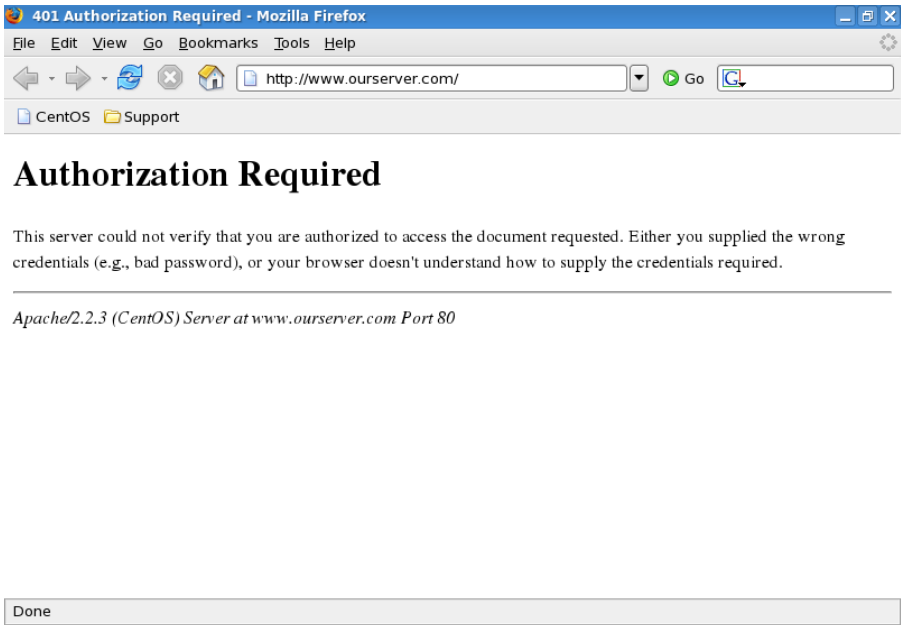
Waiting for www.ourserver.com...

If we succeed, we will reach the webpage, like before.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/
Go
CentOS
Support
Done

This web page is boring ...

If we enter the wrong username & password - or none, we will be rejected. You can customize the "reject" page, if you like.


401 Authorization Required - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/
Go
CentOS
Support
Done

Authorization Required

This server could not verify that you are authorized to access the document requested. Either you supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.

Apache/2.2.3 (CentOS) Server at www.ourserver.com Port 80

4.7 Other configurations

While this pretty much covers the basic setup of *.htaccess* files, there are several more things you should remember.

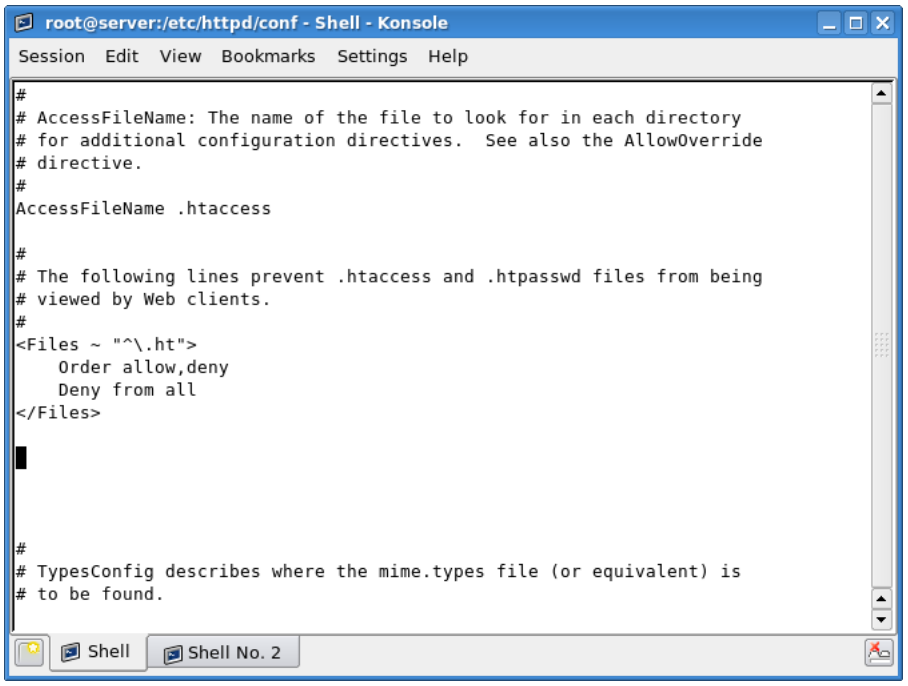
4.7.1 Inheritance & performance loss

Please remember that the *.htaccess* restrictions are inherited by all sub-directories that exist in the directory you have placed the file. This means that whenever one of your clients tries to access a page in one of the sub-directories, the server will have to make a recursive search up the directory tree until it finds the file. Furthermore, even if it does find the file, the server will have to check up every directory up the tree to create a complete set of restrictions.

4.7.2 Disable web access to .htaccess

By default, Apache prevents any file beginning with letters *.ht* to be visible through the web browser. This is a minor security consideration, which allows you to keep your *.htaccess* files safe from prying eyes, even though they are located in world-readable location (*DocumentRoot* directories and sub-directories).

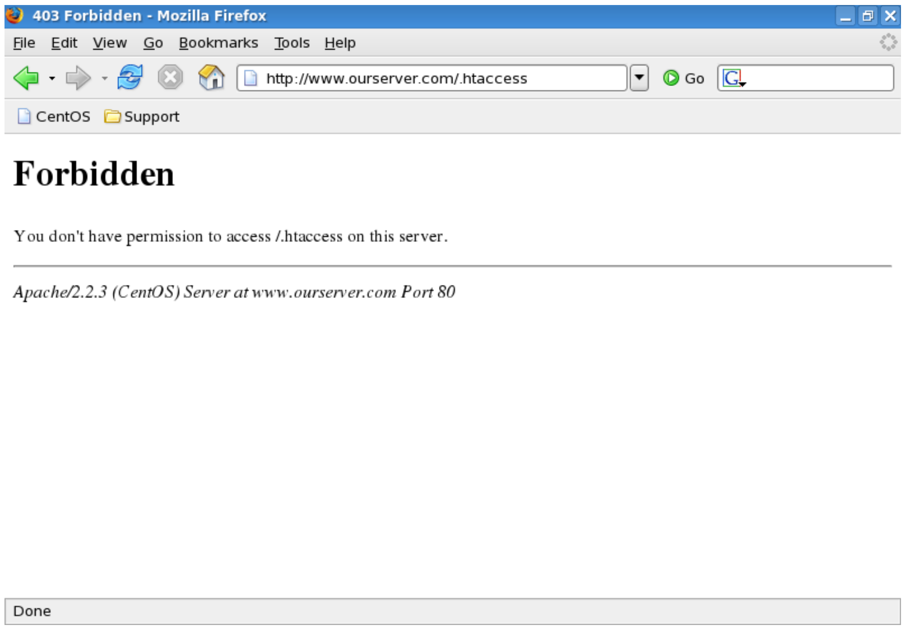
This behavior is governed by the combination of the *AccessFileName* and *Files* directives. We have seen this example earlier when we review the *Files* tag; now, we can see them in practical use.


root@server:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
#
# AccessFileName: The name of the file to look for in each directory
# for additional configuration directives.  See also the AllowOverride
# directive.
#
AccessFileName .htaccess

#
# The following lines prevent .htaccess and .htpasswd files from being
# viewed by Web clients.
#
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>




#
# TypesConfig describes where the mime.types file (or equivalent) is
# to be found.
```

You can also setup other types of files - or just specific files - from being accessible - or accessible only to certain hosts.

Indeed, if we try to reach *.htaccess* through the web browser, we will be denied access.


403 Forbidden - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/.htaccess
Go
CentOS
Support
Done

Forbidden

You don't have permission to access /.htaccess on this server.

Apache/2.2.3 (CentOS) Server at www.ourserver.com Port 80

Chapter 5

Secure Web server

Running a secure Web server is something you should consider if the daily use of your websites will include an exchange of confidential, private information from your users. Regular Web servers send and receive traffic in unencrypted form. Unfortunately, this makes them vulnerable to man-in-the-middle attacks, where a potential attacker could use sniffer tools to log packets en route from clients to the server and derive sensitive information from them. This mode of security is completely unacceptable for websites that must deal in personal data, like bank accounts, medical or financial records, or others.

The secure Web server eliminates this threat by offering two key advantages:

- It allows users to verify the identity of the server.
- It allows users to conduct safe transactions with your server by encrypting the authentication and the session.

To achieve this, the Apache Web server uses secure communication protocols like the Secure Socket Layer (SSL) or the Transport Layer

Security (TLS) to protect the flow of data.

5.1 Encrypted session

Before we setup a secure server, we should first understand how encrypted communication between the server and the client is conducted. Let us outline the details of a typical secure session:

- A client tries to connect to port 443 on the secure Web server.
- The client sends a list of available encryption methods it supports; if the client cannot support encryption, for instance very old browsers, the connection attempt will be unsuccessful. Modern browsers support both SSL and TLS without any problems.
- The server will choose the strongest available encryption method that both sides can support.
- The server will then send back to the client its certificate and the public encryption key. The certificate is a sort of an ID, telling the client important information about the server. To make this information credible, the certificate must be signed by a reputable Certificate Authority (CA), like EquiFax, Thawte or others. The public key will be used by the client to generate its own encryption hash should it choose to accept the server's certificate.
- The client receives the certificate. In most browsers, the certificate is first compared to an existing list of authorities. If the digital signature matches, the certificate will be accepted. If no match is found for the certificate, the browser might use the Online Certificate Status Protocol (OCSP) to connect to CAs in real time in an attempt to verify the certificate. Generally, the use of OCSP is not enabled by default in most browsers, in order to speed up the authentication process. If no

match is found still, the client will be issued a warning by the browser, informing it that the certificate could not be verified. The user now must decide whether he/she can take the risk and accept the certificate. In addition to being self-signed (i.e. no CA signature), the typical issues arising with certificate prompts include a mismatch between the site you are trying to access and the one registered in the certificate, dubious credentials or an expired certificate.

- Regardless of what may occur, if the client accepts the connection, it will send back a hash encrypted with the server's public key. This hash will be used to encrypt all communication between the server and the client throughout the session. Only the client will be able to decrypt the communications - or rather, anyone who possesses the private key. But if the client side is fairly secure and the server's certificate is valid, the communication is safe.

5.2 Requirements

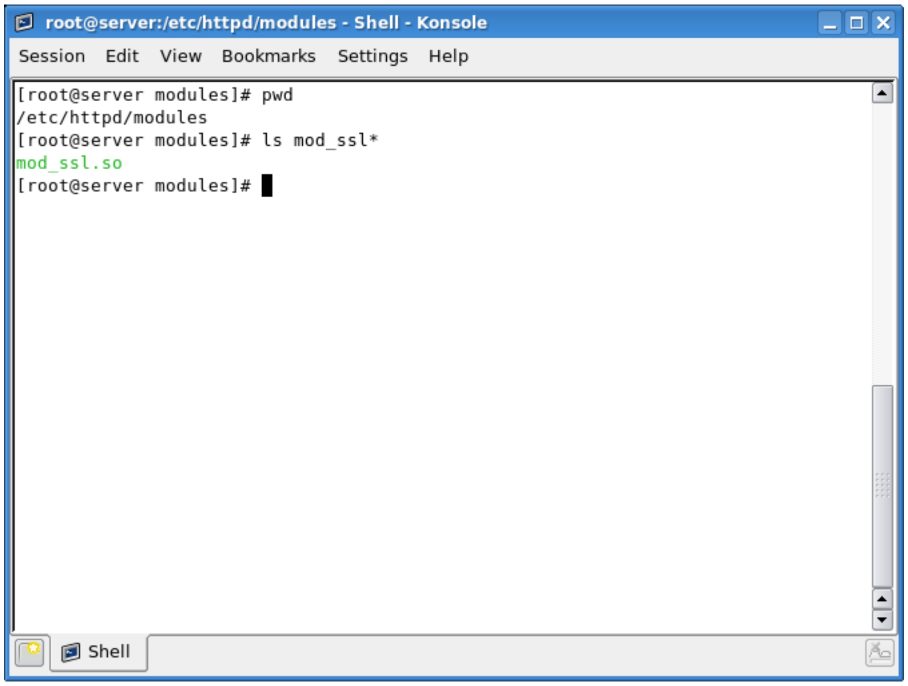
We have already mentioned that the client must support some sort of encryption to able to establish secure connections to a server. On the server end, the server must also support the secure communication protocols. The Apache Web server uses the *mod_ssl* module, which provides an interface to the OpenSSL library, allowing the use of SSL and TLS.

By default, most distributions today ship with the OpenSSL library installed and the Apache server compiled against the *mod_ssl* module. If your distro does not include either one or both, you will have to obtain them before you can use a secure Web server.

You can check if you have the OpenSSL library installed:

```
rpm -q openssl
```

And to certify if Apache uses *mod_ssl*, you should look for it in the */etc/httpd/modules* directory.


root@server:/etc/httpd/modules - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
[root@server modules]# pwd
/etc/httpd/modules
[root@server modules]# ls mod_ssl*
mod_ssl.so
[root@server modules]# 
```

5.3 Limitations

On one hand, the secure Web server offers verification of the server's identity and safe transactions. On the other hand, it is slower than the regular server. Therefore, you should take into consideration the performance loss stemming from the use of encryption. You should not use the secure Web server for regular daily content that does not include any exchange of personal information.

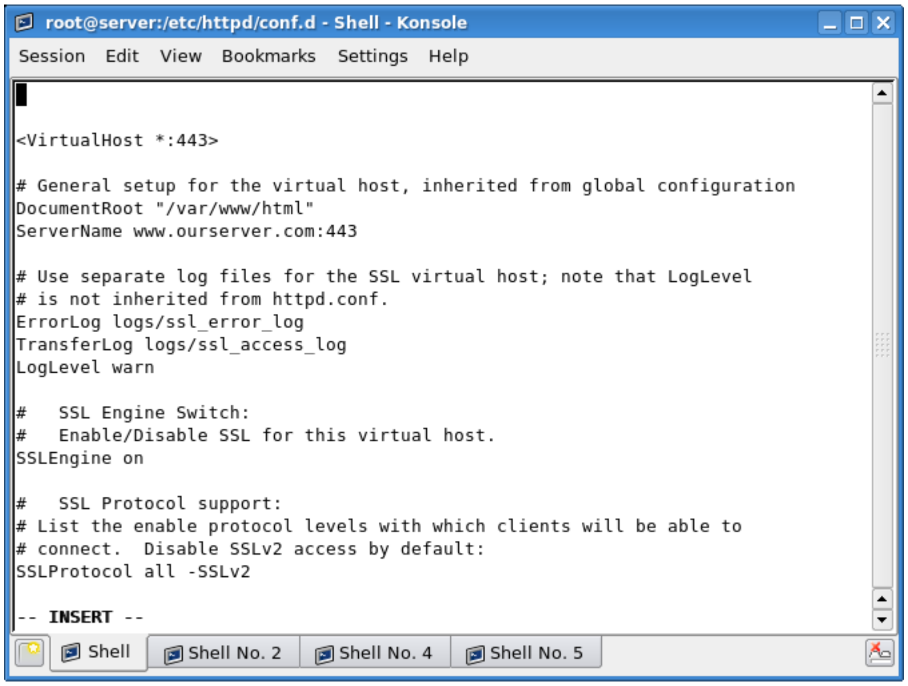
5.4 Setup

5.4.1 Main configuration file(s)

The main configuration file for the secure Apache Web server is:

```
/etc/httpd/conf.d/ssl.conf
```

This file is very similar to *httpd.conf*, except that it includes a number of special directives. But the principle remains the same. Basically, the configuration file contains a *VirtualHost* block, where all secure Web server directives should be listed. We will edit this block to suit our needs.


root@server:/etc/httpd/conf.d - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5

```


<VirtualHost *:443>

# General setup for the virtual host, inherited from global configuration
DocumentRoot "/var/www/html"
ServerName www.ourserver.com:443

# Use separate log files for the SSL virtual host; note that LogLevel
# is not inherited from httpd.conf.
ErrorLog logs/ssl_error_log
TransferLog logs/ssl_access_log
LogLevel warn

#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on

#   SSL Protocol support:
# List the enable protocol levels with which clients will be able to
# connect.  Disable SSLv2 access by default:
SSLProtocol all -SSLv2

-- INSERT --
```

5.4.2 Backup

We will first backup the file before making any changes.

```
cp /etc/httpd/conf.d/ssl.conf →
   /etc/httpd/conf.d/ssl.conf-backup
```

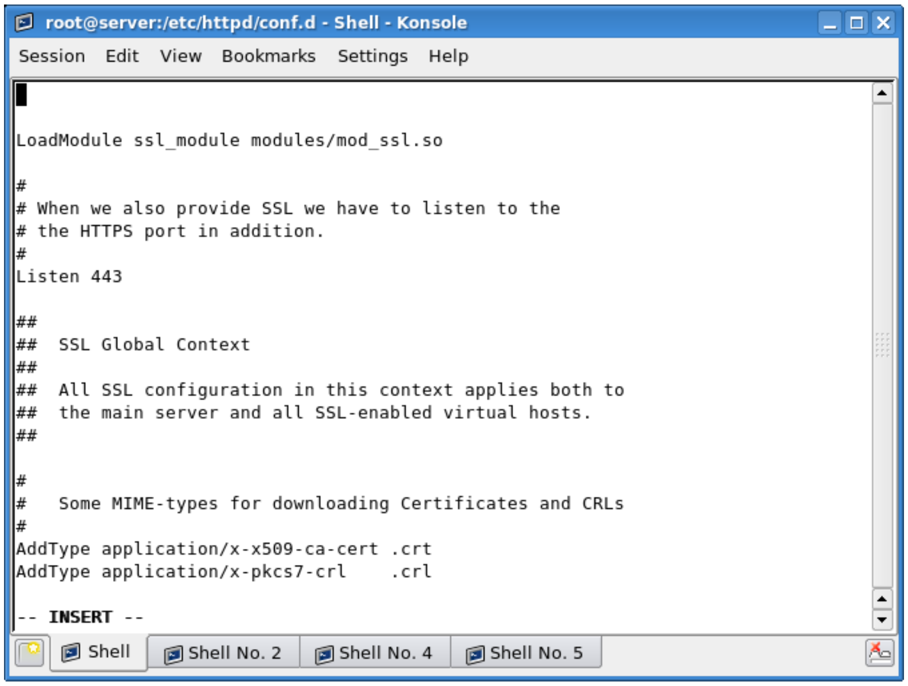
5.4.3 Edit the ssl.conf configuration file - part 1

Again, we need to make a number of changes to get our server to work. However, before we can fully edit all of the necessary options, we will have to digitally sign our server. This includes creating the public key and signing it with a certificate from a known, reputable CA. However, since we do not have a certificate, it costs money and the process takes time, for the purpose of this exercise, we will create our own CA and use it to sign our server.

But first, let us review the most important directives that we need to get our server started. The procedure is identical to what we have done earlier.

5.4.3.1 LoadModule

This directive instructs the server to use the *mod_ssl* module. The path is relative to the *ServerRoot* directive specified in the *httpd.conf* configuration file. Without loading the module, our encryption will not work.


root@server:/etc/httpd/conf.d - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5

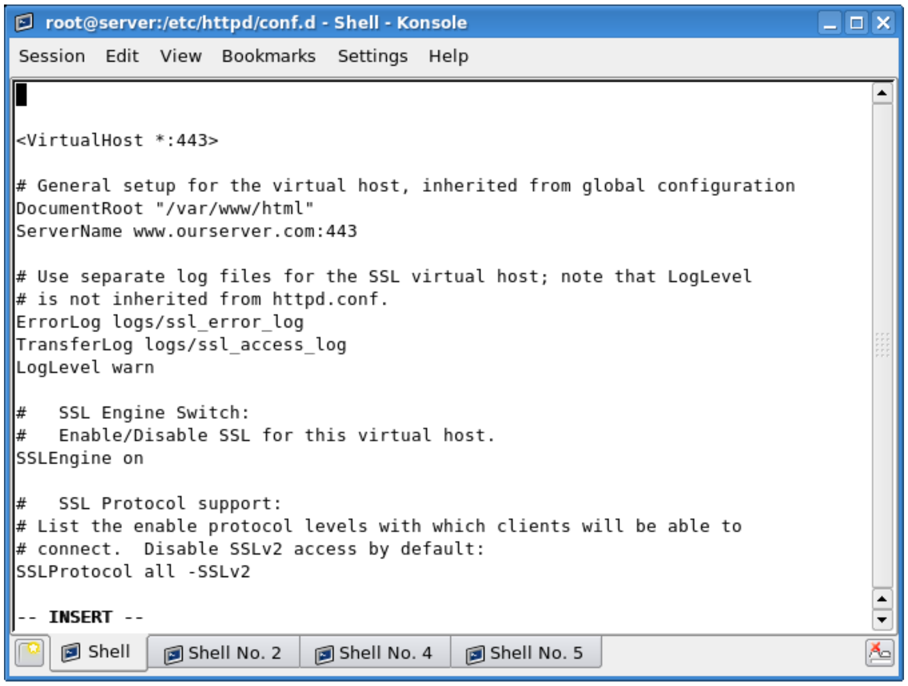
```


LoadModule ssl_module modules/mod_ssl.so

#
# When we also provide SSL we have to listen to the
# the HTTPS port in addition.
#
Listen 443

##
##  SSL Global Context
##
##  All SSL configuration in this context applies both to
##  the main server and all SSL-enabled virtual hosts.
##

#
#   Some MIME-types for downloading Certificates and CRLs
#
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl    .crl

-- INSERT --
```

5.4.3.2 Listen

This directive instructs the server to listen for incoming connections on port 443. This is the accepted convention for secure Web communications (https). It is critical that this port be different from the port used by the regular server. See the image above.

5.4.3.3 VirtualHost

Here, we define our secure Web server. Using the ***VirtualHost*** block is the most elegant way of doing it. This allows you to create additional blocks and serve additional secure sites to your clients, allowing you an extra degree of flexibility and security.


root@server:/etc/httpd/conf.d - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5

```

<VirtualHost *:443>

# General setup for the virtual host, inherited from global configuration
DocumentRoot "/var/www/html"
ServerName www.ourserver.com:443

# Use separate log files for the SSL virtual host; note that LogLevel
# is not inherited from httpd.conf.
ErrorLog logs/ssl_error_log
TransferLog logs/ssl_access_log
LogLevel warn

#   SSL Engine Switch:
#   Enable/Disable SSL for this virtual host.
SSLEngine on

#   SSL Protocol support:
# List the enable protocol levels with which clients will be able to
# connect.  Disable SSLv2 access by default:
SSLProtocol all -SSLv2

-- INSERT --
```

Like we did before, we need to setup the *DocumentRoot*, the *ServerName* and other directives. Let us review the most important elements:

```
<VirtualHost *:443>
```

This tells our server to listen on all interfaces for incoming connections on port 443. You may consider narrowing down the range to specific IP addresses. Nevertheless, it is important to remember that you can only use IP addresses! The secure Web server does not permit named-based connections in its *VirtualHost* block. This is because the SSL handshake occurs before the HTTP request can identify the named-based virtual host.

Use only IP-based *VirtualHost* directives in the *ssl.conf* configuration file! Name-based virtual hosts will fail.

```
DocumentRoot "/var/www/html"
```

This directive specifies the directory where all your web pages should be stored. It is recommended that you use a different root for non-secure and secure pages. However, in our example, we will use the default selection. Just remember that this is NOT the optimal setting.

```
ServerName www.ourserver.com:443
```

This entry defines the server name. If you do not use the *hosts* file or DNS server for name resolution, you will have to specify an IP address. We have solved this limitation earlier, so we can use the server name here. In a production setup, where your server is used by clients on the Internet, you will have to use DNS for name resolution. For study and testing and in small, private networks, the *hosts* file is an adequate solution.

This covers the first part of our setup. Now we must create the certificate.

5.4.4 Create SSL certificate

Like we said before, we will create a CA, create a server key and then sign the key with our self-created CA. In a production setup,

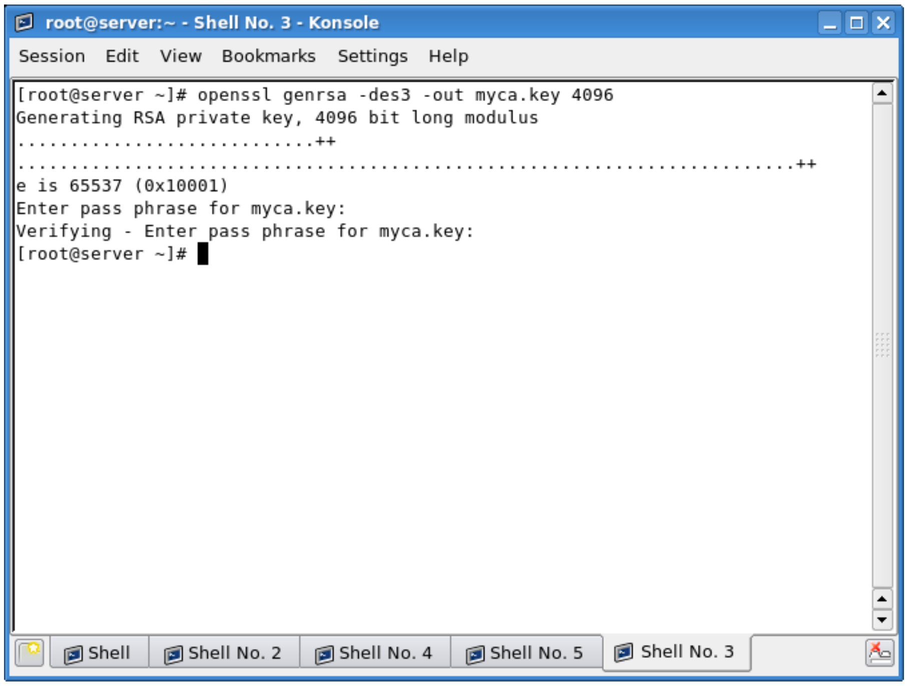
this will not work. If you intend to run any semi-serious business, you will have to use a reputable, world-acknowledged CA to sign your certificates.

Please note that the comparison between our setup and the real scenario can be slightly confusing. If you get lost, there's a table summary (5.4.7) at the end of this section, emphasizing the important differences between the two setups.

5.4.4.1 Create Certificate Authority (CA)

The first step is to create an encryption key, which we will use to sign our CA. Please note that you should use a meaningful name for the key. The best way to avoid confusion is to use the letters *ca* in the name of the CA key. Likewise, use the word *server* when creating the server key.

I have chosen the name *myca.key*, so that we do not confuse this self-generated key (and the CA) with real keys.


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3

```
[root@server ~]# openssl genrsa -des3 -out myca.key 4096
Generating RSA private key, 4096 bit long modulus
............................++
..........................................................................++
e is 65537 (0x10001)
Enter pass phrase for myca.key:
Verifying - Enter pass phrase for myca.key:
[root@server ~]# 
```

Let us review the command:

```
openssl genrsa -des3 -out myca.key 4096
```

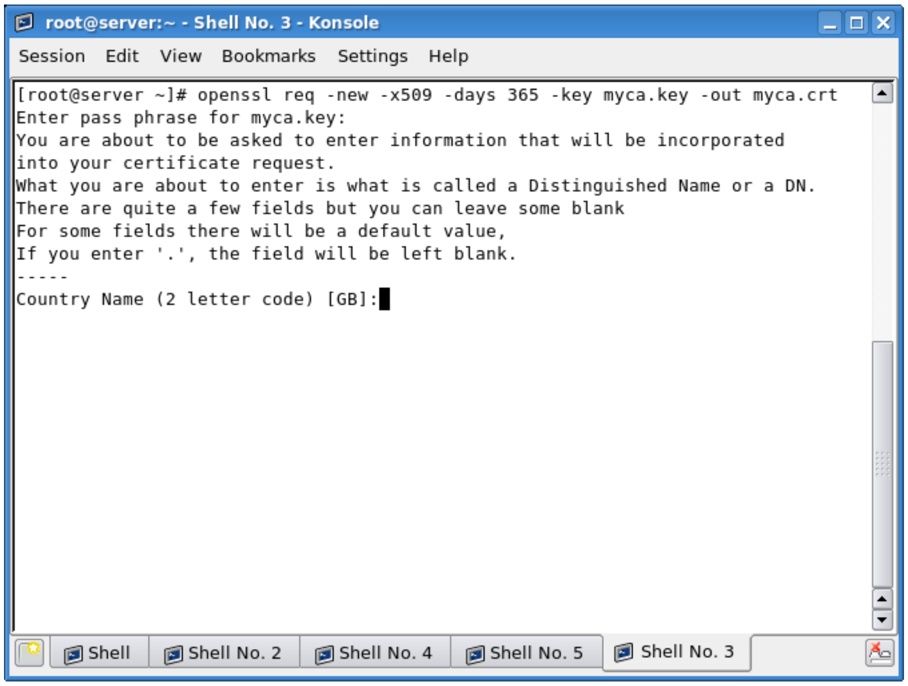
This OpenSSL command line tool will generate an RSA key, using the Triple-DES cypher. The *-out* flag signifies the output name. The number at the end of the command tells us how long the key will be; generally, the longer the better. A 4096-bit encryption is quite sufficient.

Please refer to openssl man page for more details.

After the key is created, you will be asked to use a password. This means you won't be able to use this key without providing the password. While in theory, this is an interesting security measure, it offers little actual benefit. We'll discuss this soon.

Now that we have the key, we will create a CA.

```
openssl req -new -x509 -days 365 →
   -key myca.key -out myca.crt
```


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3

```
[root@server ~]# openssl req -new -x509 -days 365 -key myca.key -out myca.crt
Enter pass phrase for myca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:
```

What do we have here? Well, basically, we are creating a certificate, using the key we have created earlier. Let us go over the details:

```
req -new -x509
```

This part of the command tells us we want to issue a new X.509 Certificate Signing Request (CSR), where X.509 is an international standard for public key and privilege management infrastructures. In simple words, we want to create a certificate that will identify our CA.

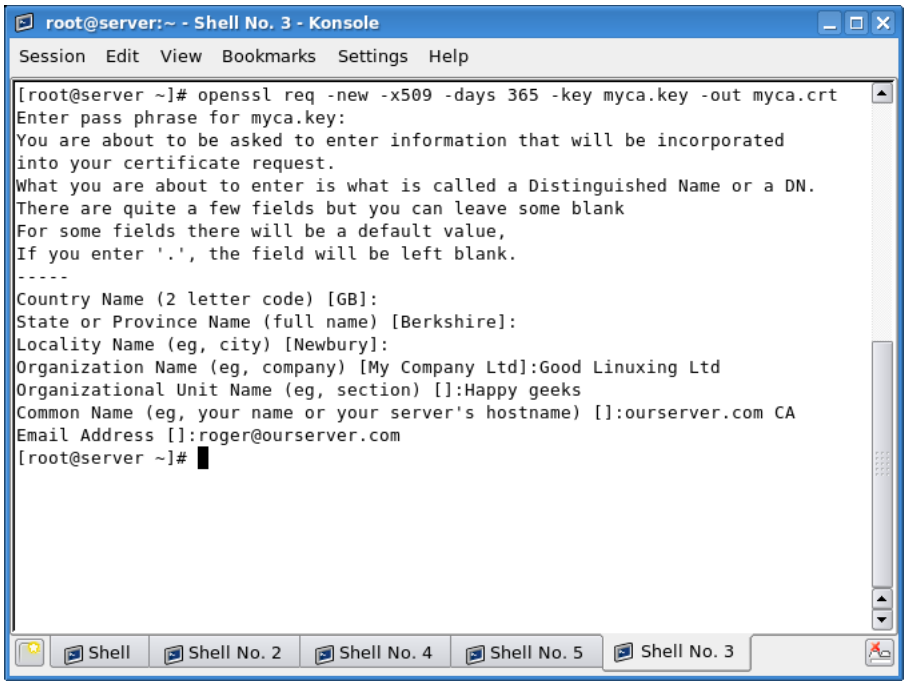
```
-days 365
```

This tells us how long the certificate will be valid. Security aspects of this parameter are examined in greater depth in the Security chapter (7.12).

```
-key myca.key -out myca.crt
```

We will use the key we have created earlier to sign the certificate for the CA.

The command will invoke a guided text-interface wizard. We will have to provide the password for the certificate key before we can continue. After that, we will have to fill out an interactive form, including the basic credentials that will identify us as the CA.


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3

```
[root@server ~]# openssl req -new -x509 -days 365 -key myca.key -out myca.crt
Enter pass phrase for myca.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:
State or Province Name (full name) [Berkshire]:
Locality Name (eg, city) [Newbury]:
Organization Name (eg, company) [My Company Ltd]:Good Linuxing Ltd
Organizational Unit Name (eg, section) []:Happy geeks
Common Name (eg, your name or your server's hostname) []:ourserver.com CA
Email Address []:roger@ourserver.com
[root@server ~]# 
```

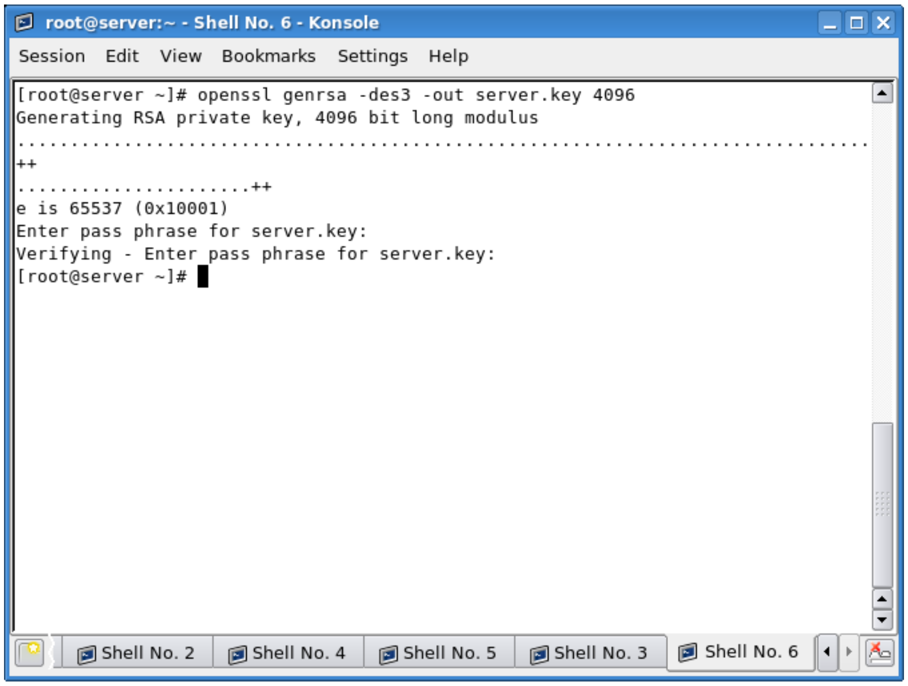
Please note that you should be careful when entering the *Common Name*. You should use meaningful entries that will allow you to easily distinguish your records, especially if you have several CAs. Most people will never have to bother with this setting, but should a need arise, here's a pair of simple rules that you should adhere to when creating CAs:

- For each CA, use the name of the site it will certify; in our case, ourserver.com (or www.ourserver.com).
- Append the letters *CA* to the end of the *Common Name*, so you will know this is the CA entry.

In a real life situation, your credentials would be replaced with those of an existing, reputable CA.

5.4.4.2 Create server key

We now have a certificate. It's time to create the server key. The principle is similar to what we've done before. The one thing you should remember is that the server key should be named ***server.key***, in order to conform with Apache conventions.


root@server:~ - Shell No. 6 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

```
[root@server ~]# openssl genrsa -des3 -out server.key 4096
Generating RSA private key, 4096 bit long modulus
................................................................................
++
......................++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
[root@server ~]#
```

After the encryption key is created, we will be asked to provide a password to make the use of our key impossible without knowing it. While this method is somewhat effective, it is not considered a serious security measure. In fact, you are advised not to use it, since the benefits do not outweigh the shortcomings.

Since you must provide the password any time the server is restarted or reloaded, this means the secure server will not be able to start

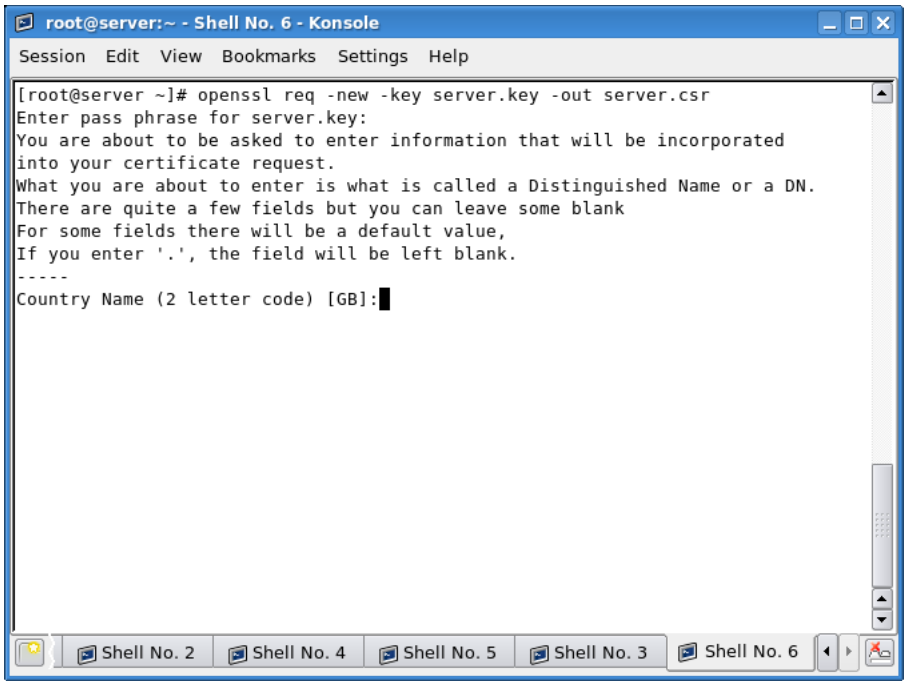
after unattended reboot and will require a presence of an administrator to activate. This is cumbersome and can even be impractical. On the other hand, should your system be compromised, the password will most likely present little challenge to the attacker. Furthermore, compromised systems cannot be trusted, whether passwords or other security methods are used.

Nevertheless, we will demonstrate both methods, so you can learn and use both, should a need arise. We will begin with the password-protected key and then later, create another one, which uses no password.

5.4.4.3 Create Certificate Signing Request (CSR)

Now, we must "ask" our CA to sign our certificate. In a real life situation, you would receive the *server.csr* from an existing, established CA. Or you might even receive the signed key, with the information you have provided in an application form, for instance.

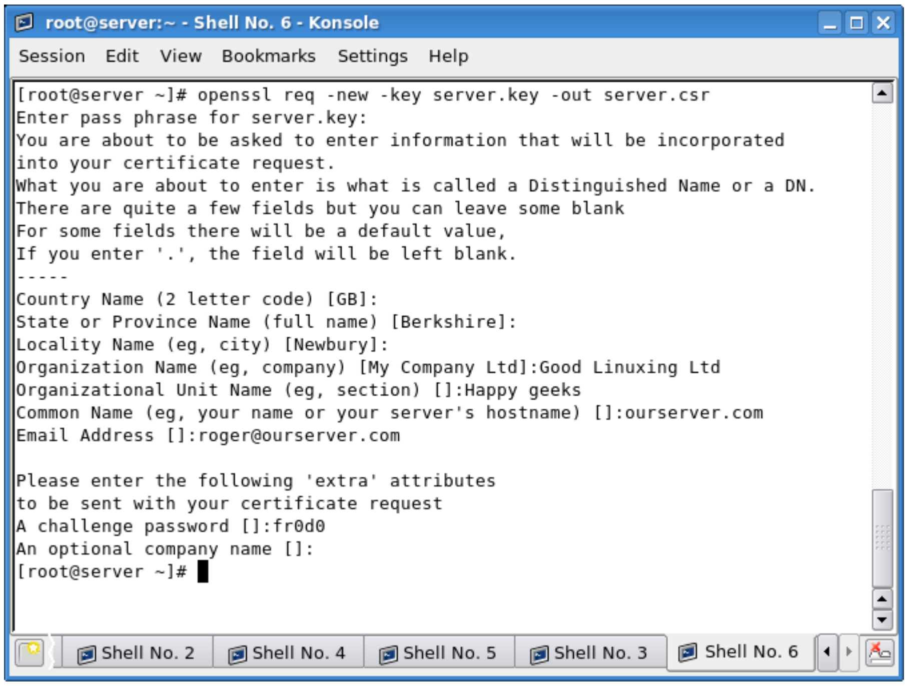
Again, we must provide a password before we can continue. Then again, we must go through an interactive form, providing details for our site. In a real life situation, a real CA would ask you for these details, whether via email, phone, an application form etc. For more details, please refer to Submission of CSR to CA sub-section below (5.5.2).


root@server:~ - Shell No. 6 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

```
[root@server ~]# openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:
```

Since our CA and our website are one and the same, the form will differ little from what we have done when creating the CA. This can be confusing. Therefore, you should remember that the *Common Name* for your CA should include the letters *CA* (or similar), to distinguish it from the server record.

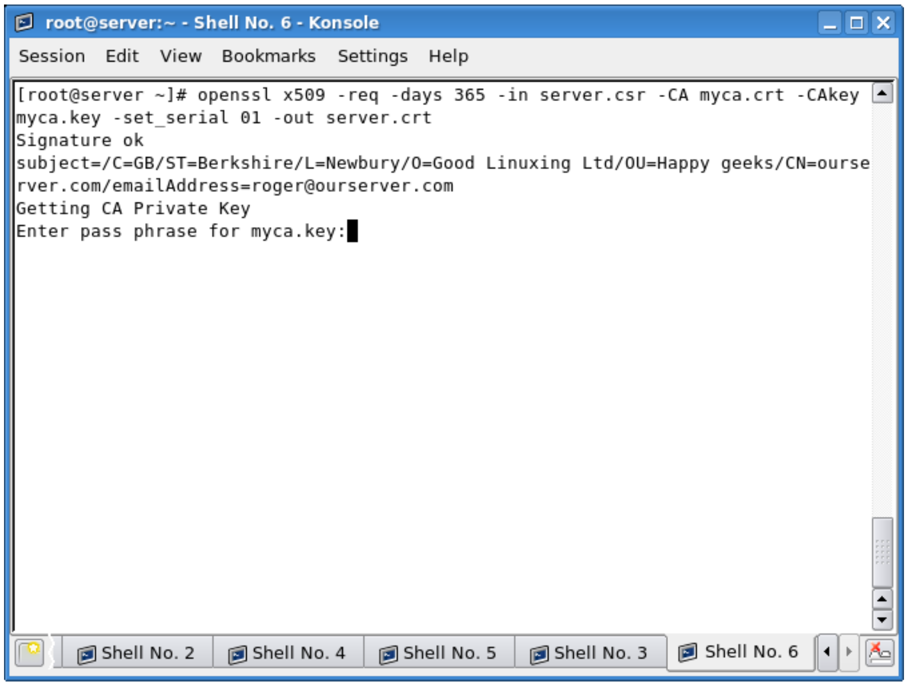
Lastly, you can provide an additional password for the server key, to make misuse more difficult.


root@server:~ - Shell No. 6 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

```
[root@server ~]# openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:
State or Province Name (full name) [Berkshire]:
Locality Name (eg, city) [Newbury]:
Organization Name (eg, company) [My Company Ltd]:Good Linuxing Ltd
Organizational Unit Name (eg, section) []:Happy geeks
Common Name (eg, your name or your server's hostname) []:ourserver.com
Email Address []:roger@ourserver.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:fr0d0
An optional company name []:
[root@server ~]# 
```

5.4.4.4 Sign Certificate Signing Request (CSR) with Certificate Authority (CA)

What remains to be done is to sign the CSR with the CA we have created. Once we do that, our certificate will be valid for the coming year. After that, we will have to renew it.


root@server:~ - Shell No. 6 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

```
[root@server ~]# openssl x509 -req -days 365 -in server.csr -CA myca.crt -CAkey
myca.key -set_serial 01 -out server.crt
Signature ok
subject=/C=GB/ST=Berkshire/L=Newbury/O=Good Linuxing Ltd/OU=Happy geeks/CN=ourse
rver.com/emailAddress=roger@ourserver.com
Getting CA Private Key
Enter pass phrase for myca.key:
```

You should be familiar with the syntax by now:

```
-CA myca.crt
```

This option instructs *openssl* to use our CA certificate.

```
-CAkey myca.key
```

This option instructs *openssl* to sign the certificate with the CA key.

```
-set_serial 01
```

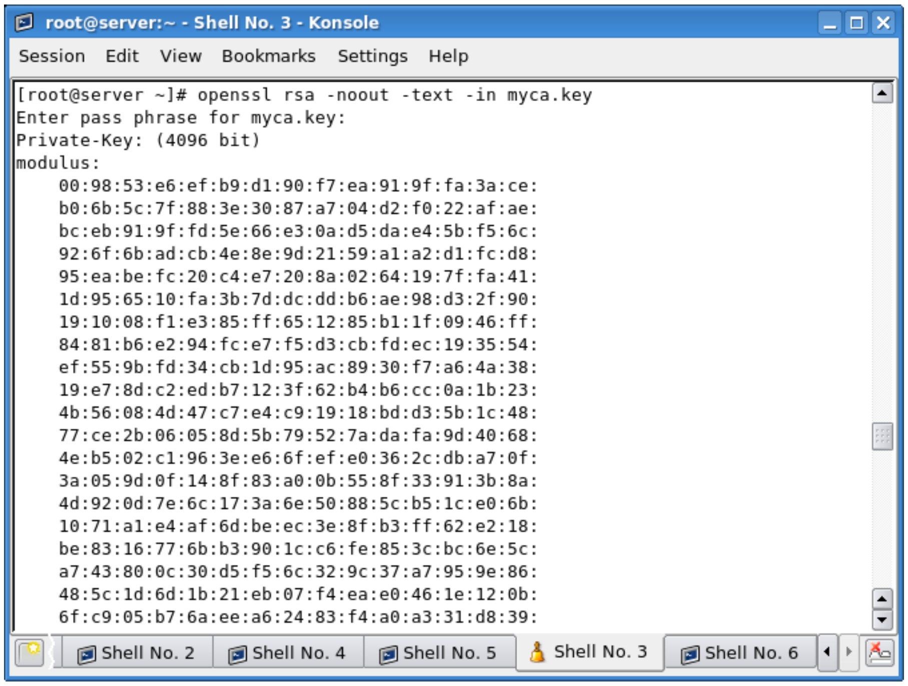
The *set_serial* option is used to create a serial number when outputting a self-signed certificate. This allows you to track the changes done to the certificate.

This covers the creation and signing of the SSL certificates.

5.4.4.5 Verify certificates

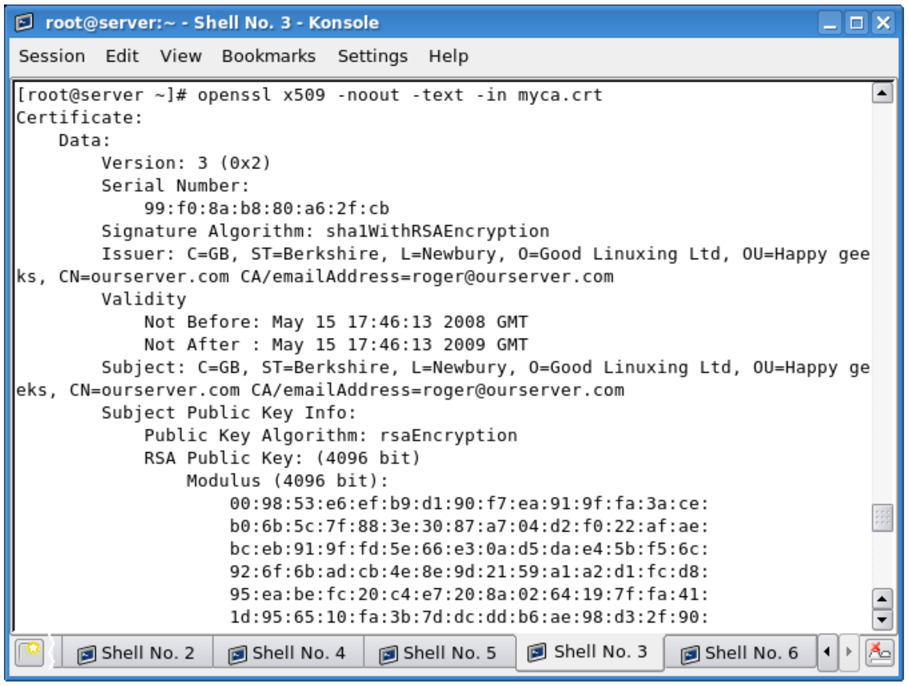
Let's examine the certificates we have just created. This can help you see if there are any problems with your files.

```
openssl rsa -noout -text -in myca.key
```


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

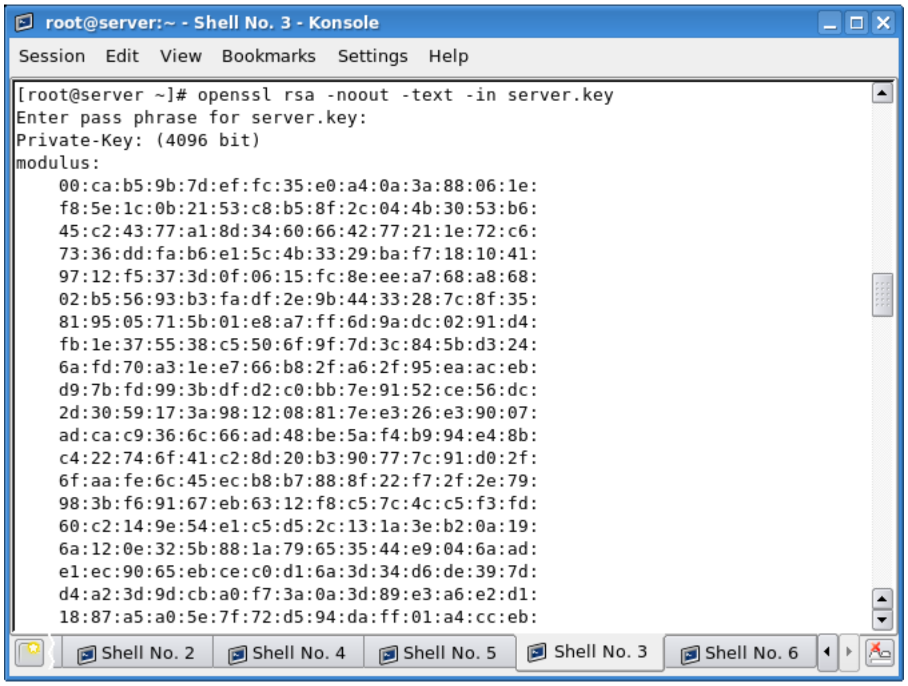
```
[root@server ~]# openssl rsa -noout -text -in myca.key
Enter pass phrase for myca.key:
Private-Key: (4096 bit)
modulus:
    00:98:53:e6:ef:b9:d1:90:f7:ea:91:9f:fa:3a:ce:
    b0:6b:5c:7f:88:3e:30:87:a7:04:d2:f0:22:af:ae:
    bc:eb:91:9f:fd:5e:66:e3:0a:d5:da:e4:5b:f5:6c:
    92:6f:6b:ad:cb:4e:8e:9d:21:59:a1:a2:d1:fc:d8:
    95:ea:be:fc:20:c4:e7:20:8a:02:64:19:7f:fa:41:
    1d:95:65:10:fa:3b:7d:dc:dd:b6:ae:98:d3:2f:90:
    19:10:08:f1:e3:85:ff:65:12:85:b1:1f:09:46:ff:
    84:81:b6:e2:94:fc:e7:f5:d3:cb:fd:ec:19:35:54:
    ef:55:9b:fd:34:cb:1d:95:ac:89:30:f7:a6:4a:38:
    19:e7:8d:c2:ed:b7:12:3f:62:b4:b6:cc:0a:1b:23:
    4b:56:08:4d:47:c7:e4:c9:19:18:bd:d3:5b:1c:48:
    77:ce:2b:06:05:8d:5b:79:52:7a:da:fa:9d:40:68:
    4e:b5:02:c1:96:3e:e6:6f:ef:e0:36:2c:db:a7:0f:
    3a:05:9d:0f:14:8f:83:a0:0b:55:8f:33:91:3b:8a:
    4d:92:0d:7e:6c:17:3a:6e:50:88:5c:b5:1c:e0:6b:
    10:71:a1:e4:af:6d:be:ec:3e:8f:b3:ff:62:e2:18:
    be:83:16:77:6b:b3:90:1c:c6:fe:85:3c:bc:6e:5c:
    a7:43:80:0c:30:d5:f5:6c:32:9c:37:a7:95:9e:86:
    48:5c:1d:6d:1b:21:eb:07:f4:ea:e0:46:1e:12:0b:
    6f:c9:05:b7:6a:ee:a6:24:83:f4:a0:a3:31:d8:39:
```

```
openssl x509 -noout -text -in myca.crt
```


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

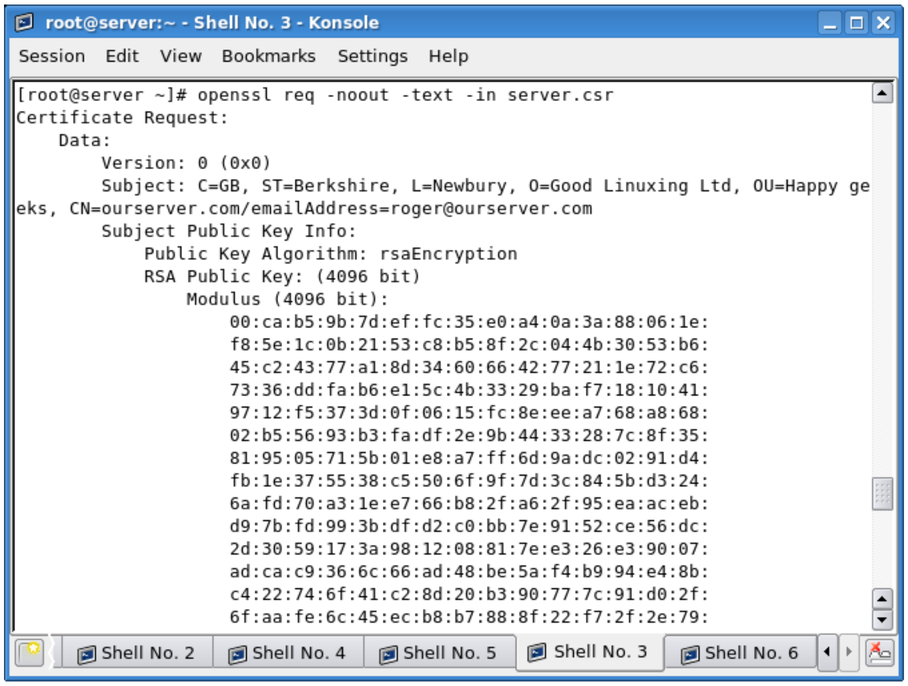
```
[root@server ~]# openssl x509 -noout -text -in myca.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            99:f0:8a:b8:80:a6:2f:cb
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=GB, ST=Berkshire, L=Newbury, O=Good Linuxing Ltd, OU=Happy gee
ks, CN=ourserver.com CA/emailAddress=roger@ourserver.com
        Validity
            Not Before: May 15 17:46:13 2008 GMT
            Not After : May 15 17:46:13 2009 GMT
        Subject: C=GB, ST=Berkshire, L=Newbury, O=Good Linuxing Ltd, OU=Happy ge
eks, CN=ourserver.com CA/emailAddress=roger@ourserver.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (4096 bit)
                Modulus (4096 bit):
                    00:98:53:e6:ef:b9:d1:90:f7:ea:91:9f:fa:3a:ce:
                    b0:6b:5c:7f:88:3e:30:87:a7:04:d2:f0:22:af:ae:
                    bc:eb:91:9f:fd:5e:66:e3:0a:d5:da:e4:5b:f5:6c:
                    92:6f:6b:ad:cb:4e:8e:9d:21:59:a1:a2:d1:fc:d8:
                    95:ea:be:fc:20:c4:e7:20:8a:02:64:19:7f:fa:41:
                    1d:95:65:10:fa:3b:7d:dc:dd:b6:ae:98:d3:2f:90:
```

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


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

```
[root@server ~]# openssl rsa -noout -text -in server.key
Enter pass phrase for server.key:
Private-Key: (4096 bit)
modulus:
    00:ca:b5:9b:7d:ef:fc:35:e0:a4:0a:3a:88:06:1e:
    f8:5e:1c:0b:21:53:c8:b5:8f:2c:04:4b:30:53:b6:
    45:c2:43:77:a1:8d:34:60:66:42:77:21:1e:72:c6:
    73:36:dd:fa:b6:e1:5c:4b:33:29:ba:f7:18:10:41:
    97:12:f5:37:3d:0f:06:15:fc:8e:ee:a7:68:a8:68:
    02:b5:56:93:b3:fa:df:2e:9b:44:33:28:7c:8f:35:
    81:95:05:71:5b:01:e8:a7:ff:6d:9a:dc:02:91:d4:
    fb:1e:37:55:38:c5:50:6f:9f:7d:3c:84:5b:d3:24:
    6a:fd:70:a3:1e:e7:66:b8:2f:a6:2f:95:ea:ac:eb:
    d9:7b:fd:99:3b:df:d2:c0:bb:7e:91:52:ce:56:dc:
    2d:30:59:17:3a:98:12:08:81:7e:e3:26:e3:90:07:
    ad:ca:c9:36:6c:66:ad:48:be:5a:f4:b9:94:e4:8b:
    c4:22:74:6f:41:c2:8d:20:b3:90:77:7c:91:d0:2f:
    6f:aa:fe:6c:45:ec:b8:b7:88:8f:22:f7:2f:2e:79:
    98:3b:f6:91:67:eb:63:12:f8:c5:7c:4c:c5:f3:fd:
    60:c2:14:9e:54:e1:c5:d5:2c:13:1a:3e:b2:0a:19:
    6a:12:0e:32:5b:88:1a:79:65:35:44:e9:04:6a:ad:
    e1:ec:90:65:eb:ce:c0:d1:6a:3d:34:d6:de:39:7d:
    d4:a2:3d:9d:cb:a0:f7:3a:0a:3d:89:e3:a6:e2:d1:
    18:87:a5:a0:5e:7f:72:d5:94:da:ff:01:a4:cc:eb:
```

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


root@server:~ - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3
Shell No. 6

```
[root@server ~]# openssl req -noout -text -in server.csr
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=GB, ST=Berkshire, L=Newbury, O=Good Linuxing Ltd, OU=Happy ge
eks, CN=ourserver.com/emailAddress=roger@ourserver.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (4096 bit)
                Modulus (4096 bit):
                    00:ca:b5:9b:7d:ef:fc:35:e0:a4:0a:3a:88:06:1e:
                    f8:5e:1c:0b:21:53:c8:b5:8f:2c:04:4b:30:53:b6:
                    45:c2:43:77:a1:8d:34:60:66:42:77:21:1e:72:c6:
                    73:36:dd:fa:b6:e1:5c:4b:33:29:ba:f7:18:10:41:
                    97:12:f5:37:3d:0f:06:15:fc:8e:ee:a7:68:a8:68:
                    02:b5:56:93:b3:fa:df:2e:9b:44:33:28:7c:8f:35:
                    81:95:05:71:5b:01:e8:a7:ff:6d:9a:dc:02:91:d4:
                    fb:1e:37:55:38:c5:50:6f:9f:7d:3c:84:5b:d3:24:
                    6a:fd:70:a3:1e:e7:66:b8:2f:a6:2f:95:ea:ac:eb:
                    d9:7b:fd:99:3b:df:d2:c0:bb:7e:91:52:ce:56:dc:
                    2d:30:59:17:3a:98:12:08:81:7e:e3:26:e3:90:07:
                    ad:ca:c9:36:6c:66:ad:48:be:5a:f4:b9:94:e4:8b:
                    c4:22:74:6f:41:c2:8d:20:b3:90:77:7c:91:d0:2f:
                    6f:aa:fe:6c:45:ec:b8:b7:88:8f:22:f7:2f:2e:79:
```

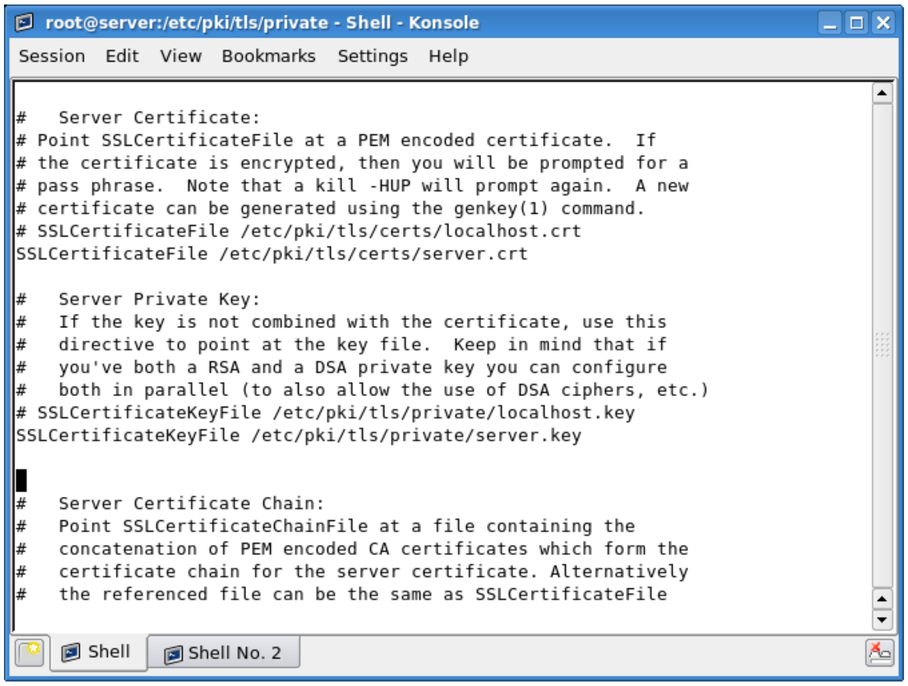
Everything looks good. Now, we can finish editing the *ssl.conf* file.

5.4.5 Edit ssl.conf configuration file - part 2

We now need to specify the location of our certificates and the keys in the *ssl.conf* file so the server can find and use them. We will comment out the sample entries in the file and use our own. Necessarily, we will have to copy the relevant files to their right location.

5.4.5.1 Server Certificate

This directive specifies the location of the server certificate (*server.crt*). On CentOS 5, the default location is */etc/pki/tls/certs*. We will use the same directory. Your choice may vary. The important thing to remember is to make the files unavailable to anyone but root.


root@server:/etc/pki/tls/private - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

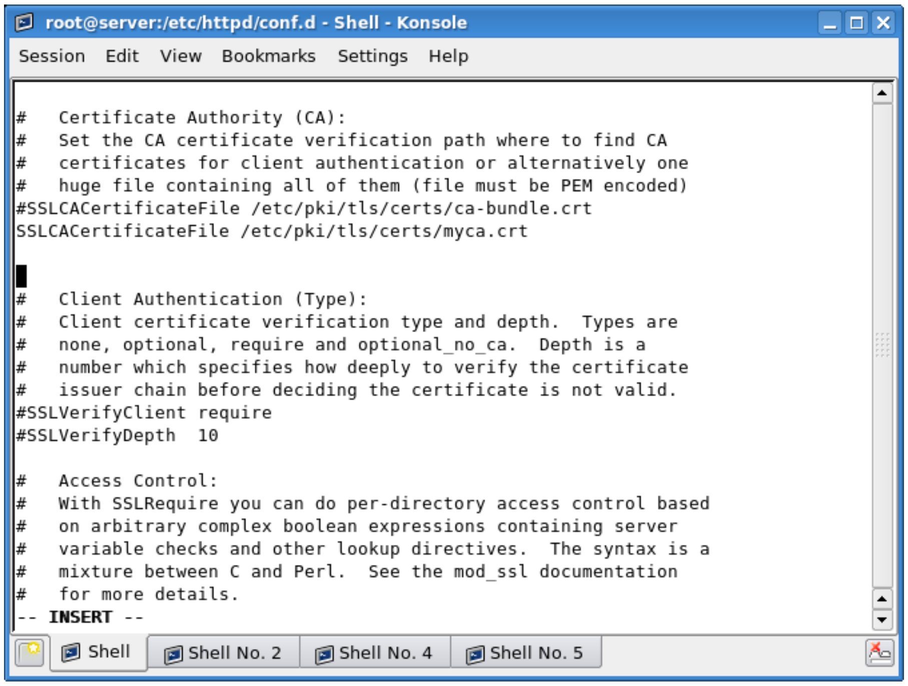
```
#   Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate.  If
# the certificate is encrypted, then you will be prompted for a
# pass phrase.  Note that a kill -HUP will prompt again.  A new
# certificate can be generated using the genkey(1) command.
# SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateFile /etc/pki/tls/certs/server.crt

#   Server Private Key:
#   If the key is not combined with the certificate, use this
#   directive to point at the key file.  Keep in mind that if
#   you've both a RSA and a DSA private key you can configure
#   both in parallel (to also allow the use of DSA ciphers, etc.)
# SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
SSLCertificateKeyFile /etc/pki/tls/private/server.key


#   Server Certificate Chain:
#   Point SSLCertificateChainFile at a file containing the
#   concatenation of PEM encoded CA certificates which form the
#   certificate chain for the server certificate. Alternatively
#   the referenced file can be the same as SSLCertificateFile
```

5.4.5.2 Server Private Key

This directive points to the location of the server key (*server.key*). Again, your choice should reflect your needs. See image above.

5.4.5.3 Certificate Authority

This directive specifies the location of the CA certificate.


root@server:/etc/httpd/conf.d - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5

```
#   Certificate Authority (CA):
#   Set the CA certificate verification path where to find CA
#   certificates for client authentication or alternatively one
#   huge file containing all of them (file must be PEM encoded)
#SSLCACertificateFile /etc/pki/tls/certs/ca-bundle.crt
SSLCACertificateFile /etc/pki/tls/certs/myca.crt


#   Client Authentication (Type):
#   Client certificate verification type and depth.  Types are
#   none, optional, require and optional_no_ca.  Depth is a
#   number which specifies how deeply to verify the certificate
#   issuer chain before deciding the certificate is not valid.
#SSLVerifyClient require
#SSLVerifyDepth  10

#   Access Control:
#   With SSLRequire you can do per-directory access control based
#   on arbitrary complex boolean expressions containing server
#   variable checks and other lookup directives.  The syntax is a
#   mixture between C and Perl.  See the mod_ssl documentation
#   for more details.
-- INSERT --
```

Now, let us copy the files to their relevant locations:

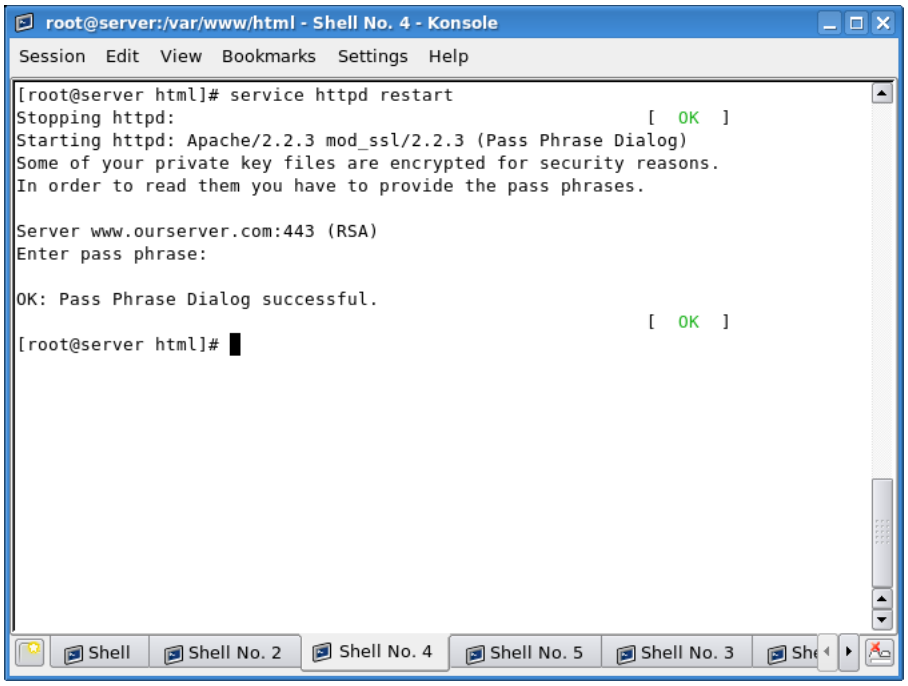
```
cp server.key /etc/pki/tls/private/server.key
cp server.crt /etc/pki/tls/certs/server.crt
cp myca.crt /etc/pki/tls/certs/myca.crt
```

We are ready. Let's test our setup.

5.4.6 Test setup

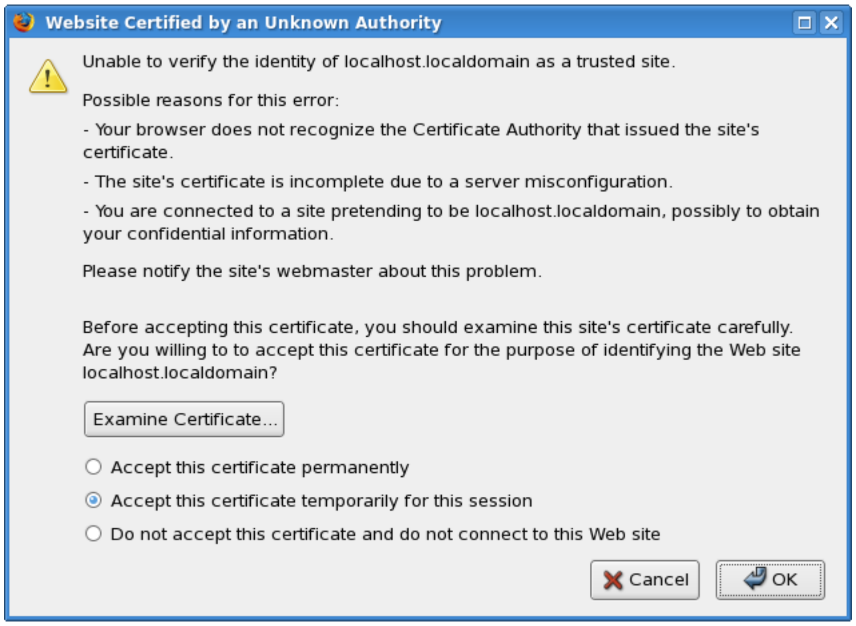
After saving the *ssl.conf* file, we need to restart our server.

```
service httpd restart
```


root@server:/var/www/html - Shell No. 4 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5
Shell No. 3

```
[root@server html]# service httpd restart
Stopping httpd:                                           [  OK  ]
Starting httpd: Apache/2.2.3 mod_ssl/2.2.3 (Pass Phrase Dialog)
Some of your private key files are encrypted for security reasons.
In order to read them you have to provide the pass phrases.

Server www.ourserver.com:443 (RSA)
Enter pass phrase:

OK: Pass Phrase Dialog successful.
                                                          [  OK  ]
[root@server html]# 
```

As you can see, we must provide a password before we can continue. Now, we will try to access our server by typing *https://www.ourserver.com* in the address line of a web browser. You will most likely receive a warning message.



Unable to verify the identity of localhost.localdomain as a trusted site.

Possible reasons for this error:

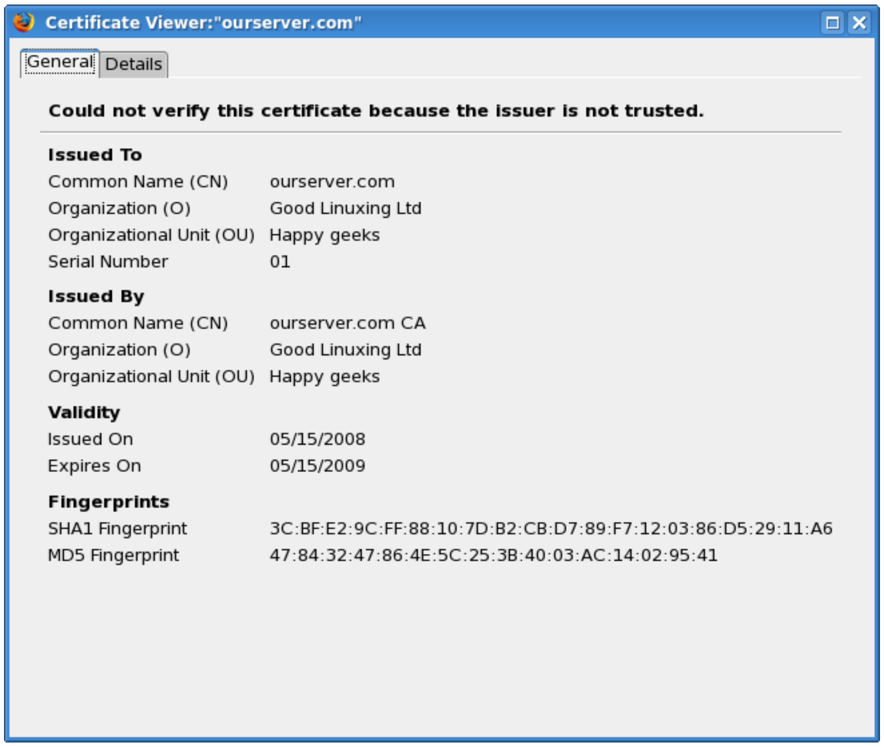
- Your browser does not recognize the Certificate Authority that issued the site's certificate.
- The site's certificate is incomplete due to a server misconfiguration.
- You are connected to a site pretending to be localhost.localdomain, possibly to obtain your confidential information.

Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to to accept this certificate for the purpose of identifying the Web site localhost.localdomain?

- Accept this certificate permanently
- [✓ Accept this certificate temporarily for this session]
- Do not accept this certificate and do not connect to this Web site

Let us examine the certificate before we accept it.



Could not verify this certificate because the issuer is not trusted.

Issued To

Common Name (CN)	ourserver.com
Organization (O)	Good Linuxing Ltd
Organizational Unit (OU)	Happy geeks
Serial Number	01

Issued By

Common Name (CN)	ourserver.com CA
Organization (O)	Good Linuxing Ltd
Organizational Unit (OU)	Happy geeks

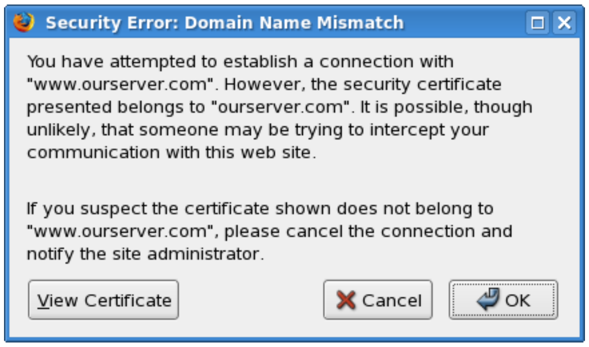
Validity

Issued On	05/15/2008
Expires On	05/15/2009

Fingerprints

SHA1 Fingerprint	3C:BF:E2:9C:FF:88:10:7D:B2:CB:D7:89:F7:12:03:86:D5:29:11:A6
MD5 Fingerprint	47:84:32:47:86:4E:5C:25:3B:40:03:AC:14:02:95:41

Indeed, everything looks fine. On the Web, though, very few people would be convinced by this certificate. But in our setup, it serves well. After accepting the certificate (either permanently or temporarily for this session only), you will hit yet another warning.

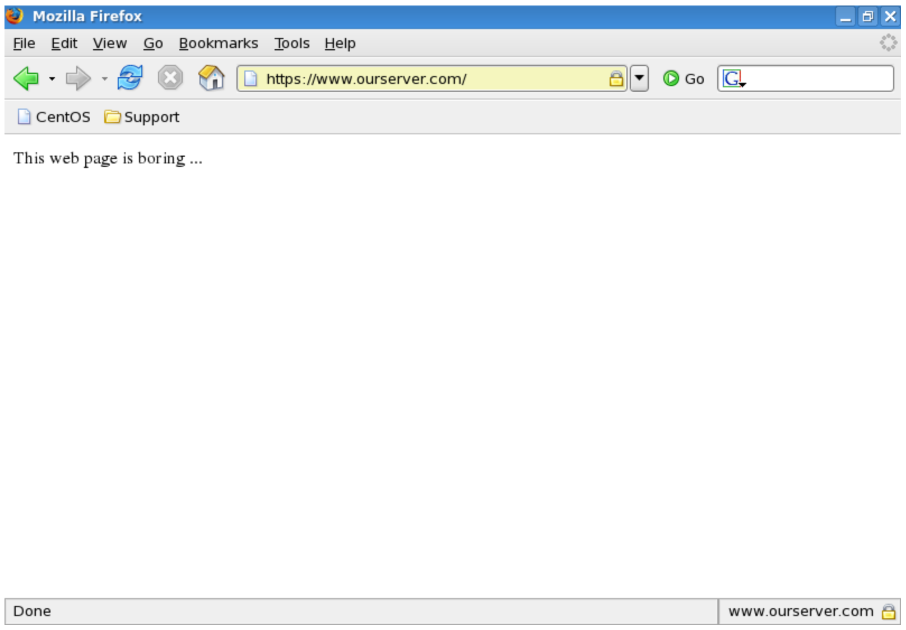


You have attempted to establish a connection with "www.ourserver.com". However, the security certificate presented belongs to "ourserver.com". It is possible, though unlikely, that someone may be trying to intercept your communication with this web site.

If you suspect the certificate shown does not belong to "www.ourserver.com", please cancel the connection and notify the site administrator.

This time, there's a mismatch between the domain name (*www.ourserver.com*) and the certificate (*ourserver.com*). This should not be an issue if you are using the DNS server, but we will discuss this separately in the next Part. For now, we will accept the certificate.

After that, we should reach our site safely. Our setup works.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://www.ourserver.com/
Go
CentOS Support
Done
www.ourserver.com

This web page is boring ...

5.4.7 Mini-summary

Setting up the secure Web server might seem a little confusing. Therefore, here's a mini summary that should clarify the setup process.

5.4.7.1 Names

This is a short list of file names used in this section:

File name	Description
myca.key	CA key
myca.crt	CA certificate
server.key	server key
server.csr	server CSR
server.crt	server certificate, signed by CA

5.4.7.2 Commands

Below, you can find a summarized list of commands you will need to run to create your certificate. Please note that the names I have used are generic and might not suit your needs. However, it is important that you use the name *server.key* for the server key file, to conform with Apache standards.

Command	**Description**
openssl genrsa -des3 -out myca.key 4096	Create CA key
openssl req -new -x509 -days 365 -key myca.key -out myca.crt	Create CA certificate
openssl genrsa -des3 -out server.key 4096	Create server key
openssl req -new -key server.key -out server.csr	Create CSR
openssl x509 -req -days 365 -in server.csr -CA myca.crt -CAkey myca.key -set_serial 01 -out server.crt	Sign CSR

5.4.7.3 Difference between self-signed and CA-signed certificates

This will help you better understand the differences between our exercise and a real, production setup.

Step	**Self-signed CA**	**Real CA**
Create CA key	Yes	No need
Create CA	Yes	No need
Create CSR	Yes	As instructed by CA
Create server key	Yes	Maybe: 1. CA creates key, signs it and sends to customer 2. CA creates CSR only and sends to customer, who then creates server key and signs with CA
Sign server key	Yes	Maybe: 1. CA sends signed key to customer 2. CA sends CSR; customer signs the key by himself/herself

5.4.7.4 Verification

You will have to run these commands to check your certificates and keys:

```
openssl rsa -noout -text -in server.key
openssl x509 -noout -text -in server.crt
openssl rsa -noout -text -in myca.key
openssl x509 -noout -text -in myca.crt
```

5.4.7.5 File names and locations

These are the locations of relevant files:

Full path and name	**Description**
/etc/httpd/conf.d/ssl.conf	Main configuration file
/etc/httpd/modules	Location of all modules
/etc/httpd/modules/mod_ssl.so	Location of *mod_ssl* module
/etc/pki/tls/certs	Location of server and CA certificates
/etc/pki/tls/private	Location of server keys

5.5 Extras

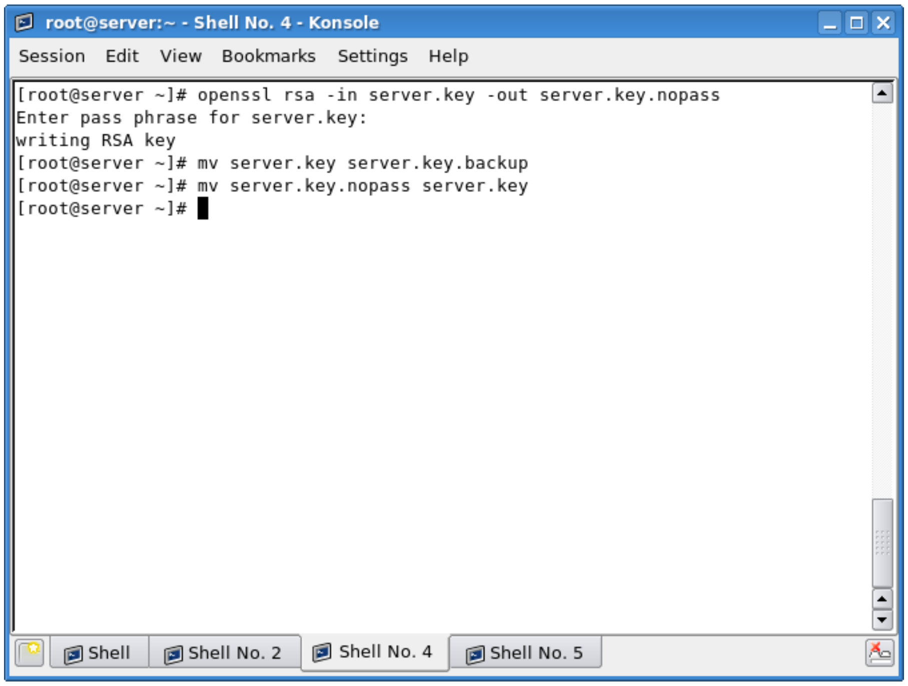
Now that we have our secure Web server running, let us review a number of other options.

5.5.1 Do not use password-protected server keys

As said before, the password protection for server keys is rather impractical, without significantly contributing to the server security. Therefore, you are encouraged not to use them. Of course, you must make sure that your server security is ensured by other means.

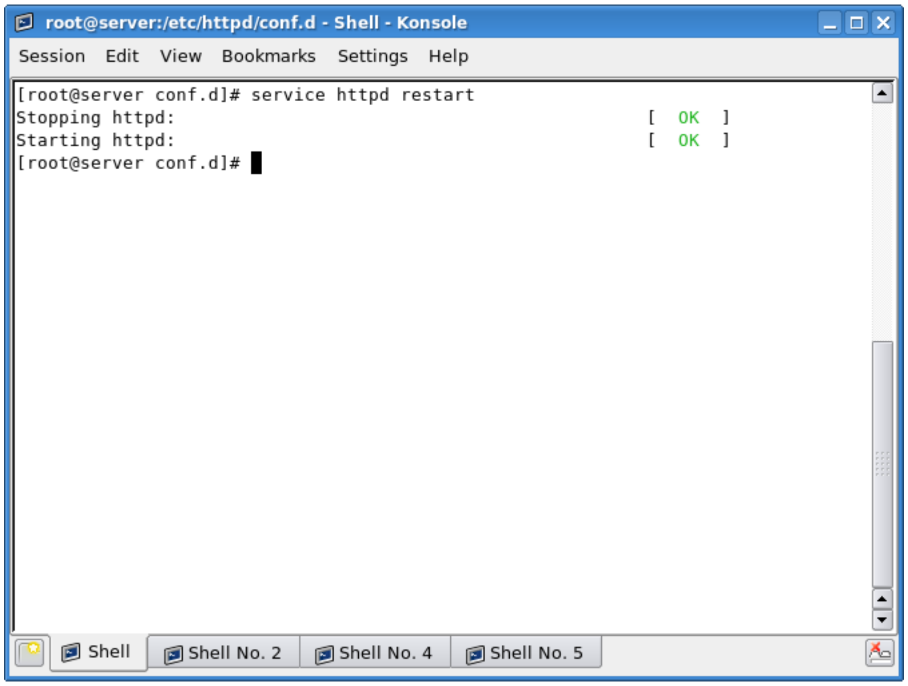
5.5.1.1 Create server key without password

We will "filter" the old key (*server.key*) into a new one (*server.key.nopass*), which will not include a password. Then, we will swap between the old and the new one. You are advised to keep a backup copy of the original key, just in case.


root@server:~ - Shell No. 4 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5

```
[root@server ~]# openssl rsa -in server.key -out server.key.nopass
Enter pass phrase for server.key:
writing RSA key
[root@server ~]# mv server.key server.key.backup
[root@server ~]# mv server.key.nopass server.key
[root@server ~]# 
```

Next time we restart the server, we won't be prompted for a password. All other settings remain unchanged.


root@server:/etc/httpd/conf.d - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 4
Shell No. 5

```
[root@server conf.d]# service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
[root@server conf.d]# 
```

5.5.2 Submission of CSR to CA

It is difficult to say what method the CA will use to certify your site. They might ask you for your credentials over a form or through an online form. Or they might ask you to submit a CSR, which they will sign, returning the key to you. Here, we will review a typical option. Again, this may not be the case you'll encounter.

5.5.2.1 Create CSR

Assuming the CA wants all records in the purely digital form, you will have to create a CSR. We have already done that. Optionally, they might ask you to convert the file into a Privacy Enhanced Mail (PEM) format. Please note that this format is not widely used and this will probably not be necessary.

```
openssl x509 -inform crt -in server.crt -out server.pem
```

Please note that you should consult your CA for detailed instructions regarding the conversions, if at all required.

5.5.2.2 Send CSR to CA

You will now have to submit the CSR file (or PEM) to the CA. Once the CA processes your application, you will receive the certificate back. The file will most likely be sent in the PEM format, so you will have to convert it back to CSR format. It will also most likely bear a different name from what you are used to, so you should rename it to *server.crt*, to conform with Apache conventions.

5.5.2.3 Verify certificate

Now, you should verify the certificate, against the relevant CA's file. You should receive this file from your CA. Alternatively, if your distro includes a list of CAs, you might try that one. On CentOS 5, a bundle containing a list of known CAs is located under */etc/pki/tls/certs*.

```
openssl verify -CAfile ca-bundle.crt →
   -purpose sslserver server.crt
```

Next, you should verify that the certificate corresponds to the private key. Please make sure the names match.

```
openssl x509 -noout -modulus -in server.pem | openssl md5
openssl rsa -noout -modulus -in server.crt | openssl md5
```

Once you have completed the above steps, you will have to edit the *ssl.conf* file, restart the server and test your setup. We're back on familiar grounds.

For more details regarding the different commands and verification, please consult:

- verify man page
- OpenSSL: Documents, req

5.6 General considerations

Here's a number of settings that you should remember.

5.6.1 Use secure server only

If you want your server to serve only secure pages, simply comment the *Listen 80* directive from the *httpd.conf* file. Sites dedicated to clients privacy and security should not run any other, non-secure content.

5.6.2 Use only IP-based virtual hosts

It is impossible to use named-based virtual hosts with the secure Web server. This is because the SSL handshake occurs before the HTTP request can identify the named-based virtual host. Using names will result in errors. You may only use IP addresses in the *VirtualHost* directives inside the ssl.conf configuration file.

5.6.3 Use server.key as file name for the server key

It is not strictly necessary, but it is good practice and in line with Apache conventions to use the *server.key* name for the server key file. If for some reason you require paranoid security, then you might change this name to something less obvious.

Chapter 6

Other configurations

6.1 Firewall rules

Here's the most basic pair of rules to allow HTTP traffic (including both secure and non-secure):

```
iptables -A INPUT -p tcp -dport 80 -j ACCEPT
iptables -A INPUT -p tdp -dport 443 -j ACCEPT
```

However, you can restrict the traffic even more. For example, you can allow incoming connections only to a certain interface:

```
iptables -A INPUT -p tcp -dport 80 -i eth0 -j ACCEPT
iptables -A INPUT -p tdp -dport 443 -i eht0 -j ACCEPT
```

Finally, you can restrict the traffic to a specific subnet, allowing only certain machines to connect:

```
iptables -A INPUT -p tcp -dport 80 -i eth0 →
   -s 192.168.1.0/24 -j ACCEPT
iptables -A INPUT -p tcp -dport 443 -i eth0 →
   -s 192.168.1.0/24 -j ACCEPT
```

You can further sharpen the rules by specifying source ports and packet states. We'll see this in the next section.

6.1.1 Advanced firewall rules

Sometimes, your Web server might not be an external client; it will connect to the Internet through a dedicated, firewalled gateway. This brings about several issues, which include the Network Address Translation (NAT) and port forwarding.

We have discussed the forwarding and masquerading in Part ?: Dynamic Host Configuration Protocol (DHCP) server - ISC DHCPD (). Now, we will look at a more complex setup.

Let's assume that your Web server is a local machine, with a local IP address. It serves both internal and external clients. The internal setup is rather simple. We need to make sure external clients can connect, too.

We will need to allow traffic destined to ports 80 and 443 of our external IP address (let's assume 1.1.1.1) to be forwarded to ports 80 and 443 of our Web server, which resides on a local address (192.168.1.128). Furthermore, interfaces are *eth0* for the Internet and *eth1* for the intranet.

Our setup will include several steps:

- We will have to "forward" our web ports, so that clients behind the gateway will be able to accept incoming communications.
- We will have to setup some sort of masquerading, only this time we will use methods slightly different from those we have adopted when setting up the DHCP server.

6.1.1.1 Port forwarding

Our gateway is configured to forward communications between internal and external hosts, with new, established and related connections permitted outbound and established and related connections permitted inbound. This is a great setup for a server-less network, but it won't do for Apache.

Here are the original rules, which we have setup for the DHCP server acting as a gateway:

```
iptables -A FORWARD -t filter -i eth1 -m state →
   -state ESTABLISHED,RELATED -j ACCEPT

iptables -A FORWARD -t filter -o eth1 -m state →
   -state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Now, we need these rules to make it work:

```
iptables -I FORWARD -p tcp -i eth1 -o eth0 →
   -d 192.168.1.128 -dport 80 →
   -m state -state NEW -j ACCEPT

iptables -I FORWARD -p tcp -i eth1 -o eth0 →
   -d 192.168.1.128 -dport 443 →
   -m state -state NEW -j ACCEPT
```

If you want to tighten the rules some more, you can also specify the source ports:

```
iptables -I FORWARD -p tcp -i eth1 -o eth0 →
   -d 192.168.1.128 -dport 80 -sport 1024:65535 →
   -m state -state NEW -j ACCEPT
```

We have placed the rules on the top of the chain, so they would be processed before the existing rules, which only allow new outbound connections. Basically, these rules are sufficient if your gateway is servicing a number of local networks, all of which can fully resolve one another's IP addresses. They are not good enough for the Internet, though.

6.1.1.2 Destination NAT

Using masquerading the way we did when we configured the DHCP server might not be good enough for us, because it will point all traffic to the default network interface. We have to use a more sophisticated method, which is the DNAT. Now, we need to allow

new incoming connections to our Web server to properly resolve to the right client, on the right ports.

```
iptables -t nat -A PREROUTING -p tcp -i eth1 -d 1.1.1.1 →
   -dport 80 -to-destination 192.168.1.128:80 -j DNAT

iptables -t nat -A PREROUTING -p tcp -i eth1 -d 1.1.1.1 →
   -dport 443 -to-destination 192.168.1.128:443 -j DNAT
```

Of course, you can use non-default ports on the Web server, like 8080 or anything alike, which makes the idea of port forwarding even more meaningful.

To reiterate, IP masquerading is good enough for "normal" browsing, but servers behind the firewall also require port forwarding. Casual peer-to-peer (P2P) home users behind routers often have to do this to make their programs work.

6.1.1.3 Static NAT

If you have more than one publicly visible IP address, you won't be able to use IP masquerading. This is because masquerading forces all traffic to the default network interface on the firewalled gateway, resulting in a single usable external IP address.

However, it is quite likely that you will want to run your servers on separate hosts, with different both internal and external addresses, both to shape your traffic in a more orderly fashion and reduce the workload on specific hosts. To this end, you will have to use SNAT rather than IP forwarding.

The basic principle remains the same, except that you use separate external IP addresses for individual hosts, groups of hosts or the entire local network, as you see fit. In our example, we will demonstrate SNAT by creating a private rule for the Web server and a general rule for all other clients.

Let's assume the Web server will use a public IP address of 1.1.1.1, while all other clients will use 1.1.1.2.

Web server rules:

These two rules are required to allow DNAT and SNAT for the client running the Web server. Please note that these two rules do not specify what kind of servers are running on the particular client. This grants you extra flexibility, if you need to run more than one server on a particular machine.

```
iptables -t nat -A PREROUTING -d 1.1.1.1 -i eth1 →
   -to-destination 192.168.1.128 -j DNAT

iptables -t nat -A POSTROUTING -s 192.168.1.128 -o eth1 →
   -to-source 1.1.1.1 -j SNAT
```

As said, the forwarding rules from before remain valid, both the specific rules for the Web server, which permit new inbound connections, and the general rules, which permit only new outbound connections.

General rules:

This rule applies to all local network clients, trying to communicate with the external network.

```
iptables -t nat -A POSTROUTING -s! 192.168.1.128 -o eth1 →
   -to-source 1.1.1.2 -j SNAT
```

6.2 Enable Web server on startup

You will most likely want your Apache server to run on startup. The simplest way to enable this is to use the *chkconfig* utility.

```
chkconfig -levels 5 httpd on
```

Chapter 7

Security

Web server security is one of the most important things in your setup. If your server becomes compromised, you run the risk of serving malicious, fraudulent or simply tasteless pages to hundreds and thousands of your visitors. Furthermore, you risk exposing the privacy of your clients and users. Forum names and passwords, email addresses, sensitive records, credit card numbers, and other information could all be leaked out, creating an identity theft nightmare.

It is paramount that you keep your Web server in tiptop shape at all times. This requires lots of work, attention and responsibility and is not a trivial task. Running a good server takes time and patience.

It is also important that you be constantly aware of what goes about on your server. If you have several users uploading material to their individual directories, you are advised to make sure that they do not post potentially dangerous content. Most Linux users are oblivious to the web threats, but a large percentage of Windows users have a hard time with sites loaded with malicious payload. As the server owner and administrator, it is your responsibility to make sure that your visitors are not at risk.

Let us review some of the most crucial settings that you should pay attention to make sure both your server and your clients are secure.

7.1 Updates

Keep your server up to date at all times. Make sure you patch new vulnerabilities instantly. You are advised to subscribe to the Apache HTTP Server Mailing Lists for information about new bugs, updates, features, and more.

7.2 Hide your server version

This is a "security through obscurity" measure. Nevertheless, it does not hurt to use it. At the very least, this will annoy and delay a potential attacker, by making his attempts to harvest server information more difficult.

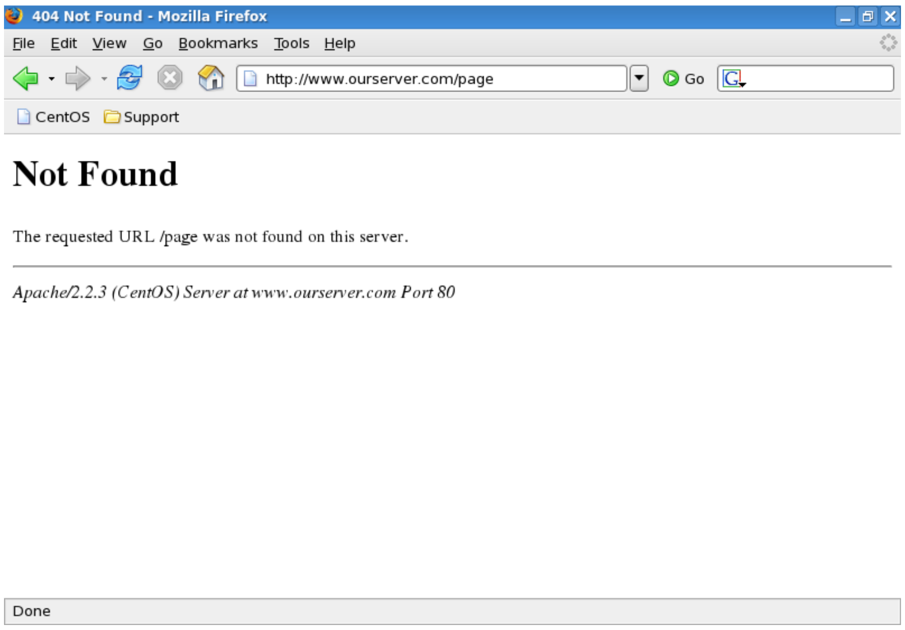
To remove server information, you will need to use these two directives:

```
ServerSignature Off
ServerTokens Prod
```

The first directive, *ServerSignature*, will remove the server version information from the pages generated by the server, like error pages

(403 Forbidden, 404 Not found), directory listings and others. The second directive, *ServerTokens*, will change the server's HTTP Response Header. By default, with the directive set to *OS*, the header will disclose both the version and the operating system. Set to *Prod*, the header will merely report *Apache*.

Here's an example without these directives.

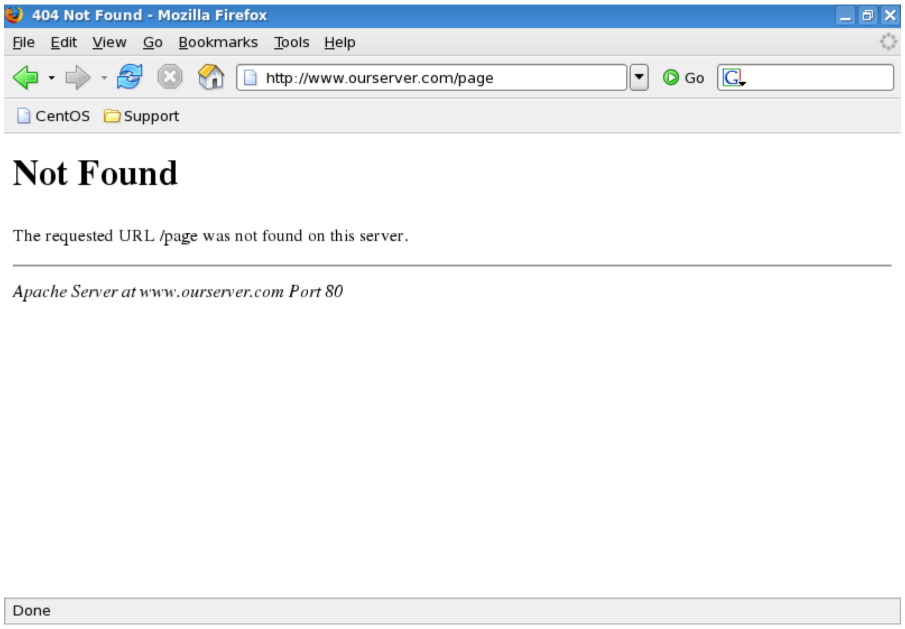

404 Not Found - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/page
Go
CentOS
Support
Done

Not Found

The requested URL /page was not found on this server.

Apache/2.2.3 (CentOS) Server at www.ourserver.com Port 80

And here's with the directives in place. The version is disguised.


404 Not Found - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.ourserver.com/page
Go
CentOS
Support
Done

Not Found

The requested URL /page was not found on this server.

Apache Server at www.ourserver.com Port 80

7.3 Logs

You should check your logs at least daily. This may be tedious and boring, but it is vital that you discover any potential breaches as quickly as possible. Keep an eye on things and look for suspicious directories and files.

7.4 Permissions

Badly implemented permissions can ruin your entire security. It is critical that you make sure the executables, configuration files, logs, access files and private keys are located outside the public HTML directories and writable only by root. The web pages should be

readable and possibly executable by your visitors but only writable by their respective owners.

Let us review the necessary permissions:

Location	**Permissions**
/usr/sbin/httpd	F: 511
/etc/httpd	D: 751
/etc/httpd/conf	D: 751, F: 644 / 600
/etc/httpd/conf.d	D: 751, F: 644 / 600
/etc/httpd/logs	symbolic link, 755 / 711
/etc/httpd/modules	symbolic link, 755 / 711
/etc/httpd/run	symbolic link, 755 / 711
/usr/lib/httpd/modules	D: 751, F: 644 / 600
/var/log/httpd	D: 751, F: 644 / 600
/var/run	D: 751, F: 644 / 600
/var/www/html	D: 755, F: 755

If you are really paranoid, then you should ONLY allow root access to the binaries and configuration files. It really depends on your setup and needs. Last but not the least, let's not forget that system files MUST be owned by root.

7.5 Access to root (/)

You must not allow anyone to access the *root* directory. Therefore, you should implement a default deny policy for the root and all subdirectories and then partially allow access to specific locations, like the public HTML directories of your users.

```
<Directory />
    Order Deny, Allow
    Deny from all
</Directory>
```

7.6 AllowOverride

This directive specifies if options used in the *.htaccess* files can conflict (and thus override) the settings configured for the particular directory. In general, you should set this directive to *none* and only permit specific tasks to a small number of trusted users. If you lease your server to numerous clients who must have some sort of protection for their content, then you can allow them to use the *.htaccess* file for authentication, as we have shown before (4).

```
AllowOverride none
```

7.7 Disable public access to .ht files

You must not allow any public user to be able to load the *.htaccess* file through the browser window. Furthermore, the *.htpasswd* file, which contains the user names and passwords, must also be protected from public access. Again, we have discussed this before, but it doesn't hurt to repeat it.

```
<Files ~ "^\.ht">
    Order allow, deny
    Deny from all
</Files>
```

7.8 Dynamic content

You need to be very careful with dynamic pages and scripts, since they allow server to perform a variety of operations that static HTML files cannot do. You should think twice before you allow any user to run scripts from his/her own public directory. In general, scripts are only allowed in special directories, as defined by the *ScriptAlias* directive.

7.8.1 Disable CGI

You should disable scripts in user directories. If you really must permit them, use the *Options +ExecCGI* directive.

```
Options -ExecCGI
```

7.8.2 Disable Server Side Includes (SSI)

SSI is a simple scripting language that allows web servers to display variables or execute other programs from within web pages. This introduces a significant load on the server and poses a security risk. You should not allow SSI to function on your server unless absolutely necessary.

```
Options -Includes
```

For more about SSI, please refer to Server Side Includes on Wikipedia.

7.9 Disable unnecessary modules

By default, Apache is very permissive when it comes to the functionality it offers. You should comment out any modules you do not need or wish to use from the */etc/httpd/conf/httpd.conf* file.

7.10 Use ModSecurity (mod_security) module

ModSecurity is a powerful Web Application Filter (WAF), which allows you to greatly enhance the security of your server by to detecting and preventing attacks before they reach web applications. ModSecurity can perform a variety of tasks, including detection of HTTP protocols violation, detection against common web attacks, detection of bots and crawlers, detection of Trojan horses, filtering based on existing rulesets, policies or regular expressions, and more. The application relies on generic rules to detect and prevent exploits and does not rely on blacklists or signatures.

Furthermore, it will not throttle the traffic throughput. And best of all, ModSecurity is very easy add to an existing and running Apache server.

Setting up and configuring ModSecurity is outside the scope of this Part. However, it will be reviewed separately, in the context of Network Security. If you are interested, you can read Part ?, Chapter ?: ModSecurity ().

Meanwhile, for more details, you should refer to ModSecurity: Open Source Web Application Firewall.

7.11 Chroot Jail

Like with many other services, it is possible to "sandbox" *httpd* to run in a virtual prison. The service will think its files are located in the default directories. In reality, they will reside inside a Chroot Jail, which will mimic the layout of the real directories, except that many of the files normally found under the real root won't be there, preventing possible privilege escalation risks.

This is an extremely important security feature and should be used whenever possible. For more details about Chroot Jail, please refer to Part ?: Chroot Jail, Chapter ?: Apache in Chroot Jail () for details how to create this setup.

For more information, you are also welcome to read: Apache in a chroot jail.

7.12 Secure web server only

The secure Web servers will offer their certificates to any client that asks for them. This means that you do not care about who your clients are. However, if your setup also requires that only a limited number of clients be allowed to access the secure content, you might consider using any one or several of the following methods: Kerberos, firewall rules, TCP wrappers, *allow* & *deny* directives, *.htaccess* files, client certificates, and more.

7.12.1 Different DocumentRoot

You are advised to use a separate directory for the secure pages. This allows you to fully separate normal content from secure, privileged data and might even mitigate potential exposure in case of an attack.

7.12.2 Permissions

The *server.crt* and the *server.key* file must only be readable by root. You should even disallow the root user from making any changes to the files. Set the permissions for these two files to 0400.

The permissions for the *ssl.conf* should be in line with your policy, which should be either 640 or 600.

7.12.3 Duration of certificates

Normally, certificates are issued for a year. However, if you are running an ultra-paranoid setup, you may want to make the certificates expire after only a few weeks. This means that potential attackers will always have only a limited access to data for a short

period of time, in case they succeed in decrypting your ciphers. You will force them to work all over again, trying to decode your data. Needless to say, you will most likely use self-signed certificates for this type of work.

7.13 Word of caution

This Part of the book cannot possibly encompass all the security aspects of running Apache. Do not presume you know everything about running a Web server just by reading a few pages here and there - I sure do not. Keep in mind that you are responsible for every bit of traffic flowing to and from your server - and then branch from there. Updates and logs are your best friends. Then, there's the matter of ethics. Your websites could be servings hundreds, thousands or even millions of people every day. The choices you make regarding your content and security policies may affect all of them.

Needless to say, nothing can replace years of hard work and experience.

Chapter 8

Additional resources

Here, you will find a number of additional links to useful resources that can help you better learn and understand the Apache Web server. You are warmly encouraged to visit them.

- The Apache Software Foundation
- The Apache HTTP Server Project
- The Apache Interface to OpenSSL
- Security Tips - Apache HTTP Server

Chapter 9

Exercises

This section is meant to help you estimate your knowledge of the Apache Web server, now that you have read the Part and worked through every step (you have, haven't you?). If you can successfully answer all of the questions below with ease, this probably means you possess a decent knowledge of the Apache Web server and you can most likely create a setup of your own without too much difficulty.

Please note that this short questionnaire is by no means an exhaustive or any sort of format test to Apache expertise. It should be a good starting point for broadening your knowledge and experience.

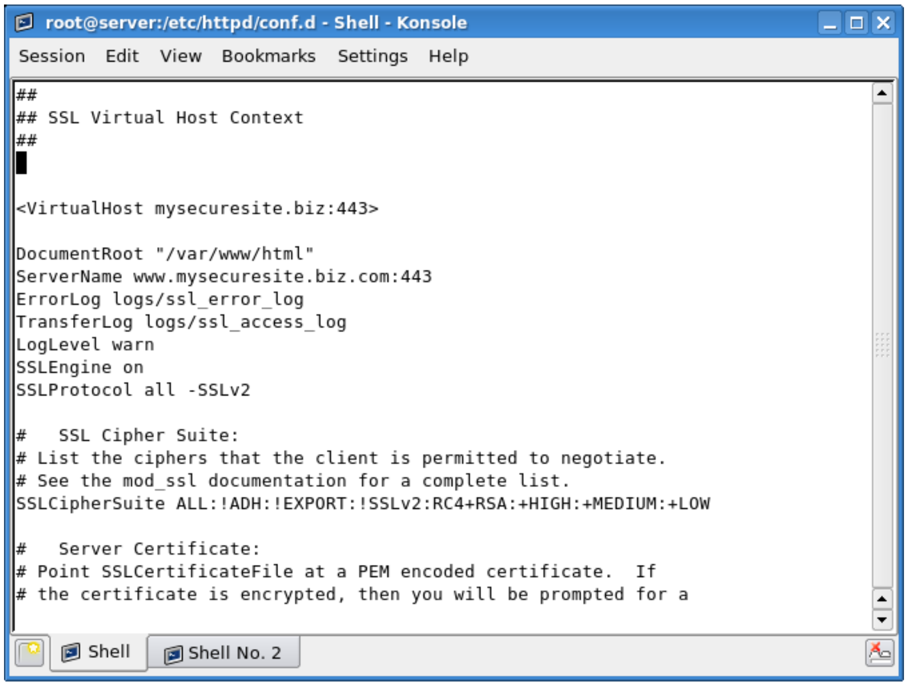
When trying to solve the questions, please do not look into the answers right away. Try to think and figure out the problem on your own. The questions might also help you identify the stronger and weaker points in your freshly earned Apache skills.

Furthermore, you are encouraged to simulate the problems by powering up Linux and practicing for real.

9.1 Questions

9.1.1 Secure Web server & VirtualHost

You want to configure a secure Web server that you will use for online transactions with your customers. You have completed all the steps successfully, but your server does not work, for some reason. Looking at the *VirtualHost* block in the *ssl.conf* file, you are unable to find the source of the problem. Can you tell what's wrong?


root@server:/etc/httpd/conf.d - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
##
## SSL Virtual Host Context
##


<VirtualHost mysecuresite.biz:443>

DocumentRoot "/var/www/html"
ServerName www.mysecuresite.biz.com:443
ErrorLog logs/ssl_error_log
TransferLog logs/ssl_access_log
LogLevel warn
SSLEngine on
SSLProtocol all -SSLv2

#   SSL Cipher Suite:
# List the ciphers that the client is permitted to negotiate.
# See the mod_ssl documentation for a complete list.
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW

#   Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate.  If
# the certificate is encrypted, then you will be prompted for a
```

Please note that there is an error only within the displayed directives in the screenshot. The *VirtualHost* container is properly closed and all directives have the correct syntax.

9.1.2 Directory, Files and Locations

You have used the *AllowOverride* directive inside a *Location* block to setup the behavior of the *.htaccess* file. However, your server ignores the change. What might be the reason?

```
<Location /dir/>
    Order allow, deny
    Allow from all
    Deny from plush-mush.info
    AllowOverride AuthConfig
</Location>
```

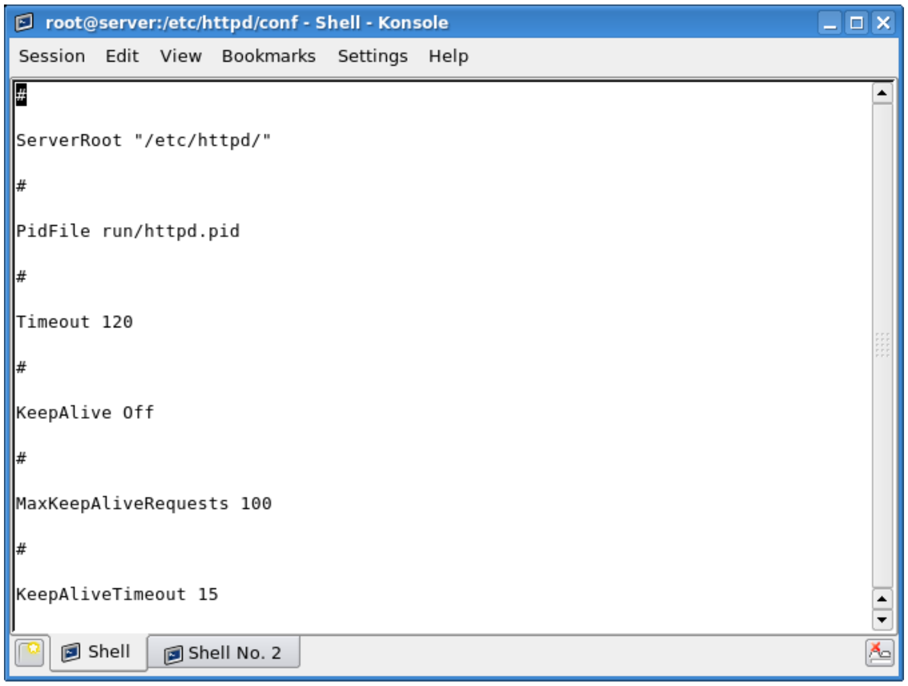
1. *AllowOverride* must be placed first inside any block.
2. *AllowOverride* can only be used inside Directory tags.
3. You have not restarted the server.
4. You need to specify the location of the *.htaccess* file in the opening *Location* tag.

9.1.3 Server functionality, 1

You want to use CGI scripts on your pages, but the server simply refuses to comply. You are absolutely convinced your syntax is correct. What could be the likely reason for your woes?

9.1.4 Server functionality, 2

Something is horribly wrong with your *httpd.conf* file. Nothing works. What's the matter?


root@server:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
#

ServerRoot "/etc/httpd/"

#

PidFile run/httpd.pid

#

Timeout 120

#

KeepAlive Off

#

MaxKeepAliveRequests 100

#

KeepAliveTimeout 15
```

9.1.5 .htaccess

What security precautions must you take to make sure authentication via *.htaccess* is not compromised easily?

1. Place the *.htpasswd* in a restricted directory inaccessible from the web.
2. Place the *.htaccess* file in a restricted directory and *chmod* it to 755.
3. Use a *Files* or *FilesMatch* directive to restrict access to *.ht* files.
4. Use a reasonably strong password for user authentication.
5. Change the *Listen 80* directive to a random number.

9.2 Answers

9.2.1 Secure Web server & VirtualHost

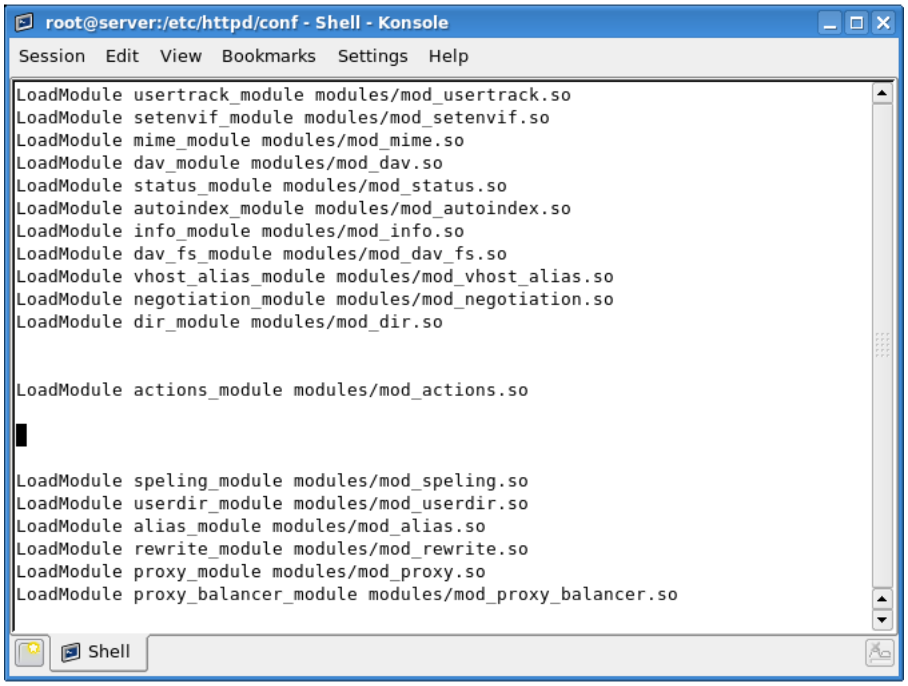
It is impossible to use named-based virtual hosts with the secure Web server. This is because the SSL handshake occurs before the HTTP request can identify the named-based virtual host. Using names will result in errors. You may only use IP addresses in the *VirtualHost* directives inside the ssl.conf configuration file.

9.2.2 Directory, Files and Locations

The right answer is 2. If you have read the Apache documentation, as suggested in the links provided, you would have realized this critical detail. The *AllowOverride* directive can only used inside *Directory* tags. However, the right answer might also be derived by running a few simple tests. Answer 3 is the trickiest, because it implies one of the classic mistakes. However, running a simple test will verify this is not the case. Answer 1 is misleading, especially since the *Order* directive was used - for which, the precedence does matter. Again, a quick test will prove this false. The last answer is pure nonsense.

9.2.3 Server functionality, 1

The CGI functionality is provided by the mod_actions module. You need to check that the httpd.conf file contains a LoadModule directive for this module.


root@server:/etc/httpd/conf - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
LoadModule usertrack_module modules/mod_usertrack.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule mime_module modules/mod_mime.so
LoadModule dav_module modules/mod_dav.so
LoadModule status_module modules/mod_status.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule info_module modules/mod_info.so
LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule vhost_alias_module modules/mod_vhost_alias.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule dir_module modules/mod_dir.so


LoadModule actions_module modules/mod_actions.so




LoadModule speling_module modules/mod_speling.so
LoadModule userdir_module modules/mod_userdir.so
LoadModule alias_module modules/mod_alias.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
```

Furthermore, you need to check that the module can indeed be found under: */usr/lib/httpd/modules*.

9.2.4 Server functionality, 2

You have placed a trailing slash at the end of the *ServerRoot* directive. This will cause all your symbolic links to fail. In the file comments that have been removed (a BAD idea!), there was a notice warning against placing the trailing slash at the end of the directive.

You should never add a slash at the end of the directory path - and ever delete comments.

9.2.5 .htaccess

The right answers are 1, 3 and 4. The *.htaccess* file must be placed in public directories, thus placing them in non-web directories would invalidate their purpose. This makes answer no. 2 wrong. Furthermore, the executable bit in the permissions is completely unnecessary for the functionality of the file.

The connection to the Web server (and the relevant port) is established before any authentication takes place. Moreover, the choice of the port number has no bearing whatsoever on the functionality of the *.htaccess* files. Thus, answer no. 5 is wrong.