Onlinevarsity
your e-way to learning

Using Apache Web Server

Are you □□□□□□□□□□ with Onlinevarsity.com?	R G E T R E E S I D
Did you □□□□□□□□ this ebook?	D O O W L N A D
Do you have a □□□□□ copy of this ebook?	L G A E L
Are you a victim of □□□□□□?	P C I A R Y

Answers: REGISTERED, DOWNLOAD, LEGAL, PIRACY

Download a **LEGAL** copy of this ebook which you can say is **mine**

because **PIRACY** is a **crime**

Using Apache Web Server Learner's Guide

APTECH LIMITED
Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2013

Dear Learner,

We congratulate you on your decision to pursue an Aptech course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc*. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

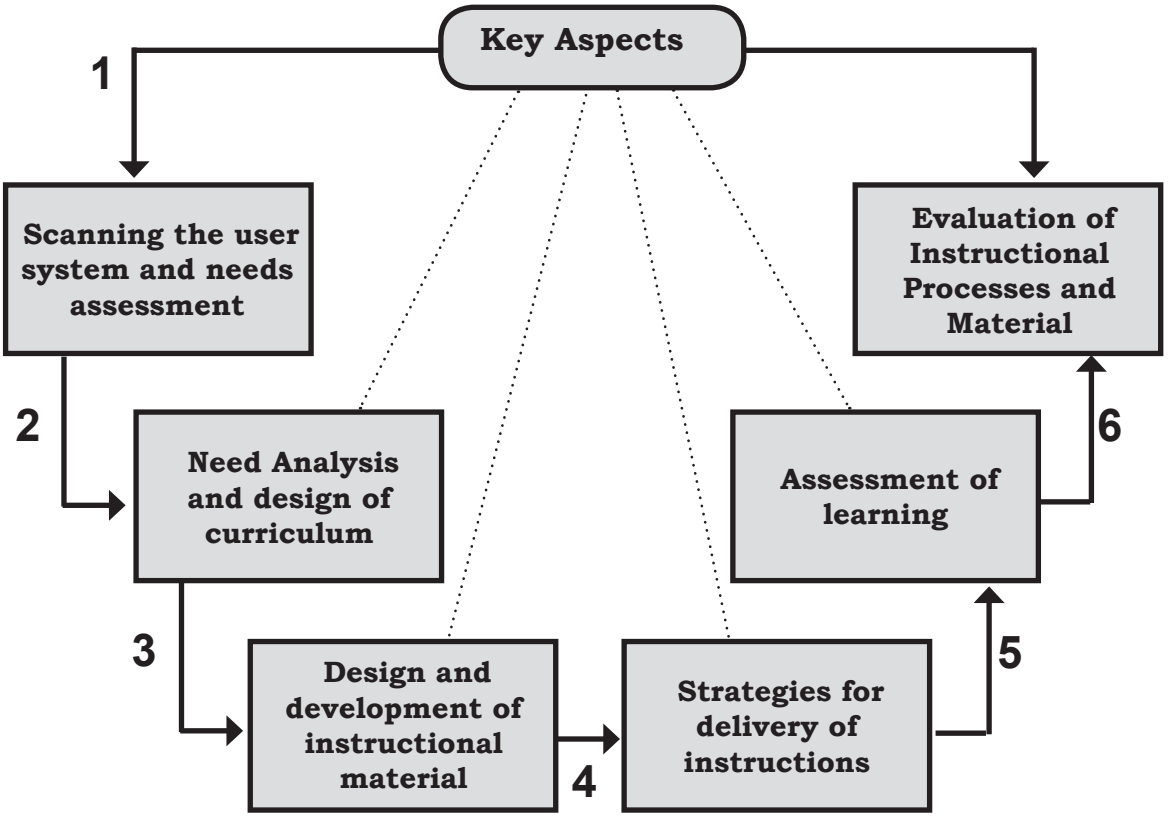
- Evaluation of instructional process and instructional materials

The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Aptech New Products Design Model



Onlinevarsity

your e-way to learning

TechnoWise

Are you a

TECHIE GEEK

looking for updates?

Logon to

www.onlinevarsity.com

Preface

Apache is one of the commonly used Web servers for hosting Web sites on the Internet. It is substantially faster, stable, and flexible than many other Web servers. It runs on Windows operating system, in addition to UNIX based operating systems such as Linux, Solaris, Digital UNIX, and AIX. Apache Web Server is the favorite server of Web administrators for its features such as support for the latest HTTP protocol, powerful file-based configuration, CGI, and virtual hosting.

In this book, we will learn the concept of a Web server and its significance in a network. We will learn to install Apache Web server. In addition, we will learn to configure and administer Apache Web server. The modules and directives form the core of Apache server. These modules and directives enable us to implement functions such as server security, secured communication, measure the server performance and implement virtual hosting on Apache server.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

Onlinevarsity

your e-way to learning

GROWTH

RESEARCH

OBSERVATION

UPDATES

PARTICIPATION

www.onlinevarsity.com

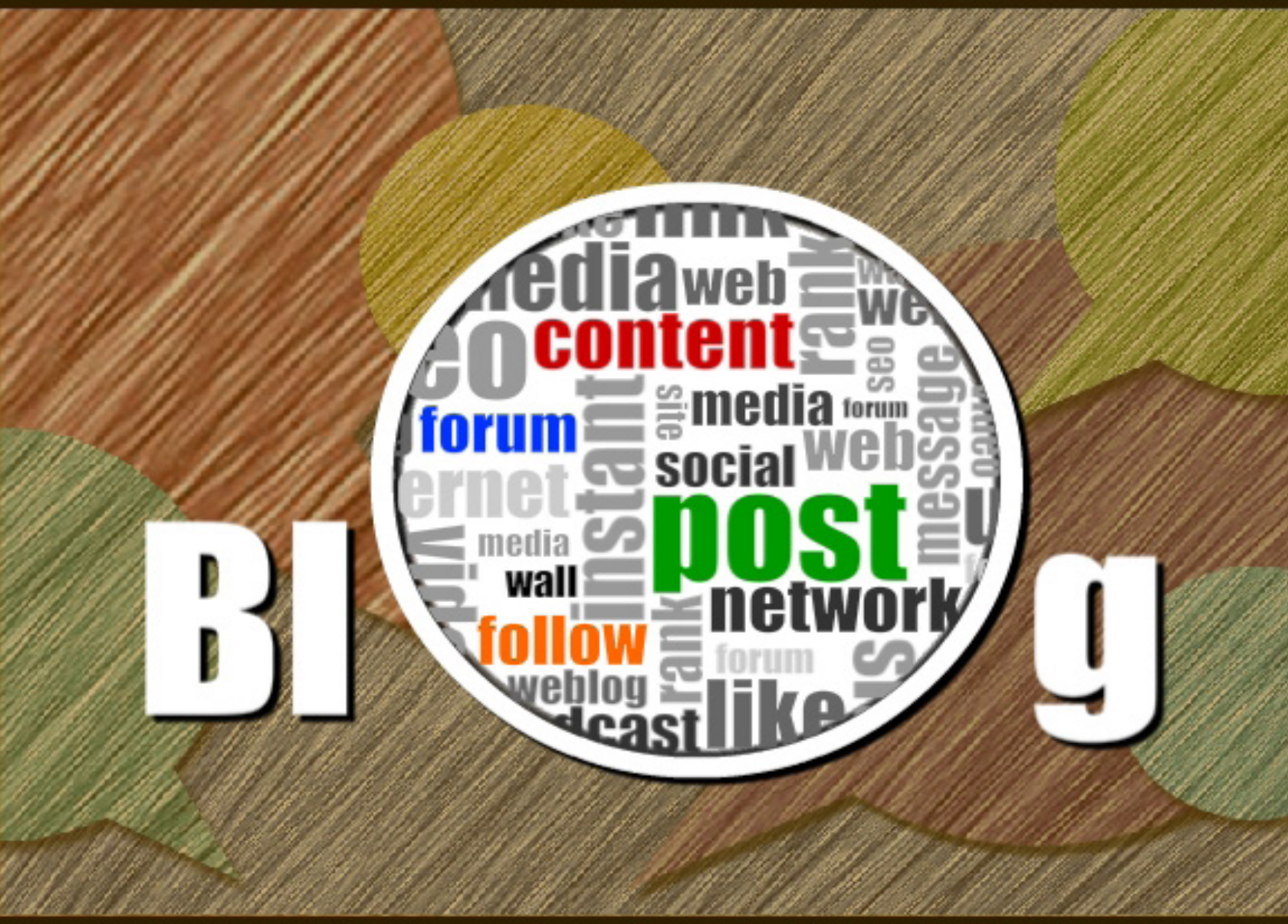
Aptech™
COMPUTER EDUCATION
Unleash your potential

Table of Contents

Sessions

Onlinevarsity

your e-way to learning



Balanced Learner-Oriented Guide

for enriched learning available @

www.onlinevarsity.com

Session

1 Introduction to Apache Web Server

Concepts

Objectives

At the end of this session, the student will be able to:

- *Describe Apache Web Server.*
- *Discuss the history of Apache Web Server.*
- *Explain the features of Apache Web Server.*
- *Explain the architecture of Apache Web Server.*

1.1 Introduction

Apache is the most commonly used Web server on the Internet. A wide range of Web sites from personal to corporate domains can be hosted using Apache. It is HTTP/1.1 compliant. Hypertext Transfer Protocol (HTTP) is a network protocol of the Web used for data communication. HTTP/1.1 allows multiple transactions, greater bandwidth, and serves multiple domains from a single Internet Protocol (IP) address. It enables to configure files and Common Gateway Interface (CGI) scripts which are returned by the server in response to errors and problems. It is compatible with most of the UNIX based operating systems, such as Linux, Solaris, Digital UNIX, and Advanced Interactive eXecutive (AIX).

In this session, you will learn about the Apache Web server, its history, and various features. In addition, you will also learn about the architecture of Apache Web server.

1.2 Introduction to Web Server

A Web server is a software application that uses HTTP protocol over the World Wide Web (WWW) to deliver the user the required resources. The primary function of a Web server is to serve resource request that are initiated by the clients or Web browsers. Using HTTP, the client initiates communication by requesting a resource to the server. The server processes the resource request and sends the requested file to the client. The server returns an error code and an error message when it is unable to complete the client resource request. The resource is a file located in the server's secondary memory. The Web server can contain pages, scripts, programs, and multimedia files.

The different types of Web servers are as follows:

- Apache HTTP Server
- Internet Information Server (IIS)

- Sun ONE
- National Center for Supercomputing Applications (NCSA) Server
- America Online's (AOL) Server

A brief description of the different servers is as follows:

- **Apache HTTP Server**

  Apache is a modular open source HTTP Web server. Apache supports the implementation of different features using the compiled modules. It includes different modules for server-side programming language support and authentication. Apache is a modular software as it allow users to install only those modules that are required for their Web sites. Language modules supported by the Apache Web server are Perl, Python, Tcl, and PHP scripts. Authentication modules supported by the Apache Web server include `.htaccess`, `mod_access`, `mod_auth`, `mod_digest`, and `mod_auth_digest`. In addition, it also contains a useful URL rewriter called `mod_rewrite` that consists of a rule-based rewriting engine to rewrite requested URLs on demand. By default, a basic set of modules is included in the core server. Also, Apache allows accessing the log files through a Web browser using free scripts, such as AWStats.

- **Internet Information Server (IIS)**

  IIS is Web server software from Microsoft Corporation. It runs only on Microsoft Windows operating system. This Web server can be used to execute Active Server Pages, FrontPage Extensions, and ASP.NET.

- **Sun ONE**

  Sun ONE was developed by Netscape and was first known as the Netscape Enterprise Web Server. It is a multi-platform Web server that is easy to setup and administer.

- **NCSA**

  NCSA is a compatible Web server for making hypertext and other documents available to the Web browsers.

- **AOL server**

  AOL server is America Online's Web server. It is a multithreaded Web server used for large scale dynamic Web sites.

1.3 History of Apache Web Server

Apache was developed at the National Center for Supercomputing Application (NCSA). It was designed and implemented by Rob McCool. However, Rob McCool was unable to continue working on the NCSA server. He quit the project and abandoned the server.

The NCSA widely used open source server because the source code was easily available. Developers worked on improvements and bug fixes and customized as per individual requirements. In 1995, Brian Behlendorf started to collect those patches and extensions. He formed a mailing list which was used for the purpose of exchanging information.

A group of eight people formed the mailing list community and released the first version of Apache. The name 'Apache' is derived from 'A PATCHy' server. It means that it is made up of 'patch files' and extensions to the NCSA server.

The first release version of Apache was version 0.6.2. The new server architecture, version 0.8.8 was designed and introduced by Robert Thau. On December 1st 1995, Apache version 1.0 was released. In the subsequent years, Apache received many new features and was ported to different operating systems.

The Apache Software Foundation group was founded in the year 1999. In March 2000, the ApacheCon, a conference for Apache developers, was held for the first time. On the ApacheCon conference in March 2000, Apache version 2.0 Alpha 1 was introduced. The version 2.0 of Apache was a complete redesign of the server architecture and was easier to port to different operating systems.

Version 2.2 of Apache was released in December 2005. This version of Apache was introduced with a new structure for authentication, improved caching modules, and support for proxy load balancing. Also, version 2.2 was introduced with an improved Web server capacity to handle files greater than 2 GB.

1.4 Features of Apache Web Server

Apache is a server that supports concurrency and serves different clients. Due to its modularity, many features can be incorporated as add-on modules and added to the server.

The features of Apache Web Server 2.2 are as follows:

- **Support for the latest HTTP 1.1 protocol** - Using HTTP 1.1, a Web browser sends parallel requests to the Web server. Apache is the first Web server compatible with HTTP 1.1 protocol and backward compatible with HTTP 1.0.

  For example, before the HTTP 1.1 protocol, a Web browser had to wait for a response from the Web server before issuing another request. With HTTP 1.1, Web browsers can simultaneously request for multiple files. This process of simultaneous request enables faster transfer of data.

- **Authentication/Authorization Modules** - Contains authentication and authorization modules with a new structured code in the access control, authentication, and authorization directory. Apache 2.2 supports digest authentication protocol. For example, the `mod _ auth` is separated into `mod _ auth _ basic` and `mod _ authn _ file`. The `mod _ auth _ basic` module is used to implement basic authentication features. The `mod _ authn _ file` contains passwords in plain text format.

- **Configuration Files** - Contains configuration snippets that are included in the configuration file to implement commonly used features.

- **Graceful stop** - Includes worker, prefork, and event processing modules to permit `httpd` to shutdown using the graceful stop signal. The prefork module implements multitasking processes and isolates requests to avoid conflicts. The `httpd` daemon is the service required to run Apache Web server.

- **Proxying** - Contains the `mod _ proxy` module to implement gateway capability for `FTP`, `HTTP/0.9`, `HTTP/1.1`, and `CONNNECT` (for ssl). The new `mod _ proxy _ ajp` module provides support for the Apache JServ protocol version 1.3 used by Apache Tomcat.

- **Smart Filtering** - Authenticates connections based on request and response header or an environment variable. Smart Filtering includes `mod _ filter` configured to the output filter chain. This feature enables filters to be conditionally inserted, based on any request or response header or environment variables and exempts the problematic dependencies and ordering problems in the 2.0 architecture.

- **Large File Support** - Supports request files larger than 2 GB on a 32-bit UNIX system.

- **SQL Database support** - Includes `mod _ dbd` and `apr _ dbd` framework that allows SQL support for the required modules in Apache. These modules not only provide database connections to Apache modules requiring SQL database functions but also manage these database connections.

- **File-based configuration** - Contains a single primary configuration file called `httpd.conf`. It also enables spreading multiple host configuration files in different files. This prevents the httpd. conf file from managing different virtual server configurations.

- **Support for CGI** - Contains `mod _ cgi` and `mod _ cgid` modules to support CGI.

- **Support for Fast CGI** - Contains the `mod _ fcgi` module to implement the FastCGI environment within Apache. FastCGI is a fast, open, and secure Web server interface. The performance problems inherent in CGI are eliminated in FastCGI.

- **Support for virtual hosts** - Is compatible with IP-based addresses and named virtual hosts.

- **Support for HTTP authentication** - Supports Web based basic authentication using standard password files, DBMs, SQL calls, and call to external authentication programs. Apache also supports message-digest-based authentication.

- **Integrated Perl** - Includes Perl, the standard program for CGI scripting. The `mod _ Perl` module is used to load Perl based CGI programs in Apache. This process removes the start-up penalties that are associated with an interpreted language like Perl.

- **Support for PHP scripting** - Supports PHP scripting using the `mod _ php` module.

- **Java Servlet support** - Provides the Tomcat environment to execute Java Servlets and Java Server Pages (JSP).

- **Server status and customizable logs** - Supports customizing log files and monitoring the server status using the Web browser.

- **Support for Server-Side Includes (SSI)** - Provides set of SSI for Web site developers.

- **Support for Secured Socket Layer (SSL)** - Provides the OpenSSL facility and the `mod _ ssl` module to a Web site developer for creating a SSL Web site using Apache.

The Apache server also includes many features such as, directory indexing, directory aliasing, content negotiations, configurable HTTP error reporting, and SetUID execution of CGI programs. Resource management for child processes, server-side image maps, URL rewriting, URL spell checking, and online manuals are the other features that are also available.

1.5 Architecture of Apache Web Server

Apache 2.2 Web server has a conceptual architecture which uses modular approach. The architecture comprises two components, the core and the Multi-Processing modules. The core component consists of the basic features and the Multi-Processing modules contain the additional features that can be included according to the requirements.

For example, the `mod_cgi` and `mod_cgid` modules can be implemented to support execution of CGI scripts.

Concepts

Figure 1.1 displays the architecture of Apache Web Server.

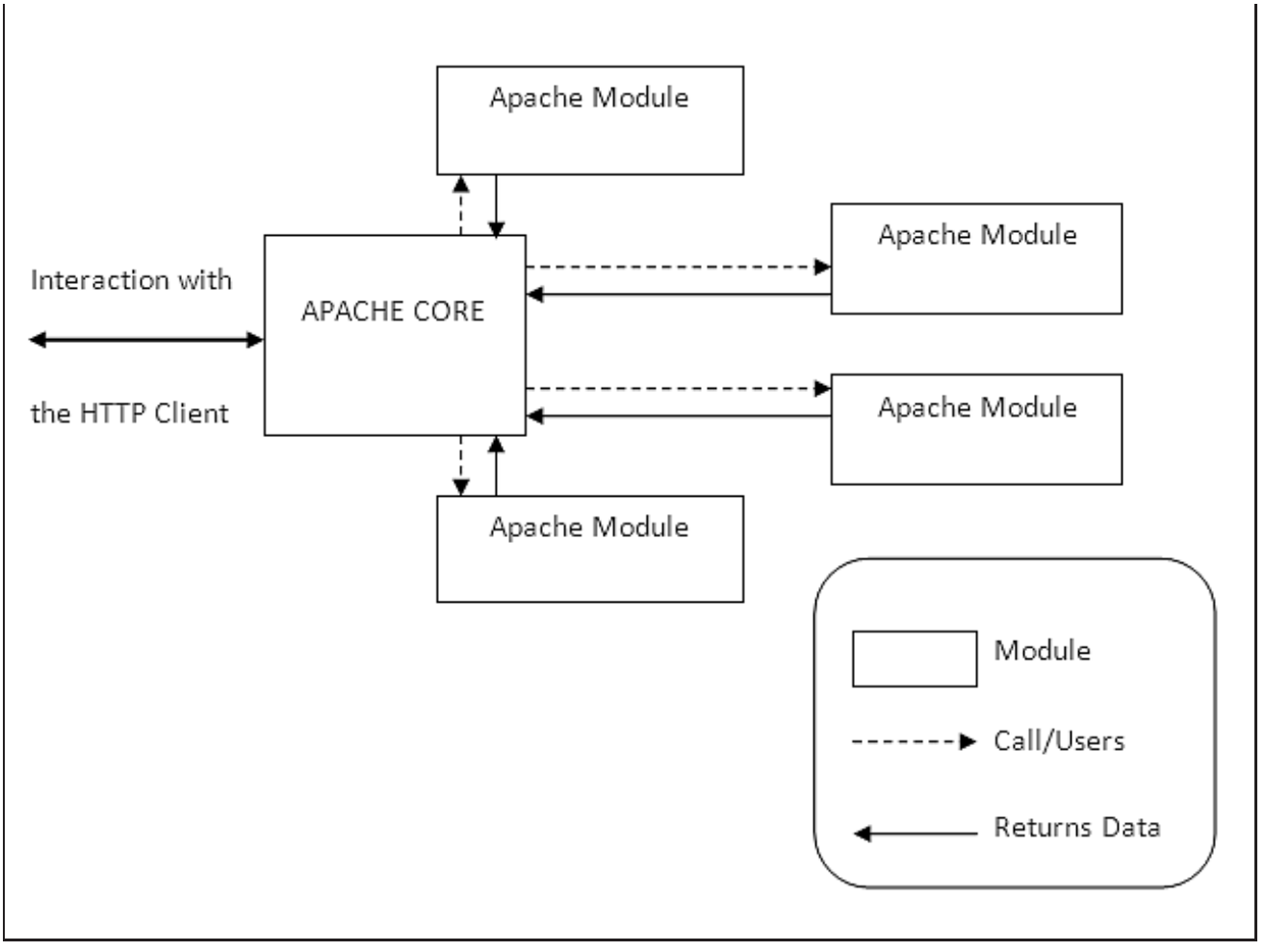


Figure 1.1: Architecture of Apache Web Server

According to figure 1.1, the core contains the basic functionality of the server. It also contains the implementation details of a number of utility functions. The different components of the core are as follows:

`http_protocol.c` - contains functions that communicate with the client and transfers data to the client using this component

`http_main.c` - contains the functionality for starting up the server and waits for and accepts connections

`http_request.c` - contains the logic for handling request processing, dispatching control to the modules, and for error handling

`http_core.c` - contains the logic for implementing the basic functionality

`alloc.c` - contains the functionality for allocating resource pools and keeping track of resources

`http_config.c` - contains the logic for reading and managing the information present in the configuration files

- **The Core Module**

  The default installation directory contains all the files required for the Apache core. The core module contains maximum possible functionality implemented as separate modules. The core modules are not required to be modified while adding new features or modules.

- **Multi-Processing Modules**

  Multi-processing modules (MPMs) are used to isolate tasks in a program. In Apache 1.3, the parent process was split into a set of child processes that served the actual request. The parent processes monitors the child processes and spawns or kills child processes depending upon the request received. The non process-centric platforms such as Microsoft Windows had trouble running this module; therefore the Apache group implemented the MPM based solution. MPMs enable features to be implemented as per requirements.

  In the MPM module, each MPM starts with the server process and services request through child processes or threads depending upon which MPM is implemented. A thread is a single sequential flow of control within any program. Table 1.1 lists the MPM present in the Apache Web server.

MPM	Description
Prefork	Creates a pool of child processes to handle the requests. Each child process contains a single thread. For example, if the server starts 40 child processes then it handles 40 requests simultaneously. In case of an error, if a child process ends then only a single request is lost. The maximum and minimum setting controls the number of child processes. If the number of requests increases then the new child processes are created until the maximum number is reached. In a similar manner, any extra child processes are killed when the requests fails or is completed.

MPM	Description
Worker	Enables thread support. This module also has child processes similar to the prefork MPM and enables a predefined number of threads under each child process. Each thread within a child process serves a different request. For example, if Apache is running 30 child processes and each child is allowed to have 10 threads each, the total number of threads that Apache server handles simultaneously is 30 x 10 = 300 requests. Apache adds or deletes the process by monitoring the spare - thread count. In this process maximum number of idle threads is killed. All the processes run under the same user and group ID assigned to Apache server.
Filtering I/O	Architecture for layered I/O is a major characteristic of the Apache 2.2 server. The characteristic being, the output of one module is input to another module. The CGI script produces output as SSL tags which are then processed before the final output is sent to the Web browser.
CGI Daemon	The CGI program runs in the background and defines how Web servers delegate the generation of Web pages to CGI scripts.

Table 1.1: Multi-Processing Modules in Apache

The CGI Daemon is required to execute Perl and PHP scripts on the Web browser. The CGI Daemon uses the `mod_cgid` module to interact with threads.

Figure 1.2 displays the execution of CGI request for a script called `myscript.pl`.

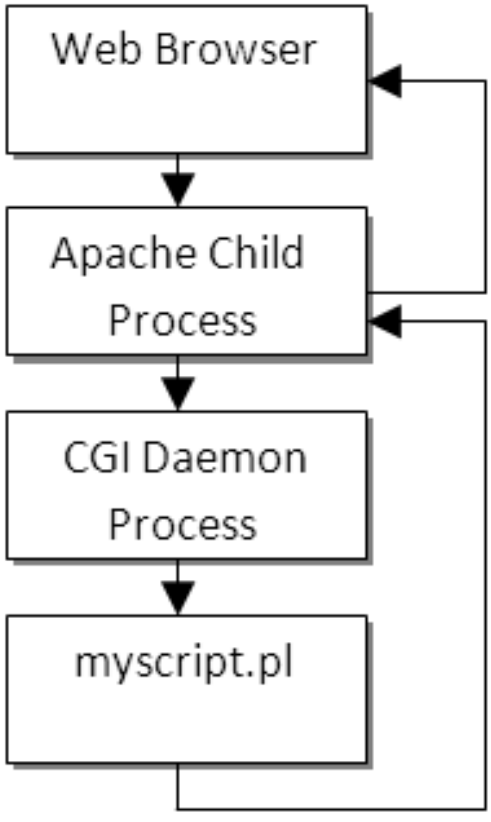


Figure 1.2: Working of CGI Daemon

The CGI Daemon works as follows:

1. The Web browser creates the child process.

2. When a CGI reads a thread within a child process, it passes the request to the CGI daemon.

3. The CGI daemon spawns the CGI script and the CGI script generates data to the thread in the child process.

4. The thread returns the data to the Web browser.

Summary

- A Web server is a software application that uses the Hypertext Transfer Protocol (HTTP).
- Apache was designed and implemented by Rob McCool. It was developed at the National Center for Supercomputing Application (NCSA).
- Apache is the first Web server to be compatible with HTTP 1.1 protocol and backward compliant with HTTP 1.0.
- Apache originated as a UNIX based Web server and contains a single primary configuration file called `httpd.conf`.
- Each MPM starts with the server process and services requests through child processes or threads depending upon which MPM is implemented.
- The prefork MPM creates a pool of child processes to handle the requests. Each child process has a single thread.
- The threaded MPM process enables a predefined number of threads. Each thread within a child process serves a different request.

Check Your Progress

1. Which of the following protocol does a Web server application use?

   a. Hyper Text

   b. Hypertext Transform

   c. Hypertext Transfer

   d. Hypertext Markup

2. _____ designed and implemented Apache Web server.

   a. Brian Behlendorf

   b. Rob McCool

   c. Robert Thau

   d. Ken Coar

3. Which of the following options represents the features of Apache Web server?

   a. Integrated Proxy server

   b. CGI Daemon

   c. Powerful file-based configuration

   d. Modularity

4. Which of the following modules enables a predefined number of threads in a child process?

   a. The perchild MPM

   b. The threaded MPM

   c. The prefork MPM

   d. The Worker MPM

Check Your Progress

5. In _____ a thread receives a request to execute a CGI script.

   a. Filtering I/O

   b. The prefork MPM

   c. CGI processes

   d. CGI Daemon

Session

2 Installing Apache Web Server 2.2

Concepts

Objectives

At the end of this session, the student will be able to:

- *Identify the system requirements for installing Apache Web Server.*
- *Explain the installation process of Apache Web Server.*
- *Evaluate the installation of Apache Web Server.*
- *Explain the process of editing the configuration file, httpd.conf.*

2.1 Introduction

Apache is an open source Web server software that can be downloaded from the official Apache Web site. It is a free software. The installation of Apache consists of unpacking the source code, configuring, compiling, and installing. The main configuration file of Apache `httpd.conf` contains modules and directives.

In this session, you will identify the system requirements for installing Apache Web server. You will also learn to install and test the Apache Web server. In addition, you will learn to edit the `httpd.conf` file.

2.2 System Requirements for Installing Apache Web Server

The minimum system requirements for installing the Apache Web server are as follows:

- **Hard Drive Space** - Requires 50 MB free space on the hard drive.

Note: The actual disk space varies depending upon the selection of the configuration options and third party modules.

- **Compiler** - Requires the installation of ANSI-C compiler prior to Apache 2.2.17 installation.
- **Interpreter** - Requires a Perl 5 interpreter for supporting scripts written in Perl such as, `apxs` or `dbmmanage`. The `apxs` tool is used to install extension modules in Apache. The extension modules can be developed by third-parties to enable implementation of different features in Apache. The `dbmmanage` tool manages user authentication files in Database Management (DBM) format that store usernames and passwords. This requirement is optional.

Note: The Perl 5 interpreter is an optional requirement for the installation of Apache Web server.

Concepts

2.3 Installing Apache Web Server

The installation of Apache Web server must be executed on a Linux platform with the root account privileges.

To install Apache 2.2.17, perform the following steps:

1. Login to Linux as `root` in the GNOME environment.

2. Download the **httpd-2.2.17.tar.gz** file from `http://httpd.apache.org/download.cgi`.

3. Right-click the **httpd-2.2.17.tar.gz** file and select **Extract Here**. The package contents are extracted to the folder named **httpd-2.2.17** in the current directory.

4. Right-click the **httpd-2.2.17** folder and select **Open in Terminal**.

   Figure 2.1 displays the Terminal window.

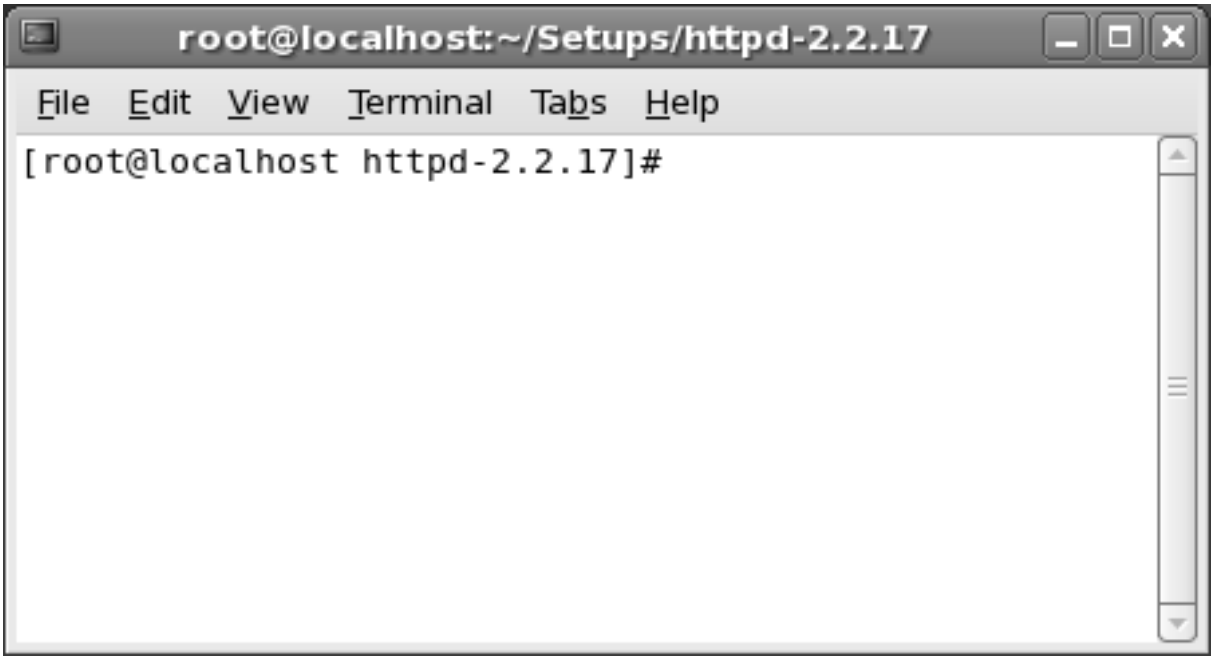


Figure 2.1: Terminal Window Displaying the httpd-2.2.17 Folder

5. To configure Apache with the default options, enter the following command at the command prompt:

```
./configure --prefix=/usr/local/apache2
```

Note: The `--prefix` option in the `configure` command specifies the location where the Apache Web server will be installed.

The `configure` script requires time to build the `Makefiles` and compile the source code. Figure 2.2 displays the output of the `configure` command.

Figure 2.2: configure Command

6. To compile the Apache source code, enter the following command at the command prompt:

```
make
```

Figure 2.3 displays the output of the `make` command.

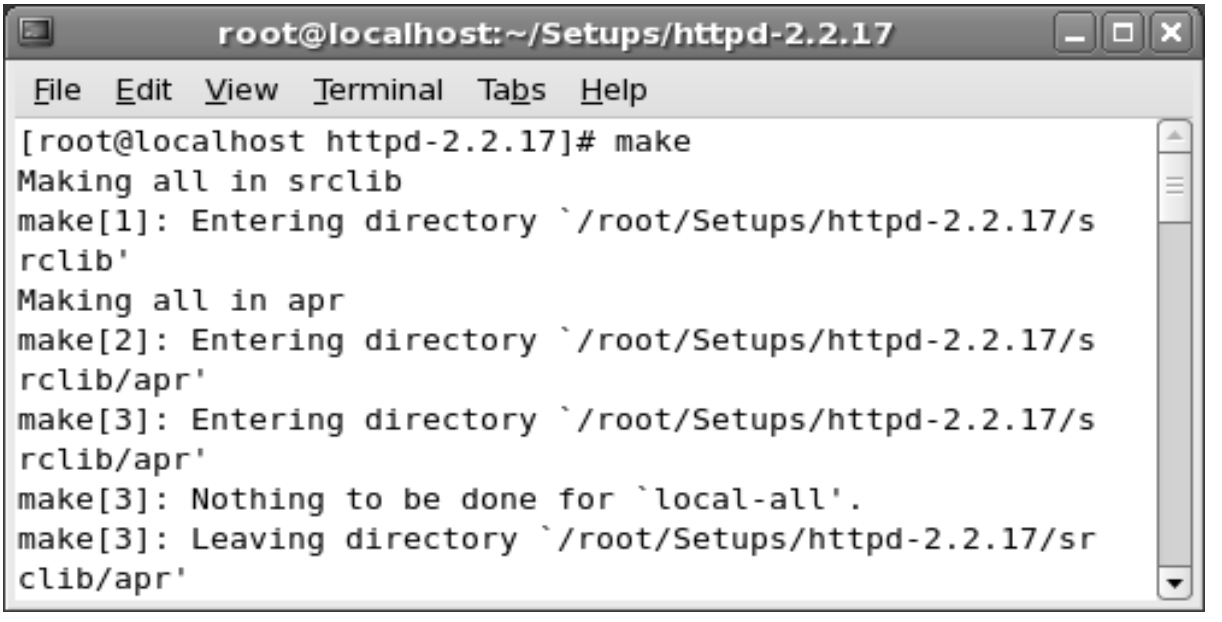


Figure 2.3: make Command

Note: All the executable files are generated when you compile the source code.

7. To install the resultant binary files and documentation to their respective directories, enter the following command at the command prompt:

```
make install
```

Figure 2.4 displays the output of the `make install` command.

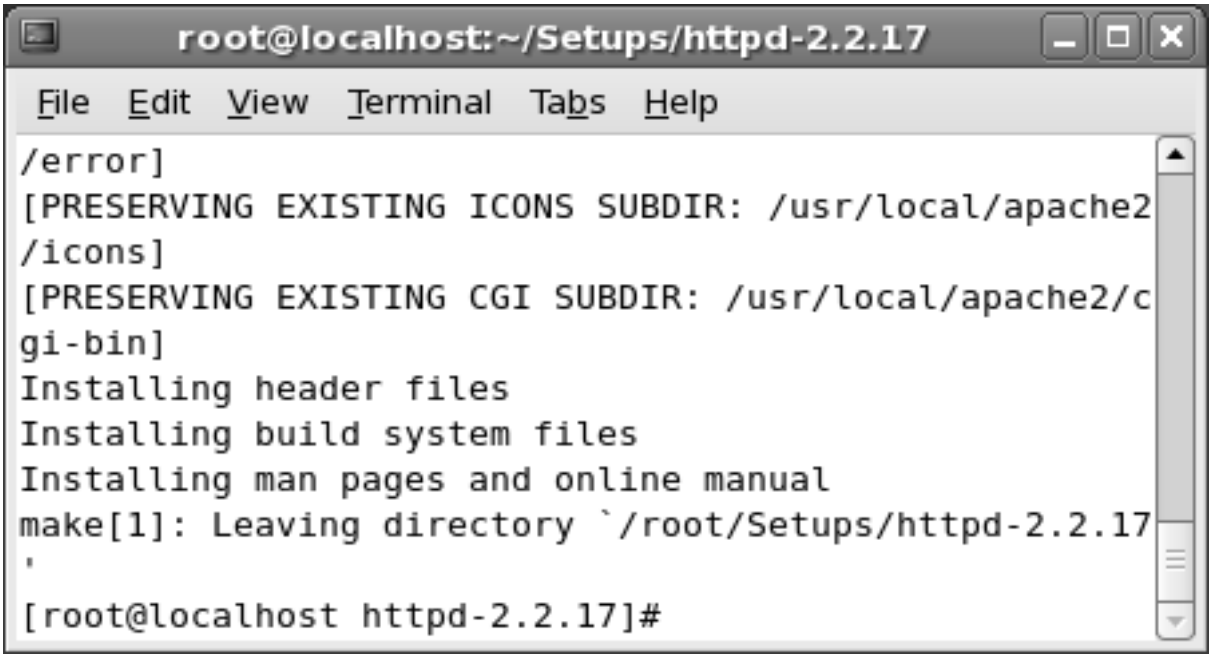


Figure 2.4: make install Command

Note: The `make install` command compiles the source files as per the `Makefile` generated from the configure script. This completes the installation procedure for the Apache Web server.

2.4 Starting Apache Web Server

After installation, the Apache Web server must be started. The scripts can be executed on the Web browser only after starting Apache Web server.

To start Apache Web server, perform the following steps:

1. To start the Web server, enter the following command at the command prompt:

```
/usr/local/apache2/bin/apachectl start
```

Figure 2.5 displays the output of the command.

Figure 2.5: Starting Apache Server

2. Open the Mozilla Web browser and enter `http://localhost/` in the address bar and press **Enter**.

Figure 2.6 displays the localhost page.

Figure 2.6: Localhost Page

3. To stop the server, enter the following command at the command prompt:

```
/usr/local/apache2/bin/apachectl stop
```

Figure 2.7 displays the output of the command.

Figure 2.7: Stopping Apache Web Server

2.5 Editing the httpd.conf File

The `httpd` is the main executable file used for reading the configuration settings provided in the `httpd.conf` file of Apache. It runs as a standalone daemon process in the background and performs a specified operation, such as processing requests at predefined times.

The Apache server launches several child processes at the start that listen and answer to the requests from clients. The `apachectl` control script invokes the `httpd` program. When the `httpd` process is invoked, it locates and reads the configuration file, `httpd.conf` and executes its contents.

The `httpd.conf` is the main configuration file for Apache. The installation of Apache creates this file in the `/usr/local/apache2/conf/` directory. The `httpd.conf` file consists of directives and modules. Directives are configuration commands that control the performance of Apache Web server and are specified one per line. The modules contain the Apache functionality.

For the changes to take effect, you must restart the Apache Web server after editing the contents of the `httpd.conf` file. During startup, Apache identifies the changes made to the main configuration file.

In the `httpd.conf` file, the '`#`' (hash) symbol is used as a comment. The '`\`' (back-slash) symbol is used as the last character on a line to indicate that the directive continues from the next line.

Note: There must be no characters or white space between the '\' (back-slash) and the end of the line.

To open the `httpd.conf` file, perform the following steps:

1. Login to Linux as `root` in the GNOME environment.

2. To start a new terminal, select **Applications** → **Accessories** → **Terminal**. The Terminal window is displayed.

   Figure 2.8 displays the Terminal window.

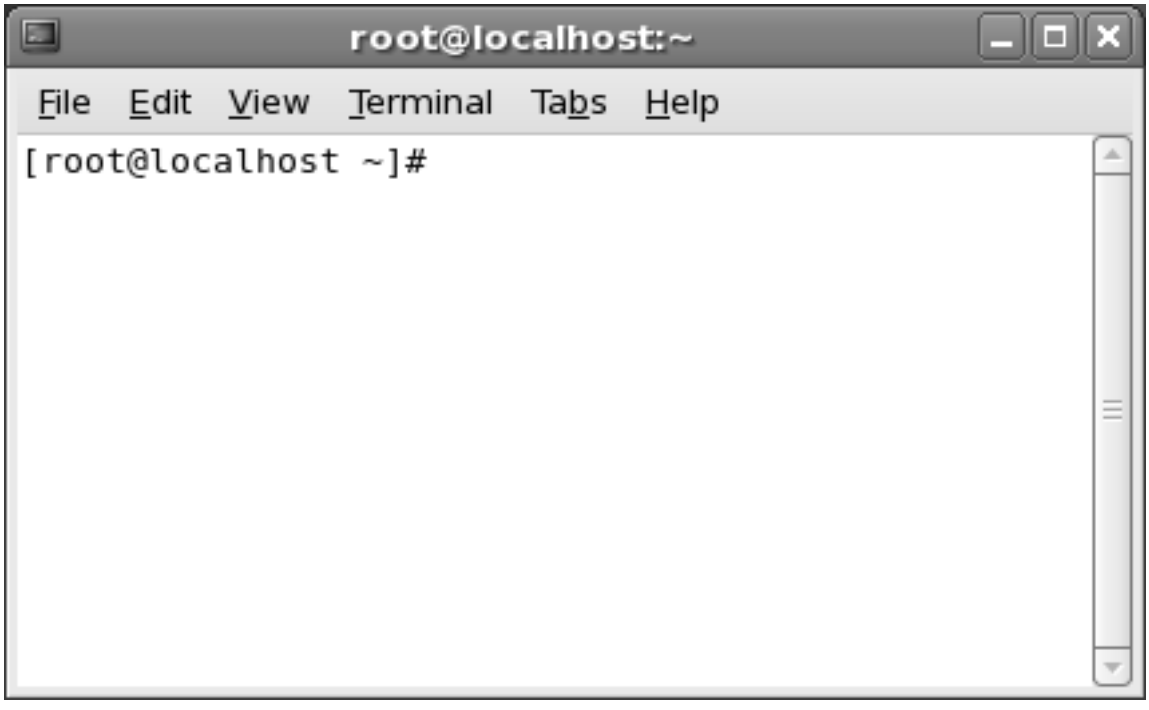


Figure 2.8: Terminal Window

3. To browse to the `conf` directory, enter the following command at the command prompt:

```
cd /usr/local/apache2/conf
```

4. To open the `httpd.conf` file in the `vi` editor, enter the following command at the command prompt:

```
vi httpd.conf
```

Figure 2.9 displays the `httpd.conf` file.

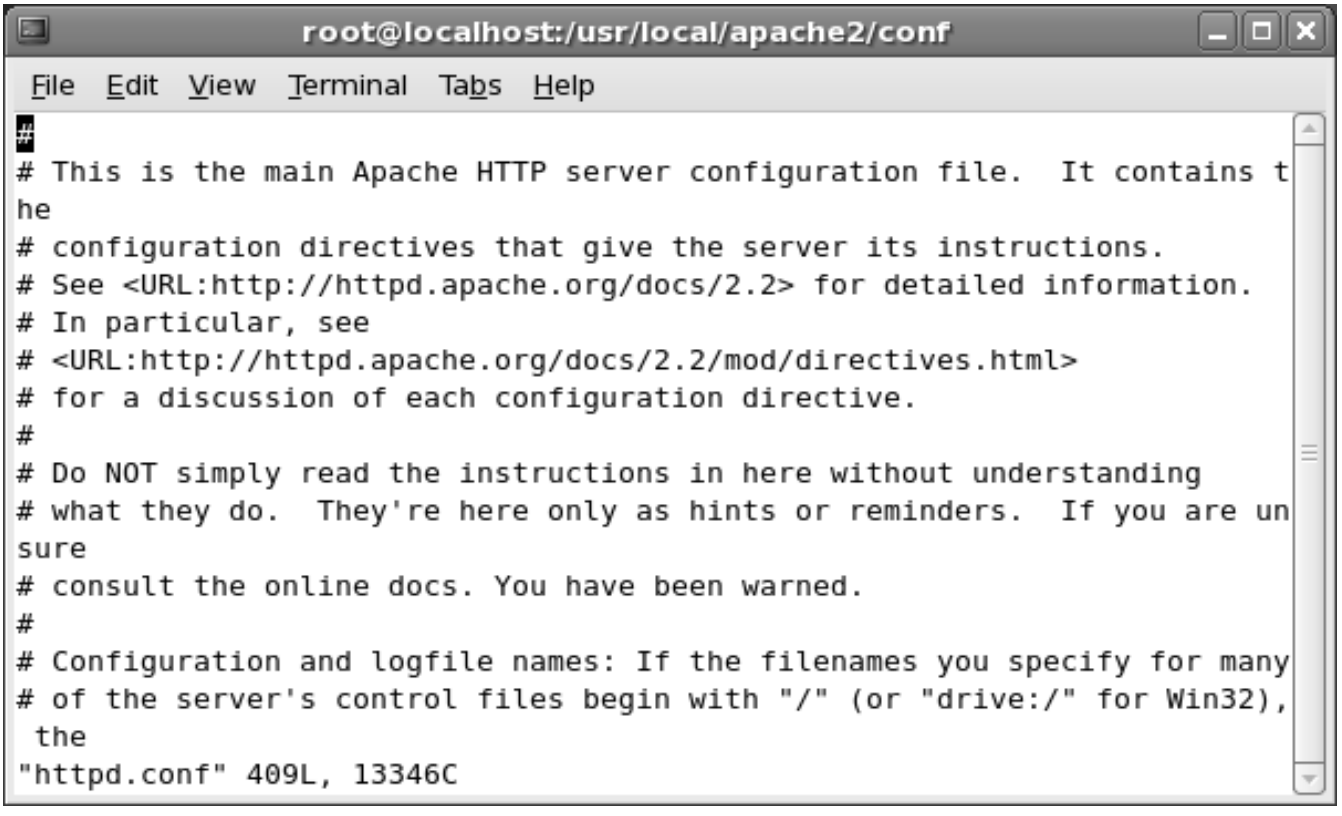


Figure 2.9: httpd.conf File

The `httpd.conf` file consists of directives to configure the Apache server at startup.

Table 2.1 describes some of the directives.

Directive Name	Description
`ServerName`	Specifies the hostname and the port for the server.
`Listen`	Instructs the server to listen to the specific IP addresses and ports.
`ServerRoot`	Specifies the directory path where the server is installed. The default option is `/usr/local/apache2`.
`DocumentRoot`	Specifies the directory path from where the server processes the files. The default option is `/usr/local/apache2/htdocs`.
`LanguagePriority`	Sets the preferred language.
`DefaultLanguage`	Defines the default language option.

Table 2.1: Directives in httpd.conf

Note: The values for the directives can be modified as per requirements.

5. To save and exit the `vi` editor, enter the following command at the command prompt:

```
:wq
```

Concepts

Summary

- The procedure to install Apache includes unpacking the source code, configuring the Apache source tree, compiling the source code to build executables, and installing Apache using the `configure`, `make`, and `make install` command.
- `httpd.conf` is Apache's main configuration file that contains modules and directives.
- The `./configure` command enables to customize the installation of the Apache Web server.
- The `make` command compiles the binary files to generate the executables required to install Apache.
- The `make install` command copies the resultant binary files and documentation to their respective directories during installation.
- The Apache server performs preliminary activities at startup, such as opening the log files, launching child processes that listen and respond to the requests from clients.
- The `apachectl` control script invokes the `httpd` program.
- You can configure the Apache Web server by using the `httpd.conf` file that contains modules and directives.
- Apache Web server must be restarted for any changes made to the configuration file, `httpd.conf` to take effect.

Check Your Progress

1. Which of the following option represents the default syntax to configure the Apache source code?

   a. `.//configure`

   b. `configure./`

   c. `./configure`

   d. `configure.//`

2. Which of the following command compiles the Apache source code?

   a. `make install`

   b. `install make`

   c. `make`

   d. `install`

3. The _____ command installs Apache and the required documentation files.

   a. `make install`

   b. `install`

   c. `make`

   d. `./configure`

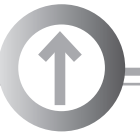
4. Which of the following control script is used to invoke the `httpd` program?

   a. `apachectl`

   b. `configure`

   c. `start`

   d. `apachect11`



Check Your Progress

5. Which of the following file is used to configure Apache?

   a. `httpd.conf`

   b. `http.conf`

   c. `config`

   d. `config.httpd`

6. Apache requires the _____ compiler to be installed prior to its installation.

   a. Borland C

   b. ANSI-C

   c. Turbo C

   d. Turbo C++

Session

3 Installing Apache Web Server 2.2 (Lab)

Objectives

At the end of this session, the student will be able to:

- *Install Apache Web Server.*
- *Start and stop Apache Web Server.*
- *Edit the httpd.conf file.*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Installing Apache Web Server

The installation of Apache on Linux must be done using the `root` account privileges.

To install Apache Web server:

1. **Login to Linux as `root` in the GNOME environment.**
2. **Download httpd-2.2.17.tar.gz. file from `http://archive.apache.org/dist/httpd/`**
3. **Right-click the downloaded httpd-2.2.17.tar.gz file and select Extract Here. The package contents are extracted to the folder named httpd-2.2.17 in the current directory.**
4. **Right-click the httpd-2.2.17 folder and select Open in Terminal.**

Figure 3.1 displays the Terminal window.

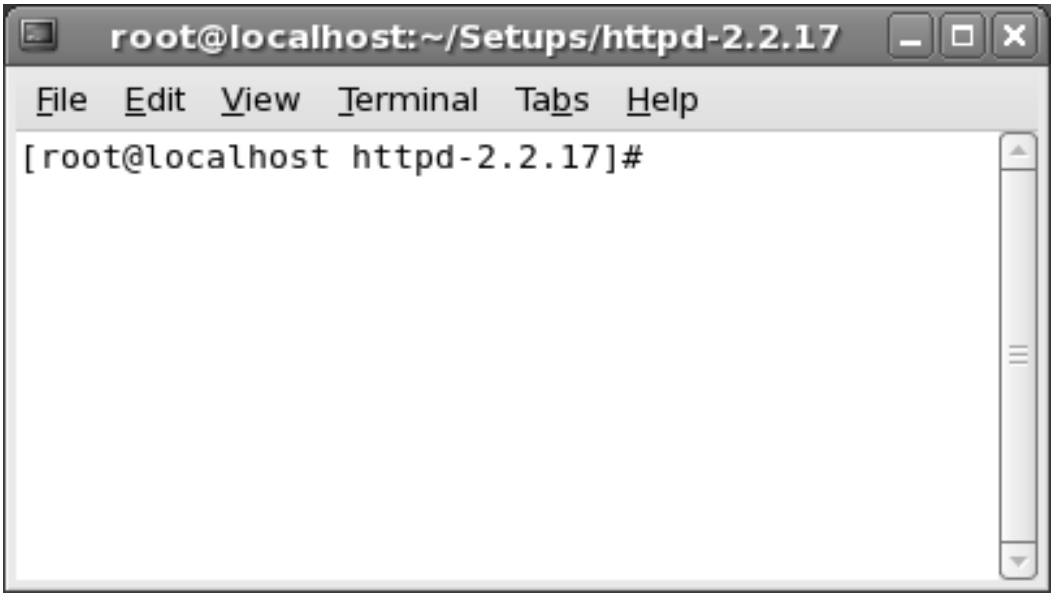


Figure 3.1: Terminal Window

5. **To configure Apache, and specify the location where Apache will be installed, enter the following command at the command prompt:**

```
./configure --prefix=/usr/local/apache2
```

Figure 3.2 displays the output of the `configure` command.

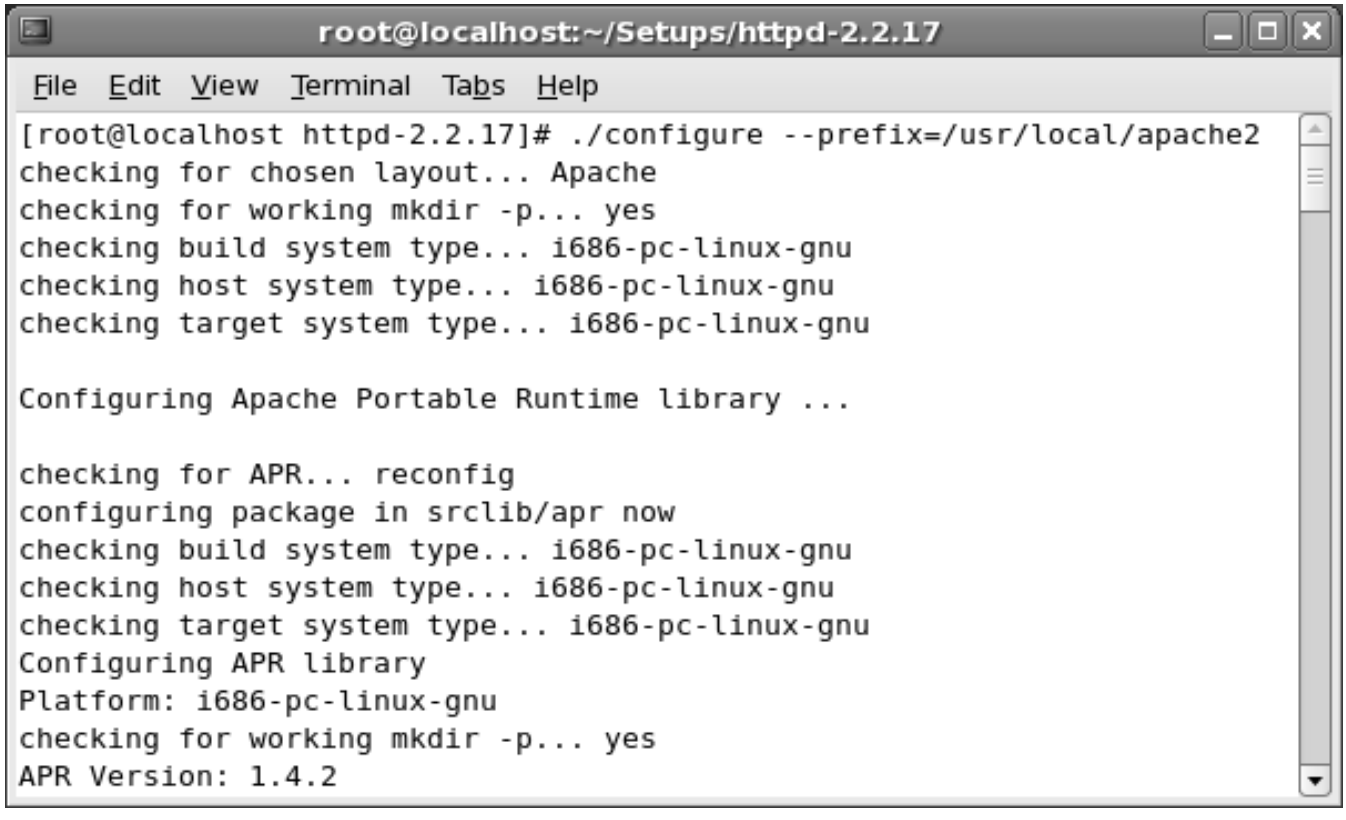


Figure 3.2: configure Command

6. **To compile the Apache source code, enter the following command at the command prompt:**

```
make
```

Figure 3.3 displays the output of the `make` command.

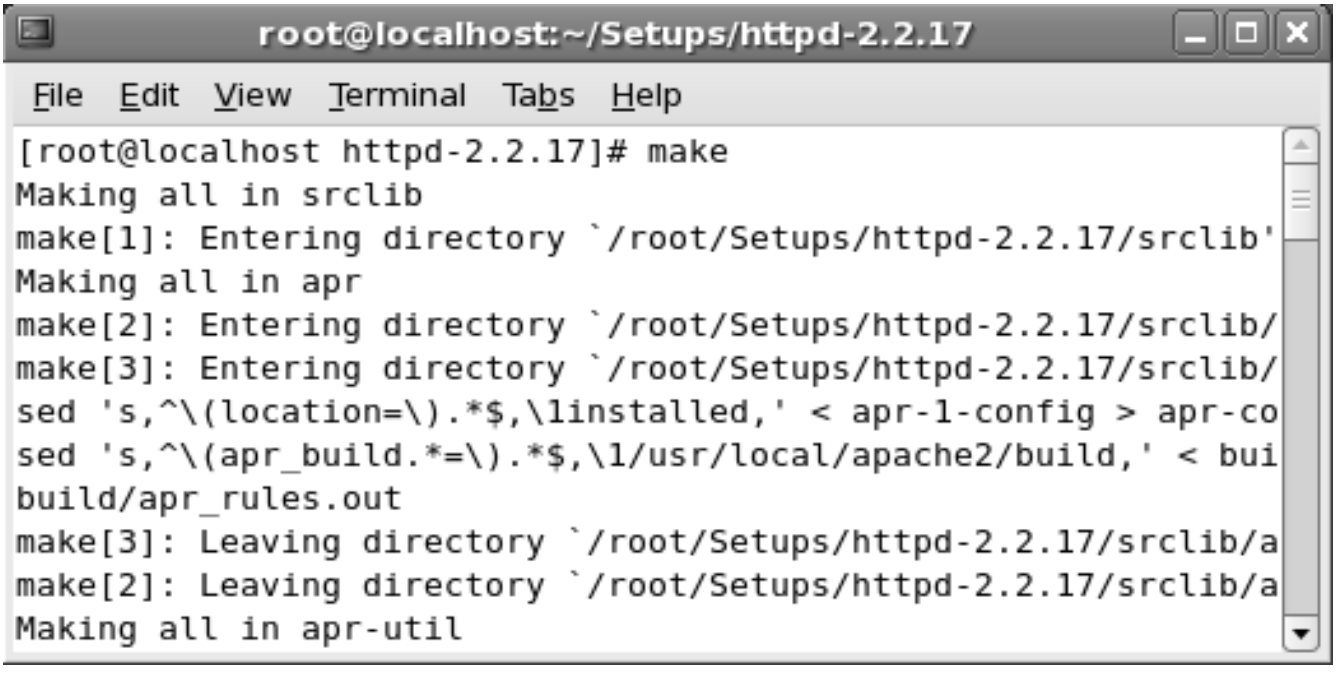


Figure 3.3: make Command

Note: All the executable files are generated when you compile the source code.

7. **To install the resultant binary files and documentation to their respective directories, enter the following command at the command prompt:**

```
make install
```

Figure 3.4 displays the output of the `make install` command.

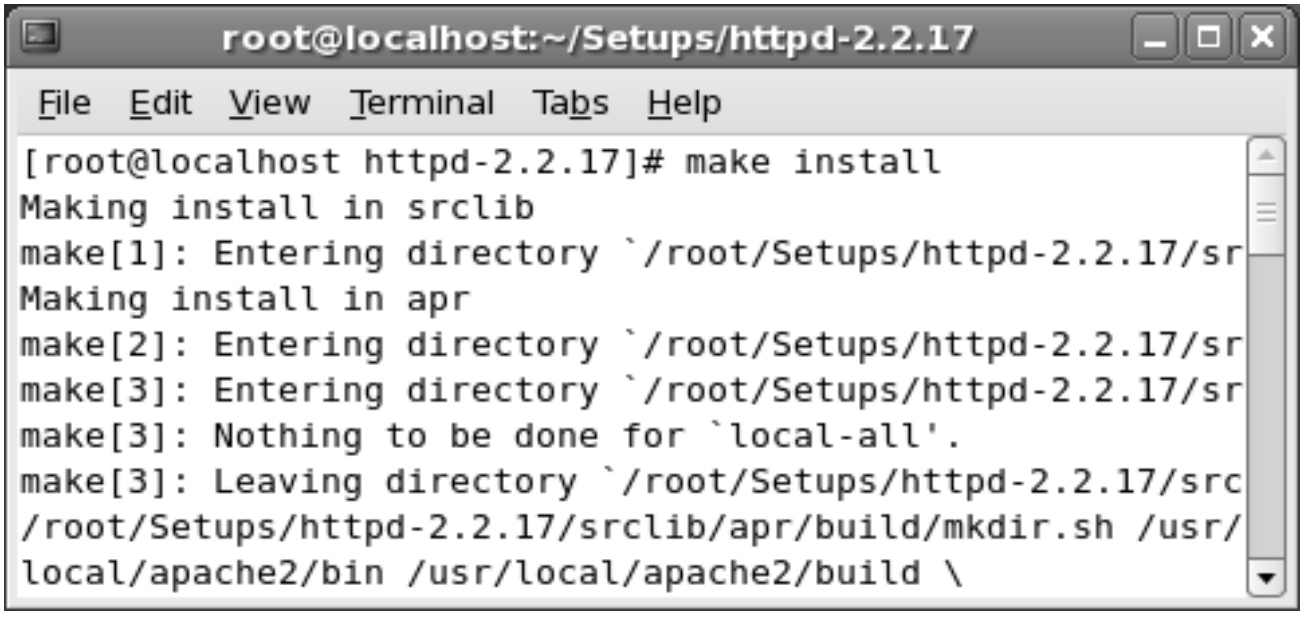


Figure 3.4: make install Command

The `make install` command copies the compiled binary files and the executables to their respective locations. This completes the installation procedure for Apache Web server.

Starting and Stopping Apache Web Server

After the installation of Apache Web server is complete, you must test the installation. You need to check whether the server starts and stops without any errors.

To start and stop Apache Web server:

1. **Open the Linux terminal.**
2. **To start Apache Web server, enter the following command at the command prompt:**

   ```
   /usr/local/apache2/bin/apachectl start
   ```

Figure 3.5 displays the output of the command.

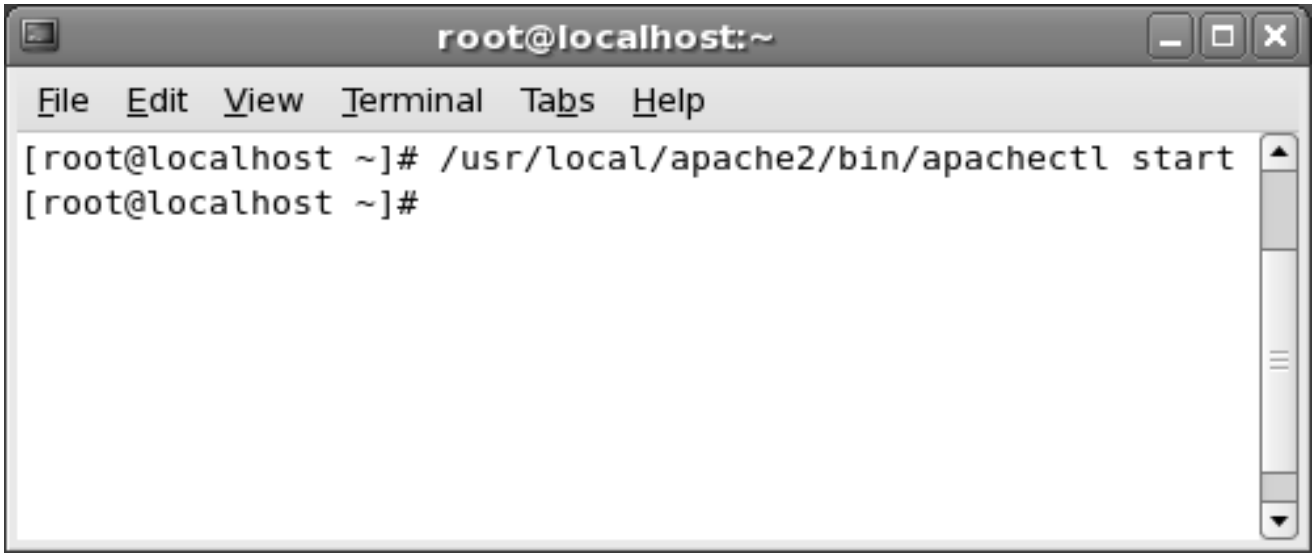


Figure 3.5: Starting Apache Web Server

3. **Open the Mozilla Firefox Web browser.**

4. **Enter `http://localhost/` in the address bar and press Enter.**

Figure 3.6 displays the localhost page.

Figure 3.6: Localhost Page

5. **To stop Apache Web server, enter the following command at the command prompt:**

```
/usr/local/apache2/bin/apachectl stop
```

Figure 3.7 displays the output of the command.

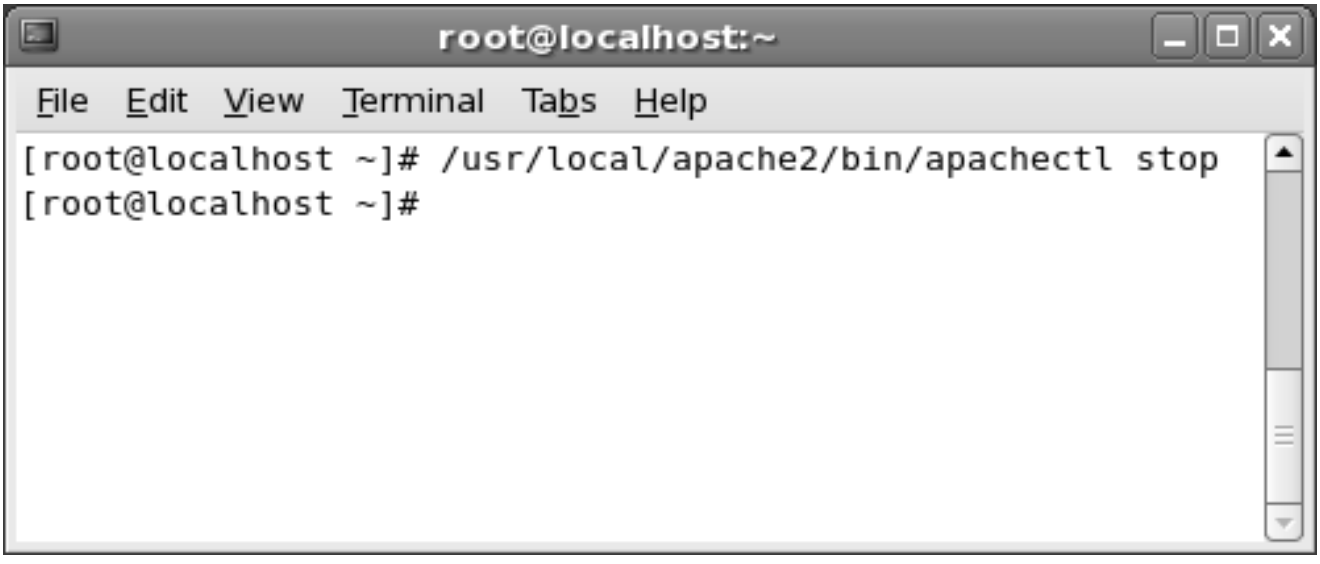


Figure 3.7: Stopping Apache Web Server

Note: An attempt to view the localhost page on the Web browser after stopping Apache Web server will display an error.

Editing the httpd.conf file

The `httpd.conf` is the main configuration file for Apache. To configure Apache as required, you need to edit this file.

To edit the `httpd.conf` file, perform the following steps:

1. **Login to Linux as `root` in the GNOME environment.**
2. **Open the Linux Terminal.**
3. **To browse to the `conf` directory, enter the following command at the command prompt:**

   ```
   cd /usr/local/apache2/conf
   ```

Figure 3.8 displays the output of the command.

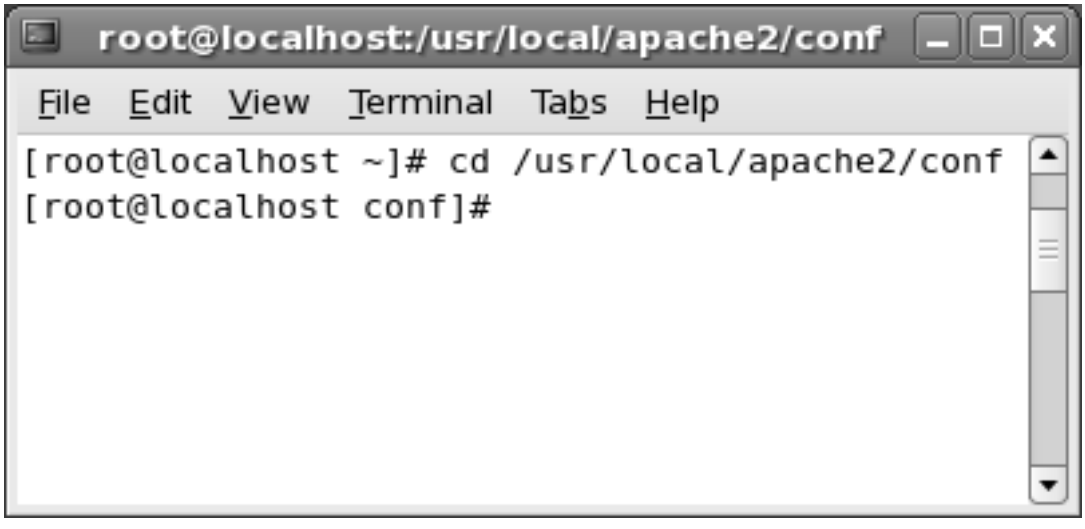


Figure 3.8: Browse to conf Directory

4. **To open the `httpd.conf` file in the `vi` editor, enter the following command at the command prompt:**

```
vi httpd.conf
```

Figure 3.9 displays the output of the command.

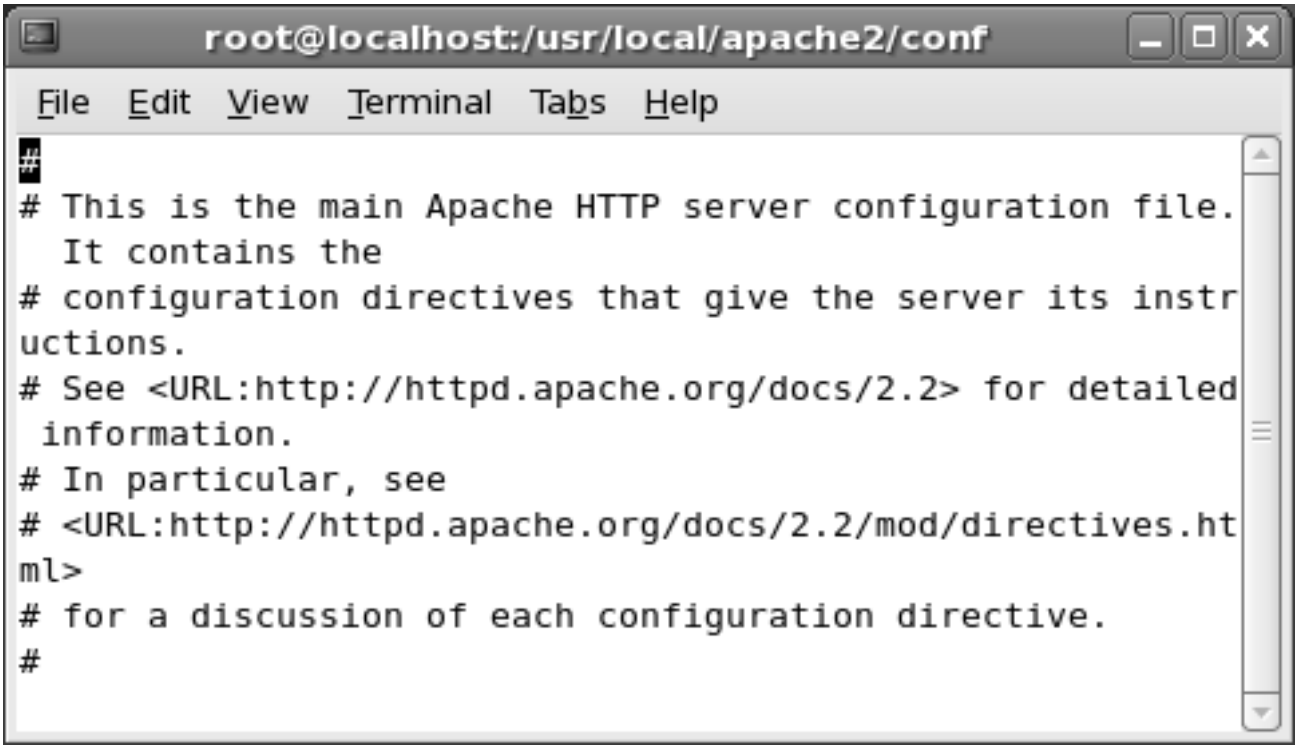


Figure 3.9: httpd.conf File

5. **To save and exit the `httpd.conf` file in the `vi` editor, enter the following command at the command prompt:**

   ```
   :wq
   ```

Do It Yourself

1. Start the Apache Web Server.
2. Stop the Apache Web Server.
3. View the `httpd.conf` file.

Onlinevarsity
your e-way to learning

WRITE-UPS BY

EXPERTS AND **LEARNERS**

TO PROMOTE NEW AVENUES AND
ENHANCE THE LEARNING EXPERIENCE

FOR FURTHER READING, LOGIN TO

www.onlinevarsity.com

Session

4 Configuring Apache Web Server

Concepts

Objectives

At the end of this session, the student will be able to:

- *Describe the process of configuring Apache Web Server.*
- *Explain the use of Directives.*
- *Explain the use of Modules.*
- *Explain the process of modifying the default Index page.*

4.1 Introduction

Apache server can be configured using the `httpd.conf` file. The `httpd.conf` file contains directives and modules that are configured for an instance of Apache server. Directives are instructions and are included in plain text format. The changes made to this file control the functioning of Apache Web server.

In this session, you will learn to configure Apache server, using directives and modules. In addition, you will also learn to check configurations and create index pages.

4.2 Configuring Apache Web Server

Apache server is configured from the source code depending on the platform and requirements. This is done using the configure script file present in the root directory. The `httpd.conf` file is the main configuration file, that is located in `/usr/local/apache2/conf` directory.

You can configure Apache server by including directives in the configuration files. A directive is similar to a command. The configuration file contains one directive per line. Apache Web server implements the changes specified in the configuration file only after a restart.

The `./configure` command is used to configure Apache Web server from the source code. To change the default option, `configure` command accepts different variables and command line options.

The most important option in the configure command is the `–prefix=PREFIX` option. This option specifies the installation location of Apache Web server. You can also specify the features that are to be included in Apache server by enabling and disabling modules. By default, Apache server contains a set of included modules. You can enable modules using the `--enable-module` option.

Modules are compiled using shared objects (DSOs) that are loaded or unloaded at runtime by using the option `--enable-module=shared`.

Similarly, base modules can be disabled using the `--disable-module` option. It is sometimes necessary to provide the configuration script with extra information about the location of the compiler, libraries, or header files.

Apache Web server is configured using different configuration options.

Table 4.1 lists some of the basic configuration options.

Options	Description
`--configure-cache`	Configures the `config.cache` file
`--cache-file=FILE`	Records test results in the specified filename. This option is disabled by default
`--help`	Displays the help file
`--no-create`	Executes the `configure` script without generating the output files. This is used to verify the test results before generating `makefiles` for compilation
`--srcdir=DIR`	Defines the source file directory
`--version`	Displays the copyright information
`--quiet`	Does not generate messages during configuration (An alternative to this option is the `--silent` option.)
`--enable-layout=LAYOUT`	Configures the source code and builds the scripts based on the installation tree specified in the `LAYOUT`. The `config.layout` specifies the different configuration. Apache server uses the default layout named `LAYOUT`
`--bindir=DIR`	Installs third party executables in the specified location
`--includedir=DIR`	Installs C header files in the specified location
`--information=DIR`	Installs information documentation to the specified location
`--libdir=DIR`	Installs object code libraries to the specified location
`--build=BUILD`	Defines the type of the system. For example, version number
`--disable=FEATURE`	Disables the specified feature
`--enable=FEATURE`	Enables the specified feature

Table 4.1: Basic Configuration Options

4.3 Modules

Apache Web server provides a number of `MultiProcessing` Modules (MPMs) that enable Apache server to run in different modes, to satisfy the requirements of every infrastructure. You must include the required MPMs and configure Apache server to allow execution in a variety of modes, such as a process-based, hybrid (process and thread), or event-hybrid mode. Modules add functionality to the Web server. Modules extend the range of functions available in Apache server that can range from server-side programming language support to authentication methods. Modules are statically or dynamically included with the core.

To include modules statically, the source code has to be added to the server source distribution. Dynamically included modules add functionality to the server by executing as shared libraries during start-up or restart of the server.

Some of the modules are as follows:

- **mod_mime** - Allocates content metadata to the content selected for an HTTP response by linking patterns or extensions in the filenames to the metadata values. This module associates the contents' language, character set, and content-encoding to the file. Additionally, the `mod _ mime` module defines the handler and filters that process the content. It delivers the content to the client based on language and provides a list of the currently accepted MIME type files that can have more than one extension. For example, if the file `hello.html.ge` maps to content type `text/html` and the language German, then the file `hello.ge.html` will also map to the same information. You can assign multiple content type and language to a single file.

  These files are encoded to simplify transactions on the Internet. The `mod_mime` module allows you to append a content-encoding entity-header field to a given file. The directives used by this module are listed in table 4.2.

Directive	Function
`AddCharset`	Specified content character set is mapped to the filename extensions
`AddEncoding`	Corresponding encoding type is mapped to the specified filename
`AddHandler`	Specified handler is mapped to the filename extensions
`AddLanguage`	Specified content language is mapped to the filename extensions
`AddType`	Specified content type is mapped with the filename extension
`TypesConfig`	Specifies the location of the `mime.types` file

Table 4.2: Directives used by the mod_mime Module

- **mod_so** - Loads modules and executable files into Apache server at runtime. Table 4.3 lists the directives used by this module.

Directive	Function
`LoadFile`	Links the named object files at startup or after a restart
`LoadModule`	Links the object file and includes the module structure to the list of existing modules

Table 4.3: Directives used by the mod_so Module

- **mod_log_config** - Logs the client requests sent to the Apache Web server. The log format can be customized and the log report can be generated and written to a file.

In addition, the log can be written to a program. Table 4.4 lists the directives used by this module.

Directive	Function
CookieLog	Specifies the filename used to log cookies
CustomLog	Defines the filename and the format of the log file
Logformat	Defines the log format or relates an explicit format with a nickname
TransferLog	Specifies the location of a log file

Table 4.4: Directives used by the mod_log_config Module

- **mod_setevnif** - The `mod _ setenvif` module enables you to define internal environment variables. The environment variables can be used by different modules of the server to decide the action to be executed. Table 4.5 lists the directives used by this module.

Directive	Function
BrowserMatch	Defines environment variables depending on HTTP User-Agent
BrowserMatchNoCase	Defines environment variables depending on User-Agent without respect to case
SetEvnIf	Defines environment variables on the basis of request attributes
SetEvnIfNoCase	Defines environment variables based on the basis of request attributes without respect to case

Table 4.5: Directives used by the mod_setenvif Module

Note: The directives are processed in the order in which they appear in the configuration file.

- **mod_alias** - The `mod _ alias` module allows you to map the requests for URLs. Table 4.6 lists the directives used by this module.

Directive	Function
Alias	Links URLS to file system location
AliasMatch	Links URLS to file system location with the help of regular expression
Redirect	Transmits an external redirect (after seeking the client's consent) to obtain a different URL
RedirectMatch	Transmits an external redirect based on a regular expression match of the current URL
RedirectPermanent	Returns an external permanent redirect link. This directive instructs the client to request a different URL
RedirectTemp	Returns an external temporary redirect link. This directive instructs the client to request a different URL

Directive	Function
ScriptAlias	Directs a URL to a file system location and defines the target as a CGI script
ScriptAliasMatch	Maps a URL to a file located on the Web server using a regular expression and defines the target as a CGI script

Table 4.6: Directives used by the mod_alias Module

- **mod_cgi** - Enables execution of CGI scripts. Table 4.7 lists the directives used by this module.

Directive	Function
ScriptLog	Defines the location of the CGI script error log file
ScriptLogBuffer	Records the maximum number of PUT or POST requests in the scriptlog
ScriptLogLength	Defines the size limit of the CGI script log file

Table 4.7: Directives used by the mod_cgi Module

- **mod_userdir** - The `mod _ userdir` directive enables you to control access to specific directories. This module uses the `UserDir` directive. The syntax to use this directive is as follows:

```
UserDir directory-filename
```

where,

`directory-filename` - contains any one of the following options:

- Name of a directory
- The keyword `disabled` to stop username-to-directory translations
- The keyword `disabled` used with a list of usernames
- The keyword `enabled` used with a list of usernames

- **mod_include** - The `mod _ include` module provides a filter to process request for files before sending them to the client. Apache server processes the files using specially formatted comments. These Standard Generalized Markup Language (SGML) comments are also known as elements. The elements enable inclusion of files and programs and setting and printing of environment variables.

The directives used by this module are listed in table 4.8.

Directive	Function
SSIEnableAccess	Sets the -A flag when processing flow control based on conditions
SSIEndTag	Specifies the string (or character) to terminates an include element
SSIErrorMsg	Defines the error message to be displayed when an SSI error is encountered
SSIStartTag	Specifies the string (or character) to initiate an include element
SSITimeFormat	Defines the format to display date strings
SSIUndefinedEcho	Defines the string to display while a variable is not set and echoed
XHitBack	Parses SSI directives in files when the execute bit is enabled

Table 4.8: Directives used by the mod_include Module

- **mod_dir** - The `mod _ dir` module provides a trailing slash redirect, which helps in eliminating automatic index generation. Apache server uses the trailing slash redirect when the client requests for a directory. Table 4.9 lists the directives used by the `mod _ dir` module.

Directive	Function
DirectoryIndex	Defines a list of resource to search when the user requests a directory
DirectorySlash	Enables or disables trailing slash redirects
FallbackResource	Sets a default URL for requests that do not contain a match

Table 4.9: Directives used by the mod_dir Module

- **mod_status** - The `mod _ status` module displays information related to the activities and performance of Apache Web server. The following information is displayed in an HTML page:

  - Number of requests processed
  - Number of idle processes
  - Status of modules, number of requests served, and the total number of bytes processed
  - Server startup, and restart time
  - Averages of requests processed per second, and the number of bytes processed
  - Percentage of CPU used by Apache Web server and each module
  - Hosts and requests processed

Table 4.10 lists the directives used by the `mod_status` module.

Directive	Function
`ExtendedStatus`	Monitor extended status information for every request
`SeeRequestTail`	Verifies whether the first or the last 63 characters of the request is displayed when the number of characters in a request is greater than 63

Table 4.10: Directives Used by the mod_status Module

- **worker** - The `worker` module implements a hybrid server. This module uses multiple threads to process a number of requests, thereby utilizing the available resources economically. It is very stable and processes multiple requests with threads. Table 4.11 lists some of the directives used by the `worker` module.

Directive	Function
`ThreadsPerChild`	Defines the number of threads generated by each child process
`MaxClients`	Specifies the highest number of connections that will be processed at the same time
`StartServers`	Defines the number of child server processes to be created at startup
`MinSpareThreads`	Defines the minimum number of idle threads available to process requests
`MaxSpareThreads`	Defines the maximum number of idle threads available to process a request
`ServerLimit`	Defines the upper limit on the number of processes that can be configured
`ThreadLimit`	Defines the upper limit on the number of threads that can be configured per child process
`MaxRequestPerChild`	Defines the number of requests to be processed by an individual child server
`User`	Defines the user ID with which the server will process requests
`Groups`	Specifies the group for which the server will handle requests

Table 4.11: Directives used by the worker Module

- **mod_vhost_alias** - The `mod _ vhost _ alias` module generates dynamically configured virtual hosts. The IP address and the `Host: header` of an HTTP request specified in the pathname help this module to determine the files to be served.

Table 4.12 lists the directives used by the `mod_vhost_alias` module.

Directive	Function
`VirtualDocumentRoot`	Defines the location of the document root for a specific virtual host
`VirtualDocumentRootIP`	Defines the location of the document root for a specific virtual host. This directive uses the IP address of the server as compared to the named virtual host.
`VirtualScriptAlias`	Defines the location of the CGI directory for a specific virtual host
`VirtualScriptAliasIP`	Defines the location of the CGI directory for a specific virtual host. It uses the IP address of the server to differentiate between multiple virtual hosts.

Table 4.12: Directives used by the mod_vhost_alias Module

- **mod_rewrite** - The `mod _ rewrite` module implements a rule-based rewriting engine to rewrite requested URLs. It provides a flexible and strong URL manipulation mechanism by supporting different rules and conditions for each rule. This module works with absolute URLs and generates a query string on the result. Table 4.13 lists the directives used by the `mod _ rewrite` module.

Directive	Function
`RewriteBase`	Defines the base URL for a directory rewrite
`RewriteCond`	Specifies a condition when the rewriting will take place
`RewriteEngine`	Enables or disables the rewriting engine
`RewriteLock`	Defines the name of the lock file used for managing the `RewriteMap` directive
`RewriteLog`	Defines the name of the file used for logging rewrite action it performs
`RewriteLogLevel`	Defines the verbosity of the log file used by the rewrite engine
`RewriteMap`	Defines a mapping function for searching a key
`RewriteOptions`	Defines options for the rewrite engine. The option can include inheritance of the parent server configuration.
`RewriteRule`	Specifies rules for the rewriting engine

Table 4.13: Directives used by the mod_rewrite Module

- **mod_expires** - The `mod _ expires` module manages the `Expires` HTTP header and the `max-age` directive of the `Cache-Control` HTTP header in a server response. You can set the expiry date relative to either the instance the source file was last modified or to the instance when the client last accessed the file. These headers contain information about the validity of the file.

Table 4.14 lists the directives supported by the `mod_expires` module.

Directive	Function
`ExpiresActive`	Enables creation of Expires headers
`ExpiresByType`	Sets the value of the Expires header configured by MIME type
`ExpiresDefault`	Defines the default algorithm for determining the expiry time

Table 4.14: Directives used by the mod_expires Module

- **mod_proxy** - The `mod _ proxy` module creates a gateway for the Apache Web server. This module incorporates a number of protocols and different load-balancing algorithms. A set of modules must be loaded either statically or dynamically into the server to provide the essential features. Table 4.15 lists some of the directives used by the `mod _ proxy` module.

Directive	Function
`ProxyRequests`	Enables redirecting of requests to the forward proxy server
`ProxyPass`	Maps the URL of remote servers into the local server URL space. This enables the remote servers to function as a proxy server.
`<Proxy>`	Contains directives applicable to proxy servers
`ProxyBlock`	Contains resources that are prohibited from being proxied. The resources include words, host names or IP addresses, and domain names.
`ProxyRemote`	Specifies the remote proxy to process specific requests
`NoProxy`	Specifies a list of subnets, hosts, and/or domains that will bypass the proxy and will be connected directly
`ProxyDomain`	Specifies the default domain name for proxied requests
`ProxyBadHeader`	Defines the method of processing bad header lines in a response

Table 4.15: Directives used by the mod_proxy Module

Note: You can configure the Apache server as a forward or a reverse proxy.

- **mod_ssl** - The `mod _ ssl` module provides Secure Sockets Layer v2/v3 and Transport Layer Security v1 protocol for the Apache Web server. Table 4.16 lists some of the directives used by the `mod _ ssl` module.

Directive	Function
`SSLCACertificateFile`	Defines the file consisting integrated PEM-encoded CA Certificates for client authorization
`SSLCARevocationPath`	Defines the directory where the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) of clients are located to verify the communication

Directive	Function
SSLCertificateFile	Specifies the PEM-encoded Certificate file for the Web server. In addition, it also directs to the corresponding RSA or DSA private key file.

Table 4.16: Directives used by the mod_ssl Module

4.3.1 Installing Modules in Bulk

The configuration script in Apache server provides options that permit you to enable or disable a number of modules at the same time. The set of modules available in Apache server are categorized into three levels, `default`, `most`, and `all`.

Table 4.17 lists the modules categorized into `default`, `most`, and `all` levels.

Default	Most	All	Explicitly Enabled
mod_access	mod_auth_anon	mod_cern_meta	mod_file_cache
mod_actions	mod_auth_dbm	mod_mime_magic	mod_charset_lite
mod_alias	mod_auth_digest	mod_unique_id	mod_cache
mod_asis	mod_dav	mod_usertrack	mod_disk_cache
mod_auth	mod_dav_fs		mod_mem_cache
mod_autoindex	mod_expires		mod_deflate
mod_cgi/mod_cgid	mod_headers		mod_ext_filter
mod_dir	mod_info		mod_proxy
mod_env	mod_rewrite		mod_proxy_connect
mod_http			mod_proxy_ftp
mod_imap			mod_proxy_http
mod_include			mod_ssl
mod_log_config			mod_suexec
mod_mime			
mod_negotiation			
mod_setenvif			
mod_so			
mod_status			
mod_userdir			

Table 4.17: Apache Modules

The modules included in the `most` category are also included in the default modules. The modules in the `All` category are also included in the `default` and `most` modules.

The most and all arguments of the --enable-modules option of the ./configure command enable Apache server to install multiple modules simultaneously.

4.4 Directives

Directives are instructions included in the configuration files that control the functioning of Apache Web server. Directives that are included in the httpd.conf file is applicable to the entire Web server. Apache server implements the changes included in the httpd.conf file only when the Web server is restarted. Therefore, Apache Web server must be restarted to implement changes made to the configuration file.

Apache processes the directives in the httpd.conf file, line-by-line. The first word in such a line is the name of the directive whereas the remaining lines are the directive parameters. If the parameter is exceeding more than one line of a directive, a backslash \ is included at the end to indicate that the parameter is continuing to the next line. Each directive has a fixed set of contexts.

The following directives can be used to configure Apache Web server:

- **Allow Directive** - The Allow directive, provided by mod _ authz _ host, controls access of the hosts to a specific area of the server. You can control the access by hostname, IP address, IP address range, or by other characteristics of the client request.

  The syntax for Allow directive is as follows:

```
Allow from all|host|env=[!]env-variable [host|env=[!]env-variable] …
```

  where,

  all - specifies that all hosts are allowed access

  hosts - specifies that only a particular host or group of hosts are allowed access

  env=env-variable - specifies access only if environment variables are present

Note: from is always specified as the first argument to this directive.

For example, to allow access from the site example.com, enter the following code as shown in Code Snippet 1 in the httpd.conf file.

Code Snippet 1:

```
Allow from example.com
```

The code enables access to the example.com domain.

For example, to allow access to `142.2.0`, enter the following code as shown in Code Snippet 2 in the `httpd.conf` file.

Code Snippet 2:

```
Allow from 142.2.0
```

Note: If you define `Allow from all`, then the entire range of hosts is allowed access.

- **Deny Directive** - Provided by `mod _ authz _ host` module, this directive restricts access to the server based on the hostnames, IP address, or environment variables.

  The syntax of `Deny` directive is as follows:

  ```
  Deny from all|host|env=[!]env-variable [host|env=[!] env-variable] …
  ```

  where,

  `all` - restricts access to all hosts

  `host` - restricts access to the host specified

  `env=env-variable` - restricts access to an environment variable

  For example, to deny access to `142.2.0`, enter the following code as shown in Code Snippet 3 in the `httpd.conf` file.

  Code Snippet 3:

  ```
  Deny from 142.2.0
  ```

- **Order Directive** - Provided by the `mod_authz_host`, this directive controls the default access state. It evaluates the order in the manner the `Allow` and `Deny` directives are evaluated.

  The syntax of `<Order>` directive is as follows:

  ```
  Order ordering
  ```

  For example, to allow access to all the hosts in `example.com` and deny from all, enter the following code as shown in Code Snippet 4 in the `httpd.conf` file.

Code Snippet 4:

```
Order Deny Allow

Deny from all

Allow from example.com
```

The code allows access to hosts only from `example.com`.

- **<Location> Directive** - Provided by core module, this directive limits the use of other enclosed directives by URL. In other words, this directive encloses other directives between the `<Location>` and `</Location>` tags and the enclosed directives are applied only to the matching URLs. It is processed in the order in which it appears in the configuration file.

The syntax for `<Location>` directive is as follows:

```
<Location URL-path | URL> … </Location>
```

If the URL paths match any of the following conditions, then the enclosed directives will be applied to the request.

- The exact location is specified
- The specified location ends in a forward slash
- The specified location ends in a trailing slash

Note: When the specified location ends in a forward or trailing slash, it is known as a context root.

For example, to generate the server information page using `Location` directive, in the configuration file, enter the following code as shown in Code Snippet 5 in the Location section of the `httpd.conf` file.

Code Snippet 5:

```
<Location /status>

SetHandler server-status

Order deny,allow

Deny for all
```

Concepts

```
Allow from 127.0.0.1

</Location>
```

In the example, the `Location` directive specifies the location. The `SetHandler` directive forces all the matching files to be processed by the handler. The `Order` directive enables the two directives `Deny` and `Allow`. The `Allow` directive allows access from a specified IP address.

The example displays a status report at the location `127.0.0.1`.

- **<IfDefine> Directive** - Provided by core module, this directive is a conditional directive. The directives in the `<IfDefine>` section are processed only if the test conditions specified at startup is fulfilled. If the test return false, instructions between the `<IfDefine>` and `</IfDefine>` tag is ignored.

The syntax for `<IfDefine>` directive is as follows:

```
<IfDefine [!]parameter-name> … </Ifdefine>
```

where,

`IfDefine` - specifies conditional directives. The directives within `IfDefine` are processed only if the test is true. If the test is false, directives included between the markers are ignored.

`parameter-name` - processes only if the parameter-name is defined

`!parameter-name` - processes only if the parameter-name is not defined.

For example, to enable network connectivity to a specific IP address, enter the following directives in the `IfDefine` section of the `httpd.conf` file.

Code Snippet 6 displays the use of the directives.

Code Snippet 6:

```
<IfDefine no_network>

<Location>

Order Deny,Allow

Deny from all

Allow from 127.0.0.1
```

```
    </Location>

</IfDefine>
```

In the example, the <IfDefine> directive specifies the method used for network connectivity. The <Location> directive specifies the location from where the network should be connected. The Order directive enables the two directives Deny and Allow. The Allow directive allows access from a specified IP address.

The <IfDefine> directive enables network connectivity from 127.0.0.1 and denies for all others.

- **Include Directive** - The Include directive incorporates other configuration files from within the server configuration files at runtime. To include files in alphabetical order (fnmatch()) is used. The path to these configuration files starts with a slash. Apache httpd reads all files in a directory or subdirectory if the Include directive specifies a directory. Temporary files in directories can cause the Apache daemon, httpd to fail. Therefore, including entire directories is not recommended.

  The syntax for Include directive is as follows:

```
Include file-path | directory-path
```

  For example, to include a file in the conf directory using Include directive, in the configuration file, enter the following code as shown in Code Snippet 7 in the Include section of the httpd.conf file.

  Code Snippet 7:

```
Include /usr/local/apache2/conf/ssl.conf
```

Note: It is recommended to use the wildcard syntax, such as *.conf.

In the example, the Include directive specifies the directory in which the file should be included.

- **TypesConfig Directive** - The TypesConfig directive defines the location of the media types configuration file. This file creates the default list of mappings from file extensions to content types. Web server administrators use the mime.types file that relates common filename extensions with the official list of IANA registered media types and a large number of unofficial types.

  The syntax for the TypesConfig directive is as follows:

```
TypesConfig file-path
```

- **<IfModule> Directive** - The `IfModule` directive is used to process the directives enclosed within it only if the module is present or absent. In other words, the directives are processed subject to the availability of a module. The directives in the `<IfModule>` section are processed only if the test returns true. If the test returns false directives between the start and the end markers is ignored.

  The syntax for `<IfModule>` directive is as follows:

```
<IfModule[!]module-name… </IfModule>
```

  where,

  `<IfModule> </IfModule>` - contains the list of directives to be processed

  `module name` - executes the directives only when the module is included

  `!module name` - reverses the test and executes the directives only when the module is not included

  For example, to specify the `ServerName` if the core module exists, enter the following code as shown in Code Snippet 8 in the `httpd.conf` file.

  Code Snippet 8:

```
<IfModule  core.c>

  ServerName www.myserver.com: 80

</IfModule>
```

  In the example, if the module `core.c` is included in Apache server, then the `ServerName` directive is processed. The name of the server is set as `www.myserver.com` and the port is set to `80`.

- **Listen Directive** - Provided by `prefork`, `perchild`, and `worker` modules, this directive instructs Apache server to process requests from a specific IP addresses or port. By default Apache server processes requests from all IP address. This is one of the most important directive of the `httpd.conf` file. The server fails to start in the absence of this directive.

  The syntax for Listen directive is as follows:

```
Listen[IP-address:]portnumber
```

  For example, to bind Apache server to a specific port using the `Listen` directive, in the configuration file, enter the following command as shown in Code Snippet 9 in the Listen section of the `httpd.conf` file.

Code Snippet 9:

```
Listen 192.168.2.1:80

Listen 192.168.1.5:800
```

In the code, the `Listen` directive specifies that the Apache server should process requests from the specified port. It binds Apache server to the port `192.168.2.1:80` and `192.168.1.5:800`.

- **<Directory> Directive** - The Directory directive contains a group of directives that can be applied to a specific directory or sub-directories. The `<Directory>` and `</Directory>` tags are used to include a group of directives that will be applicable only to the named directory, sub-directories of that directory, and the files within the respective directories. You can use any directive that is acceptable in a directory context.

The syntax for `<Directory>` directive is as follows:

```
<Directory directory-path> … </Directory>
```

Directory path is either the full path to a directory or a wildcard string. In a wildcard string, you can use the following notations:

- `?` - for any single character
- `*` - for a series of characters
- `[]` - for character ranges

Note: Wildcards do not correspond to a / character, so `<Directory /*/example_html>` will not match `/home/user/example_html`, but `<Directory /home/*/example_html>` will match.

For example, to match directories with specific names, enter the following code as shown in Code Snippet 10 in the `httpd.conf` file.

Code Snippet 10:

```
<Directory ~ "^/var/ .*/[0-9]{3}">
```

In the code, the `Directory` directive specifies the directory to be matched with. It would match directories in `/var/` and that which consisted of three numbers.

- **<DirectoryMatch> Directive** - Provided by core module, this directive is an alternative and simpler form of `<Directory>` directive. It encloses a group of directives that apply only to the named directory and to its subdirectories.

Concepts

The syntax for <DirectoryMatch> directive is as follows:

```
<DirectoryMatch regex> … </DirectoryMatch>
```

For example, to match all the directories in the alphabetical order enter the following command as shown in Code Snippet 11 in the DirectoryMatch section of the configuration file httpd.conf.

Code Snippet 11:

```
<DirectoryMatch "/[A-Z][0-9](2)/">
```

In the code, the DirectoryMatch directive matches with all the specified directories.

The DirectoryMatch directive matches directories in [A-Z] that consist of two numbers.

- **<Files> Directive** - Provided by core module, this directive contains instructions that are specific to matched filenames. The directives specified in this section apply to any object matching with the filename. They are processed in the order they appear in the configuration file. The filename argument should contain a filename or a wildcard string.

The syntax for <Files> directive is as follows:

```
<Files filename> … </Files>
```

For example, to list all files with extension .gif enter the following command as shown in Code Snippet 12 in the Files section of the configuration file, httpd.conf.

Code Snippet 12:

```
<Files~"\.gif.* ">
```

In the code, the Files directive matches all files with the extension .gif.

- **<FilesMatch> Directive** - The <FilesMatch> directive restricts the scope of the included directives by filename. It is the same as the <Files> directive with the only difference that it accepts a regular expression.

The syntax for <FilesMatch> directive is as follows:

```
<FileMatch regex> … </FileMatch>
```

For example, to match all files with the extension .gif and .jpeg, enter the following command as shown in Code Snippet 13 in the FilesMatch section of the configuration file, httpd.conf.

Code Snippet 13:

```
<FilesMatch "\.(gif|jpe?g)$">
```

In the code, the FilesMatch directive is applied to all files with the extension of .gif and .jpeg.

- **<LocationMatch> Directive** - Provided by core module, this directive includes instructions to be applied to regular expression matching URL. The directive accepts a regular expression as an argument rather than a simple string.

The syntax for the <LocationMatch> directive is as follows:

```
<LocationMatch regex> … </LocationMatch>
```

For example, to match the location with all uppercase words enter the following code as shown in Code Snippet 14 in the LocationMatch section of the configuration file.

Code Snippet 14:

```
<LocationMatch"^/[A-Z][a-z]+/$ ">
```

The example enables to match the URLs containing words written in uppercase and lowercase.

- **SetHandler Directive** - Provided by core module, this directive redirects all files with identical extensions to be processed through a handler when included in the .htaccess file or the <Location> or <Directory> section of the httpd.conf file. The file name extensions must be specified in the syntax of the SetHandler directive.

The syntax for the SetHandler directive is as follows:

```
SetHandler handler-name|None
```

For example, to display the server information whenever a URL http://servername/info is invoked enter the following code as shown in Code Snippet 15 in the <Location> section of the httpd.conf file.

Concepts

Code Snippet 15:

```
<Location>

     SetHandler server-information

</Location>
```

In the code, the `Location` directive matches URLs to a specific location. The `SetHandler` directive processes the requests that satisfy the conditions specified in the handler.

- **AccessFileName Directive** - Provided by core module, this directive defines the name of the distributed configuration file. The names are included as a list. While processing a request, Apache server matches the names of existing configuration file with the names from this list in every directory of the path to the document.

The syntax for the `AccessFileName` directive is as follows:

```
AccessFileName filename [filename] …
```

Consider an example, where the `httpd.conf` file contains the code as shown in Code Snippet 16.

Code Snippet 16:

```
AccessFileName .php
```

The code specifies that the Apache server will search `/.php`, `/usr/.php`, `/usr/local/.php` and `/usr/local/web/.php` for directives before returning the document `/usr/local/web/index.html`.

- **AllowOverride Directive** - Provided by core module, this directive contains the list of directives that are included in the `.htaccess` files. If this directive is set to None, then the `.htaccess` file is ignored. In this case, Apache server will not attempt to read `.htaccess` files.

The syntax for the `AllowOverride` directive is as follows:

```
AllowOverride All|None|directive-type [directive-type]...
```

The following are the directive types of the `AllowOverride` directive:

- **`AuthConfig`** - specifies all the authorization directives

  For example, `AuthName`, `AuthType`, and `AuthGroupFile`

- **FileInfo** - specifies the directives controlling document types

  For example, `ErrorDocument`, `SetHandler`, and `ForceType`

- **Indexes** - specifies all directives controlling directory indexing

  For example, `DirectoryIndex`, `FancyIndexing`, `IndexOptions`

- **Limit** - specifies the directives controlling host access

  For example, `Allow`, `Order`, and `Deny`.

- **Options** – specifies the directives controlling specific directive features

  For example, `Options`, `XbitHack`.

4.5 Testing Configuration

The Apache server has to be tested for correct configuration using the `httpd.conf` file.

To configure `httpd.conf` file, perform the following steps:

1. To change the directory, enter the following command at the command prompt:

```
cd /usr/local/apache2/conf
```

2. To copy the `httpd.conf` file, enter the following command at the command prompt.

```
cp httpd.conf httpd-cp.conf
```

3. To open the `httpd.conf` file using the vi editor, enter the following command at the command prompt.

```
vi httpd.conf
```

4. Press `I` to enter `INSERT` mode.

5. To define the port, enter the following code as shown in Example 1 in the `<Listen>` section of the `httpd.conf` file.

Concepts

Example 1:

```
Listen 192.168.0.220:80
Listen 127.0.0.1:80
```

6. To set limitations for specified file extensions, enter the following code as shown in Example 2 in the `<Order>` section of the `httpd.conf` file.

Example 2:

```
<Files ~ "^\.ht">
Order Allow, Deny
Deny for all
</Files>
```

7. To define the location where the server information page is to be displayed, enter the following code as shown in Example 3, in the `<Location>` section of the `httpd.conf` file.

Example 3:

```
<Location /server-information>
SetHandler server-information
Order deny,allow
Deny for all
Allow from 192.168
</Location>
```

8. Press Esc and `:wq!` to save changes to the `httpd.conf` file.

9. Restart Apache Web server.

```
The httpd.conf file is configured.
```

Note: You can also use the `gedit` text editor to edit and make changes to the `httpd.conf` file.

4.6 Creating Index Pages

The index page for Apache is an `html` file stored in the document `root`. The default location of the document root folder of Apache server is `/usr/local/apache2/htdocs`. The `index.html` file is the first page that opens when you access a particular site. The index page contains links that enable you to navigate through the site. The `DirectoryIndex` directive specifies the name of the index file. The module `mod_dir` provides the directive `DirectoryIndex`.

Figure 4.1 displays the default index page.

Figure 4.1: Default Index Page

You can also modify the default index page.

To modify the default index page, perform the following steps:

1. Open the `gedit` text editor.
2. Open the `index.html` file from `/usr/local/apache2/htdocs` directory.
3. Delete the existing code and enter the code as shown in Example 4:

Note: Create a backup copy of the `index.html` file before editing or modifying the contents.

Example 4:

```
<html>
<head>
     <title>Custom Index Page</title>
</head>
<body>
```

```
<h1 align=”center”>MyApache.com</h1>
<hr>
<form>
<center>
     <h3>Search For:</h3>
     <input type=”text” name=”text1”>
     <input type=”submit” value=”Go”>
<br>
<a href=”/” align = “left”>Advanced Search</a>
    
<a href=”/” align=”right”> Preferences </a>
</center>
</form>
</body>
</html>
```

4. Save the file.

5. Open the Mozilla Firefox Web browser.

6. Type `http://localhost/` in the address bar and press the **Enter** key.

Figure 4.2 displays the modified Index page.

Figure 4.2: Modified Index Page

Concepts

Summary

- The main configuration file in Apache Web server is `httpd.conf`.
- The `./configure` command compiles Apache from the source code.
- The `--prefix` option of the configure command specifies the directory to install Apache server.
- Directives are commands that set options and modules add functionality to the Web server.
- Directives placed in the main configuration files apply to the entire server.
- `Include directive` includes other configuration files in the server at runtime.
- `Listen directive` instructs Apache server to respond to a specific IP address or ports.
- `mod_mime` stands for Multipurpose Internet Mail Extensions. This module associates information with files by their file extensions.
- `mod_setevnif` module makes comparisons on any variable set by Apache server and set custom variables.
- Directory indexes are generated in the `index.html` file. Indexes can be generated using the `mod_autoindex` module.

Check Your Progress

1. Which of the following file is the main configuration file of Apache Web server?

   a. `htaccess.conf`

   b. `src.conf`

   c. `httpd.conf`

   d. `http.conf`

2. Which of the following option installs Apache server at a specific location?

   a. `--enable`

   b. `--prefix`

   c. `./configure`

   d. `--disable`

3. Which of the following directive configures Apache server to respond to a specific IP address or port?

   a. `ServerName`

   b. `ServerRoot`

   c. `User`

   d. `Listen`

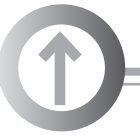
4. Which of the following module loads modules and files for Apache server at run time?

   a. `mod_mime`

   b. `mod_access`

   c. `mod_setenvif`

   d. `mod_so`



Check Your Progress

5. _____ sets the name of an index file.

   a. `Index.html`

   b. `IndexOptions`

   c. `Mod_autoindex`

   d. `DirectoryIndex`

ASK to Learn

Onlinevarsity
your e-way to learning

What
Why
Where
Questions
When
How

The E X P E R T S

are here to **HELP**

Post your questions in the ASK to LEARN section and we'll get back to you.

Session

5 Configuring Apache Web Server (Lab)

Objectives

At the end of this session, you will be able to:

- *Explain the configuration of Apache Web Server using directives*
- *Describe the process of creating a custom index page*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Configuring Apache Web Server Using Directives

The main configuration file of Apache Web server is `httpd.conf`. The functioning of Apache Web server can be controlled by including directives in the main configuration file. Directives are instructions or commands that enable or disable features in Apache Web server. Directives included in the configuration file, `httpd.conf` is applicable to the entire server. You will use some of the basic directives that are used for configuration of the server.

To configure Apache Web server, using the `<IfDefine>` directive, perform the following steps:

1. **Open the `httpd.conf` file in the gedit text editor.**

2. **Enter the following code in the `<IfDefine>` section of the configuration file.**

```
<IfDefine status>
    <Location>
    SetHandler server_status
    Order deny,allow
    deny for all
    allow from 127.0.0.1
    </Location>
</IfDefine>
```

Figure 5.1 displays the `<IfDefine>` directive.

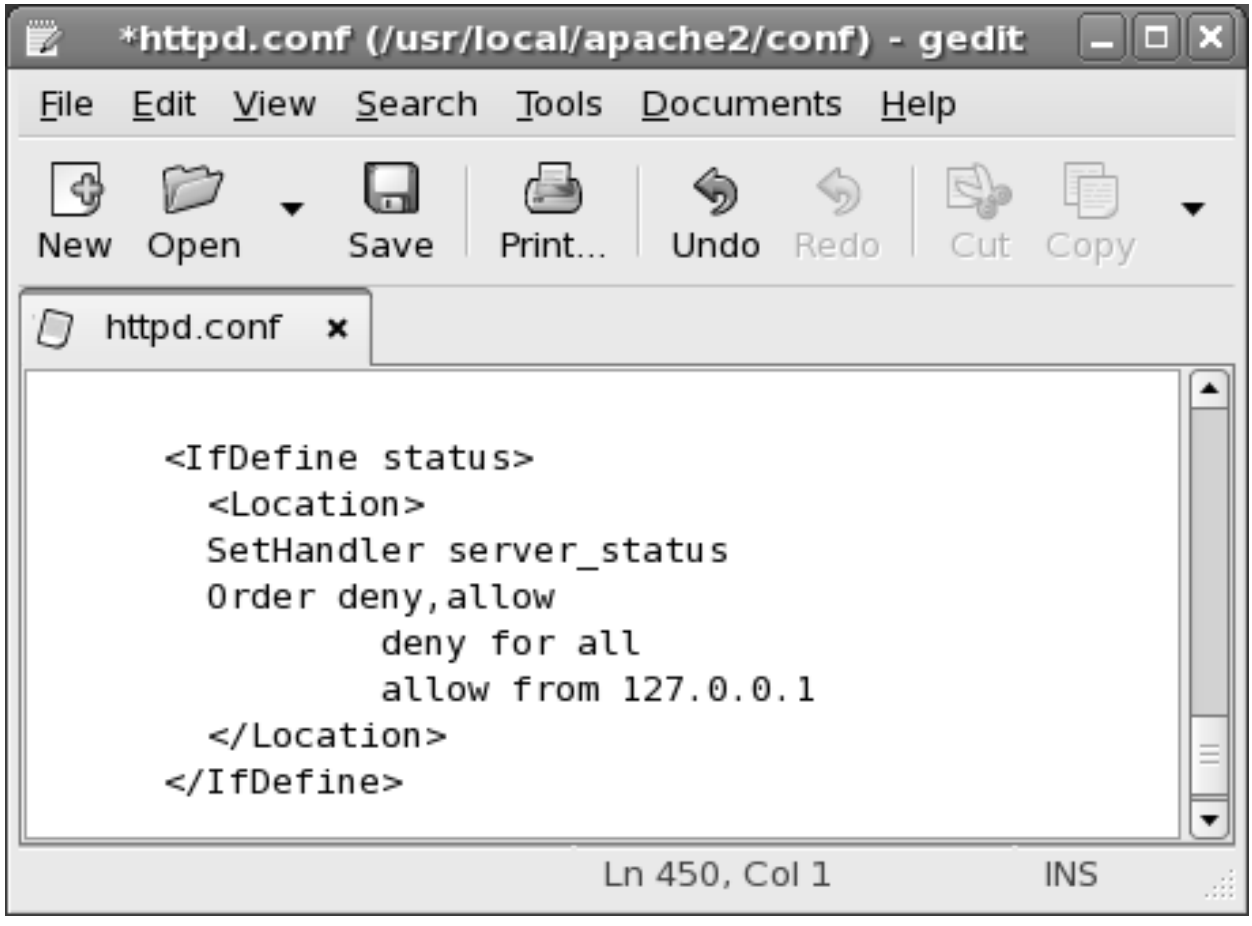


Figure 5.1: IfDefine Directive

3. **Save the `httpd.conf` file and restart Apache Web server.**

4. **Open the Mozilla Firefox Web browser.**

5. **Enter 127.0.0.1 in the address bar. The Apache home page is displayed.**

Figure 5.2 displays the home page.

Figure 5.2: Apache Home Page

To configure Apache Web server, using the `<Location>` directive, perform the following steps:

1. **Open the `httpd.conf` file in the gedit text editor.**

2. **Enter the following code in the Location section of the configuration file.**

```
<Location /server-info>
    SetHandler server-info
    order deny,allow
    Deny from all
    Allow from all
</Location>
```

Figure 5.3 displays the `<Location>` directive.

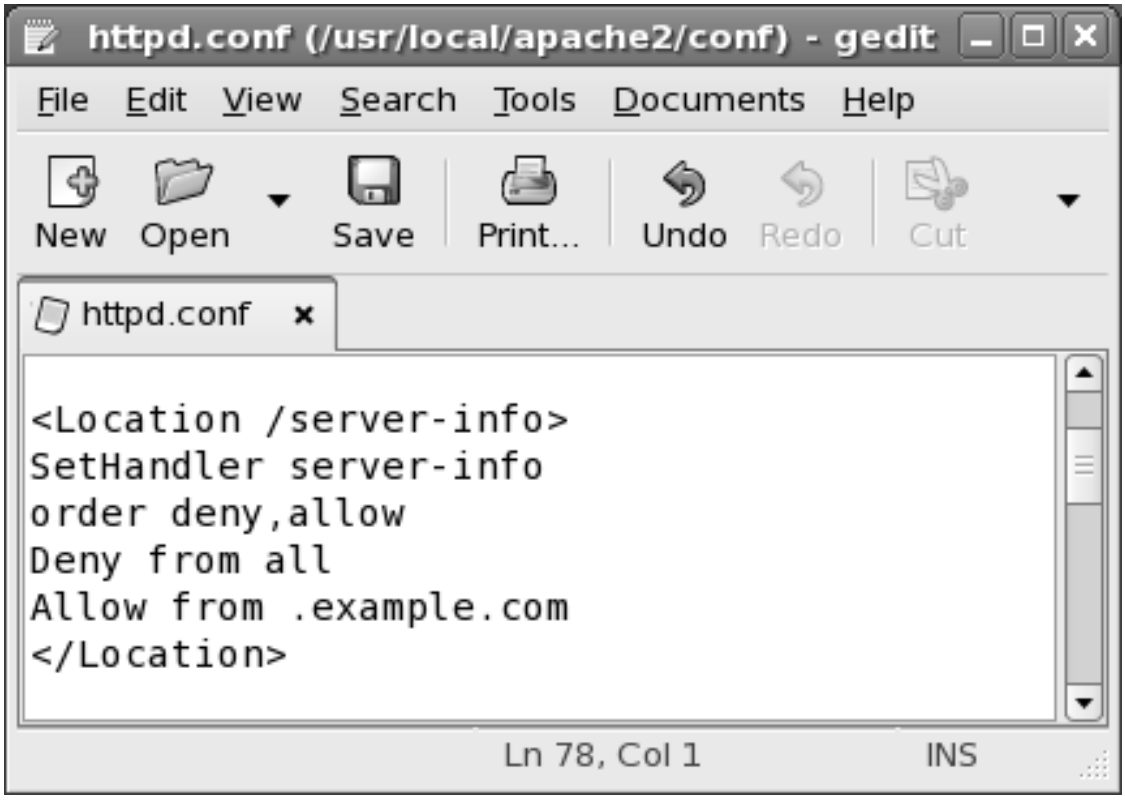


Figure 5.3: Location Directive

3.	**Save the `httpd.conf` file and restart Apache Web server.**

The `<Location>` directive requires the `mod_info` module to be enabled to display server information. The `mod_info` module enables you to view the synopsis of the Web server configuration. The server configuration information displays the enabled modules and the directives supported by the modules.

Creating a Custom Index Page

The index of a directory for a Web site is generated using the `mod_autoindex` module and stored in the `index.html` file. The `DirectoryIndex` directive defines the name of this file.

To create a custom index page, perform the following steps:

1. **Using the GUI in Linux, browse to the `/usr/local/apache2/htdocs` directory.**

2. **Open the `index.html` file, using the gedit text editor.**

3. **Delete the contents of the file and enter the following code in the `index.html` file.**

```
<html>
<head>
<meta http-equiv=”content-type” content=”text/html;
charset=UTF-8”>
<title>MyPage</title>
<style><!--body,td,a,p,.h{font-family:arial,sans-
serif;}.h{font-size: 20px;}
.q{color:#0000cc;}//-->
</style>
    <script>
    <!--function sf(){document.f.q.focus();}// -->
    </script>
    </head>
    <body bgcolor=#ffffff text=#000000 link=#0000cc vlink=#551a8b
    alink=#ff0000
    onLoad=sf()>
    <center>
    <table border=0 cellspacing=0 cellpadding=0><tr><td>
    <img src= “logo.gif” alt=”MyPage”>
    </td></tr></table><br>
    <form action=”/search” name=f><script><!--function qs(el) {if
    (window.RegExp &&
    window.encodeURIComponent) {var
    qe=encodeURIComponent(document.f.q.value);if
    (el.href.indexOf(“q=”)!=-1) {el.href=el.href.replace(new
    RegExp(“q=[^&$]*”),”q=”+qe);} else {el.href+=”&q=”+qe;}}return
    0;}// -->
    </script>
    <table border=0 cellspacing=0 cellpadding=4><tr><td nowrap
    class=q>
```

```
        <font size=-1><b><font color=#000000>
     </font>
        </b>    <a id=1a class=q
        </a>    <a id=2a class=q
        href="/wephp?hl=en&tab=wg&ie=UTF-8&oe=UTF-8" onClick="return
        qs(this);">Web</a>    <a id=3a class=q
        href="/grphp?hl=en&tab=wg&ie=UTF-8&oe=UTF-8"
        onClick="return
    qs(this);">Groups</a>    <a id=4a class=q
        href="/nwshp?hl=en&tab=wn&ie=UTF-8&oe=UTF-8" onClick="return
        qs(this);">News</a>    <a id=5a class=q
        <font color=red>New!
        </font>
        </a>
        </sup>    <b><a href="/options/index.html"
        class=q>more &raquo;</a>
        </b></font></td></tr></table>
        <table cellspacing=0 cellpadding=0><tr>
        <td width=25%> </td>
        <td align=center><input type=hidden name=hl value=en><span
        id=hf>
        </span><input type=hidden name=ie value="UTF-8">
        <input type=hidden name=oe value="UTF-8">
        <input maxLength=256 size=55 name=q value=""><br>
        <br><input type=submit value="MyPage Search" name=btnG>
        </td><td valign=top nowrap width=25%>
        <font size=-2>  <a
        href=/advanced_search?hl=en>Advanced Search
        </a><br>  
        <a href=/preferences?hl=en>Preferences</a><br>  
        <a href=/language_tools?hl=en>
        Language Tools</a></font></td></tr></table></form>
        <br><p><br><font size=-1>
        <a href="/ads/products.html">
        Advertising Products</a> -
    - <a href=/about.html>About MyPage</a>
       <span id=hp style="behavior:url(#default#homepage)"></span>
        <script>
      //<!--if (!hp.isHomePage('http://www.MyPage.com/'))
```

Lab Guide

```
{document.write("<p>
<a href=\"/mgyhp.html\"
nClick=\"style.behavior='url(#default#homepage)';
setHomePage('http://www.MyPage.com/');\">Make MyPage Your
Homepage!</a>");}//--></script></font><p><font size=-
2>&copy;2004 MyPage - Searching 4,285,199,774 web pages
</font></p></center>
</body>
</html>
```

Figure 5.4 displays the code for creating the Custom Index page.

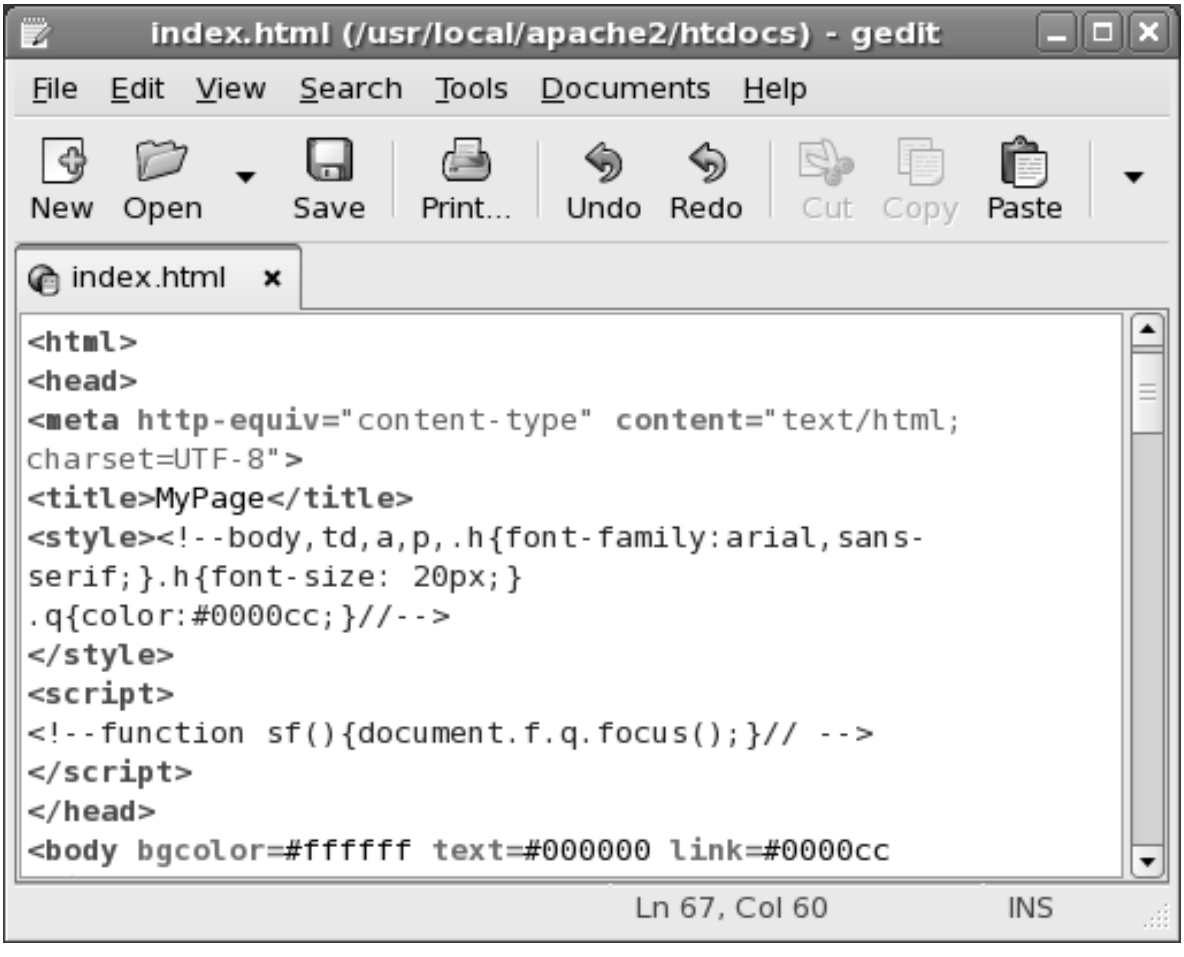


Figure 5.4: Code for Creating Custom Index Page

The image specified in the HTML code must be saved in the /usr/local/apache2/htdocs folder. Also, you need to assign read, write, and execute permissions to the image file.

4. **Save changes to the `index.html` file and restart Apache.**
5. **Copy the image file to the `/usr/local/apache2/htdocs` directory.**
6. **Using the GUI in Linux, browse to the `/usr/local/apache2/htdocs` directory.**
7. **Right-click the image file and select Properties. The File Properties dialog box is displayed.**
8. **Click the Permissions tab.**

Figure 5.5 displays the home page.

Figure 5.5: Permissions Tab

9. **Assign read, write, and execute permissions, as shown in figure 5.6.**

Figure 5.6: Assigning Read, Write, and Execute Permissions

10. **Click Close.**

11. **Open the Mozilla Firefox Web browser.**

12. **Enter `http://localhost` in the address bar and press Enter.**

Figure 5.7 displays the Custom Index page.

Figure 5.7: Custom Index Page

Do It Yourself

1. Edit the `httpd.conf` file by changing the following:

   Change the default root to `/usr/local/apache2/conf`.

   Change the port to 80.

2. Create an Index Page to display the heading, 'This is sample for Apache Web Server'. Add a command button with the name Search. The color of the text should be blue. The text on the command button should be aligned to the center of the page.

Session

6

Apache Web Server Administration

Concepts

Objectives

At the end of this session, the student will be able to:

- *Describe the administration of Apache Web Server using logs.*
- *Describe the configuration logs in Apache.*
- *Explain the contents in the log files.*
- *Explain the creation of the custom error pages.*

6.1 Introduction

The Web server adds an entry in the server log file for all the communication that takes place between the server and the client. Apache has flexible Web logging features. It allows the user to log information whenever an error is generated or a request is made to the server. You can customize the content of the log files and the error pages. You can monitor the server performance and errors using the information recorded in the log files. You can also browse the log files to analyze the recorded information.

In this session, you will learn to manage the Apache Web server by configuring logs and their file contents. You will also learn to analyze the log files. In addition, you will learn to create customized error pages.

6.2 Managing Apache Web Server

The Web administrator has to monitor the server frequently for performance and reliability. The recorded log information enables you to monitor the performance and potential security issues. The log feature of the Apache server specifies the location of the log files and the log format to the server.

The access to log files and the directory where the log files are stored must be restricted and should be granted to the administrator only. Unrestricted access to log files can be a threat to the Web server. A link can be created from the log file to a system file that is overwritten with new login information. It is therefore important to ensure that the stored log files and the directories are writeable only by the `root` user on the Linux server.

6.2.1 Types of logs in Apache

The Apache HTTP Web server provides different methods to log different server activities, such as the initial request, the URL mapping process, the final resolution of the connection, and any errors that may have occurred in the process.

In addition to this, third-party modules can render logging capabilities or include entries into the existing log files. Applications such as CGI programs, or PHP scripts, or other handlers can send messages to the server error log.

Each time a browser sends a request to the `HTTP` server, Apache server stores the information in the log files. The log files track information and server performance. Apache server supports two main types of logs namely, error logs and access logs. There are other log files that enable the server to record the required information. Following are the different types of log files available in Apache:

- **Error Log** - Records errors that are encountered during the server operation. In addition, it also records the diagnostic messages, such as server restart or shut down time. This is an important log file in Apache. This file must be first accessed to diagnose problem while starting the server or processing client requests.

- **Access Log** - Records all the requests received by the server. The format of the `access _ log` file can be configured using directives. The `CustomLog` directive defines the location and content of the `access _ log` file and the `LogFormat` directive specifies the selection of the contents of the logs. The access log is also known as the `Transfer` log.

- **PID File** - Records the process ID of the parent httpd process in the `logs/httpd.pid` file at the server startup. You can change this filename with the `PidFile` directive. Each process is assigned a unique number for distinct identification. The administrator requires the process ID for restarting and terminating the daemon.

- **Script Log** - Records the input to and the output from the CGI scripts to aid in debugging. The `ScriptLog` directive sets the CGI script error log file.

- **Piped Log** - Enables Apache server to write the error and access logs to an executable process instead of a file. This increases the flexibility of the logging feature. To write logs to a pipe, replace the filename with the pipe character "|" followed by the executable file name. On restart, the server starts the piped-log processes. The piped log programs typically run as `root` and are simple and secure. Piped logs enable log rotation without restarting the Web server.

6.3 Configuring Logs in Apache

The default location for the log files in Apache server is `/usr/local/apache2/logs`. To configure Apache server for logging, the log directives must be included in the `httpd.conf` file. The default location for the `httpd.conf` is `/usr/local/apache2/conf`.

6.3.1 Configuring Error Logs

The `ErrorLog` records the errors encountered by the server and the contents of this file cannot be customized. The `Errorlog` directive defines the name of the file where the server records any error it encounters.

If the file name and location is not specified, Apache creates the log file in the location specified in the `ServerRoot` directive. The `Errorlog` directive is supported by the core module.

The syntax for the `ErrorLog` directive is as follows:

```
ErrorLog filepath
```

where,

`filepath` - specifies the location and name of the file that will record the errors

For example, to specify the location and name of the file to record the errors, enter the code as shown in Code Snippet 1 in the `httpd.conf` file.

Code Snippet 1:

```
ErrorLog /usr/local/apache2/log/new_error_log
```

Configuring the Error Log Level

A lot of disk space and processing time is consumed if Apache is configured to record each error in the error log. The `LogLevel` directive supported by the core module defines the type of the error to be recorded in the error log. The `LogLevel` directive defines the content and alters the verbosity of the messages to be recorded in the error logs.

The syntax for the `LogLevel` directive is as follows:

```
LogLevel level
```

where,

`level` - specifies the severity of the error

The severity of errors, in order of decreasing significance is listed in table 6.1.

Level	Severity of the Error
`emerg`	Indicates that the system is unusable
`alert`	Indicates that immediate action must be taken
`crit`	Indicates a critical condition
`error`	Indicates a non-critical error condition
`warn`	Indicates a warning condition
`notice`	Indicates normal but significant condition

Level	Severity of the Error
info	Indicates an informational message
debug	Indicates a debug level message

Table 6.1: Severity of Errors in Apache

When you specify a particular level, messages from all other levels of higher significance will also be reported. For example if you set the value of LogLevel to info, then messages with log levels of notice and warn will also be posted. Therefore, it is advisable to set the LogLevel to at least crit.

Following are some of the examples for the LogLevel directive:

To log critical errors or higher-level messages, enter the code, as shown in Code Snippet 2, in the httpd.conf file.

Code Snippet 2:

```
LogLevel crit
```

The crit parameter of the LogLevel directive in Code Snippet 2 specifies to record critical condition errors. Apache also records errors with emerg and alert importance when the LogLevel directive is configured with the crit parameter.

To log all messages including the non-error messages, enter the code, as shown in Code Snippet 3 in the httpd.conf file.

Code Snippet 3:

```
LogLevel debug
```

The debug parameter of the LogLevel directive enables you to debug and troubleshoot Web server issues.

To view the error_log file, perform the following steps:

1. Using the GUI in Linux, browse to the /usr/local/apache2/logs directory.

2. Open the error_log file using the gedit text editor.

Concepts

Figure 6.1 displays the Apache server `error_log`.

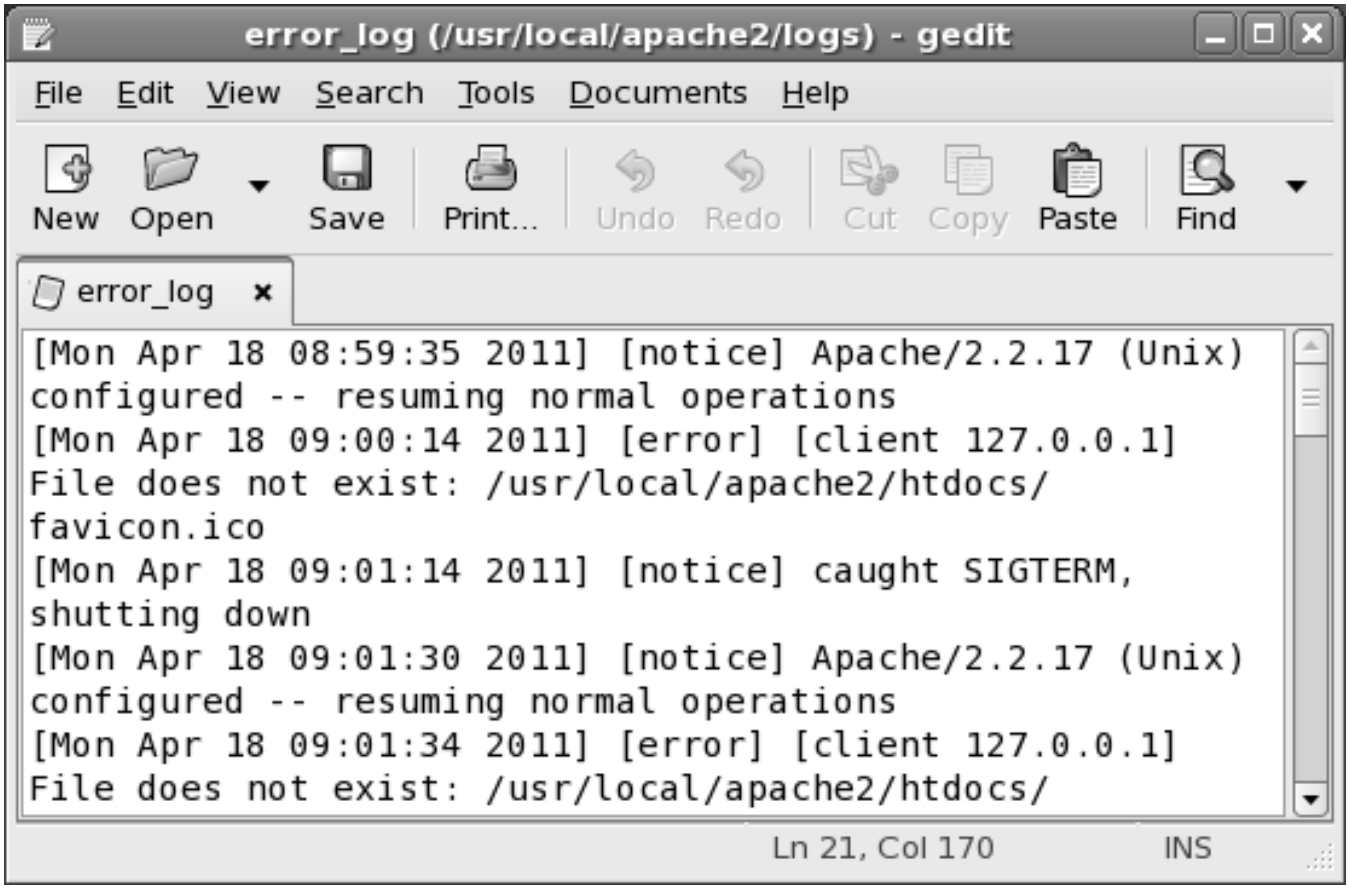


Figure 6.1: Apache Server error_log

In figure 6.1, the `error_log` displays one entry per request. Consider the second entry. It consists of the following information:

- `Mon Apr 18 09:00:14 2011` - specifies the date and time of the message
- `error` - specifies the severity of the error
- `client 127.0.0.1` - specifies the IP address of the client generating the error
- `File does not exist` - specifies the error message
- `/usr/local/apache2/htdocs/favicon.ico` - specifies the file-path for the requested document

6.3.2 Configuring Access Logs

The `mod_log_config` module provides directives to configure access logs. The `TransferLog` directive specifies the name and location of a log file. Although the `TransferLog` and the `CustomLog` directives have similar arguments, the `TransferLog` directive does not enable you to define the log format explicitly or for conditional logging of requests.

The syntax for the `Transferlog` directive is as follows:

```
TransferLog filepath
```

where,

`filepath` - specifies the location of the log file

For example, to specify the location of the `access_log` file, enter the code, as shown in Code Snippet 4, in the `httpd.conf` file.

Code Snippet 4:

```
TransferLog /usr/local/apache2/log/access_log
```

6.4 Configuring Log File Contents

The file size increases when Apache server logs information in the log files. This consumes disk space and increases the time required by the server to process requests. The `CustomLog` and `LogFormat` directives can be used to configure the contents of the `access_log` file. The two types of log format used are as follows:

- Common Log Format
- Combined Log Format
- **Common Log Format** - The common log format is the standard format for access logs. If no other log format is specified, Apache server uses the common log format with built-in definition.

  For example, a typical configuration for the `access_log` file will appear as follows:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

  The `LogFormat` directive can be used to modify the standard log format.

The syntax of the `LogFormat` directive is as follows:

```
LogFormat format-string nickname
```

where,

`format-string` - specifies the percent directives that indicates a distinct piece of information to be logged in the `access_log` file

`nickname` - specifies the name assigned to the format string. Defining the nickname is optional

The commonly used percent directives are listed in table 6.2.

Percent Directive	Description
`%{Header }i`	Specifies an incoming HTTP header in the client request
`%{Header }o`	Specifies an outgoing HTTP header in the server response
`%{Variable}e`	Specifies an environment variable as defined by the server
`%b`	Specifies the file size delivered by the server, excluding HTTP headers
`%c`	Specifies the connection status
`%D`	Specifies the time the server took to process the request
`%f`	Specifies the file path of the requested document
`%h`	Specifies the hostname of the client. If the hostname cannot be resolved, the IP address of the client is logged.
`%H`	Specifies the protocol used for the request
`%m`	Specifies the request method
`%p`	Specifies the TCP port number on which the request arrives
`%s`	Specifies the HTTP status code
`%t`	Specifies the time of the request
`%T`	Specifies the number of seconds taken by the server to process the request
`%u`	Specifies the remote user in authenticated requests
`%U`	Defines the requested URL

Table 6.2: Commonly Used Percent Directives

For example, to specify the common log format explicitly, enter the code, as shown in Code Snippet 5, in the `httpd.conf` file.

Code Snippet 5:

```
LogFormat "%h %u %t \"%r\" %>s %b"
```

In Code Snippet 5, the `LogFormat` directive enables you to log the following information:

`%h` - specifies the hostname of the client

`%u` - specifies the remote user

`%t` - specifies the time of the request

`%r` - specifies the request line received from the client

`>s` - specifies the status code returned to the client by the server

`%b` - specifies the size of the data returned to the client. The size does not include the response headers

The CustomLog Directive

The `CustomLog` directive defines the format and the filename of the log file. The `CustomLog` directive defines a new log file using the specified nickname.

The syntax of the `CustomLog` directive is as follows:

```
CustomLog file|pipe format|nickname
```

where,

`file` - specifies the location and name of the log file relative to the server root

`pipe` - specifies the path for the program that receives the input log information

`format` - specifies the log format. It can be an explicit format string

`nickname` - specifies a nickname already defined by a previous `LogFormat` directive

The code in Code Snippet 6 illustrates the use of `CustomLog` directive.

To specify a log format that includes the hostname of the client, the request time, the request and the recent status of the request, enter the code as shown in Code Snippet 6 in the `httpd.conf` file.

Concepts

Code Snippet 6:

```
LogFormat "%h %t \"%r\" %>s" common
CustomLog logs/access_log common
```

In Code Snippet 6, the `>s` denotes recent status code. The nickname `common` is assigned to the specified format string.

To specify `CustomLog` with explicit format string, enter the code, as shown in Code Snippet 7, in the `httpd.conf` file.

Code Snippet 7:

```
CustomLog logs/access_log "%h %t \"%r" %>s"
```

The result of Code Snippets 6 and 7 are the same.

- **Combined Log Format -** The Combined Log Format is the second type of commonly used format string. The format is the same as of common log format with additional fields, such as HTTP request headers, Referer, and Useragent.

The syntax of the `LogFormat` directive is as follows:

```
LogFormat format-string nickname
```

For example, to specify a combined log format for `access_log`, enter the code as shown in Code Snippet 8 in the `httpd.conf` file:

Code Snippet 8:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
combined
CustomLog log/access_log combined
```

In Code Snippet 8, `Referer` specifies the site name and `Useragent` specifies the browser used by the client to access the site. The referrer information is stored in the `referrer_log` file and the user-agent information is logged to the `agent_log` file.

To view the `access_log` file, perform the following steps:

1. Using the GUI in Linux, browse to the `/usr/local/apache2/logs` directory.

2. Open the `access _ log` file using the gedit text editor.

Figure 6.2 displays the Apache `access_log` file.

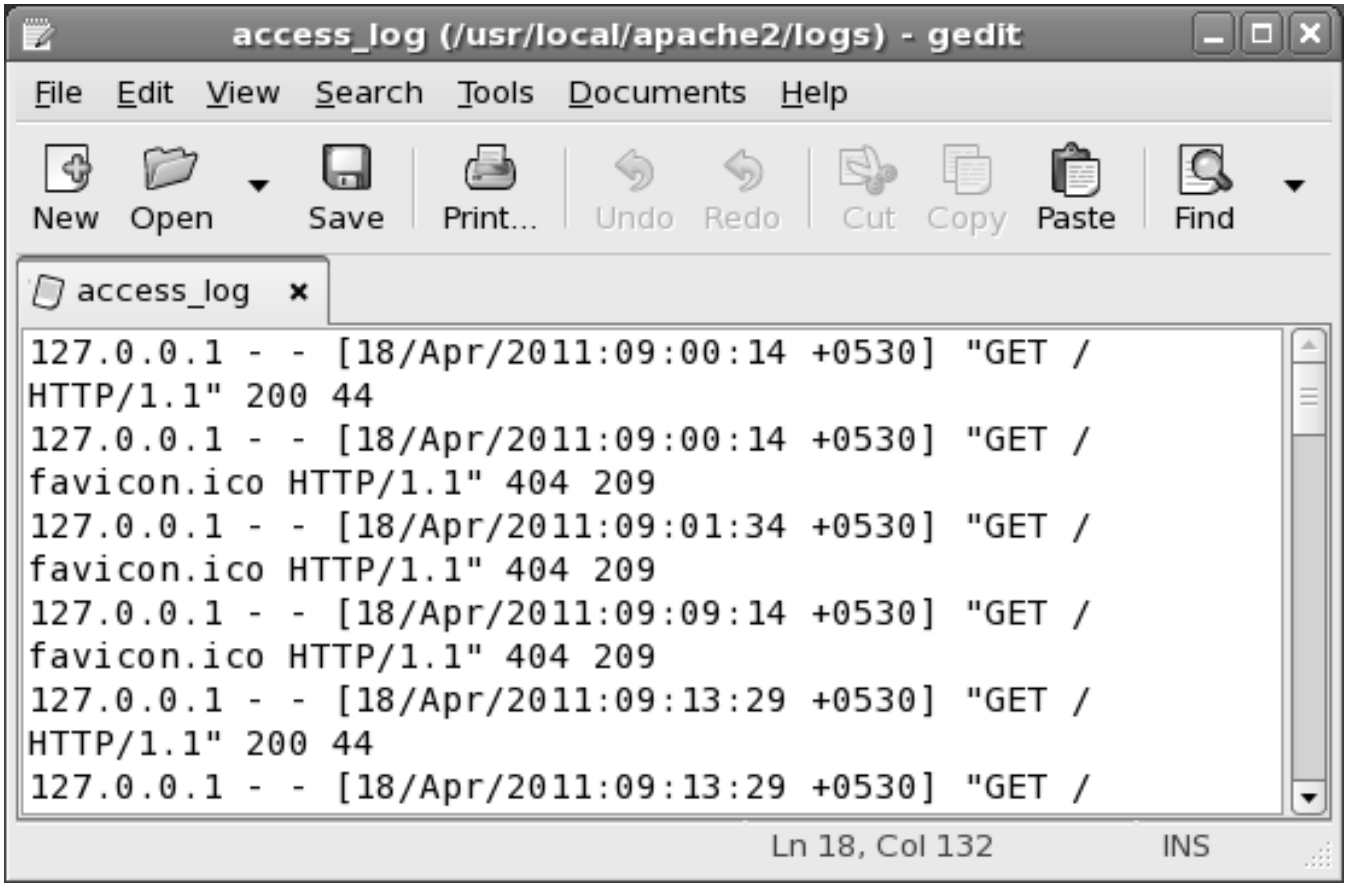


Figure 6.2: Apache access_log

In figure 6.2, the `access_log` consists of one entry per request. Each entry specifies the following:

- IP address of the client
- Date and time of the request
- Status code
- The request line method

6.5 Analyzing Logs

The Web server administrator must monitor the server activity, performance and troubleshoot errors for consistent server performance. Logs record the server activity and report any errors encountered during the server operation. HTTP servers generate different log files.

Depending on the server configuration, it may generate a list of what was accessed, who accessed it, and the browser used to access the information.

It is important to browse through the log files to analyze trends in resource utilization and identify and fix security threats. For example, the only way to identify the performance of a Web site is to generate reports based on the Web server's log file data and use the reports to determine if the problem is with the server or with the site.

There are two ways to analyze the log files. The most popular is the Analysis tools. Some of these tools include `Analog`, `Wusage`, `Getstats`, and `wwwstat`.

The second method is to import the log files into a database and then query the database to create reports. This method is less popular, more difficult to implement, but provides accurate information relating to the use of the server.

6.6 Custom Error Pages

Error messages are encountered while surfing the Internet. The reasons for these error messages include incorrect spellings, outdated links, or internal server errors. The Web server displays a generic error page when it encounters an error.

When an error page is displayed on the Web site, the Back button of the Web browser is used to navigate from the error message to different links. The number of users visiting the Web site reduces. Hence, the error page must be customized to display more user friendly and meaningful messages.

A customized error page must be created using the standard HyperText Markup Language (HTML). It must include navigational links, a search feature, and information related to the error. The `ErrorDocument` directive can be used to customize the server response to errors. This directive must be included in the `httpd.conf` file. When Apache server identifies a particular error, it responds as per the instructions specified by the `ErrorDocument` directive.

When an event or an error occurs, Apache server can be configured to perform one of the following:

- Display a simple error message
- Display a customized message
- Redirect to a local URL
- Redirect to an external URL

The syntax for the `ErrorDocument` directive is as follows:

```
ErrorDocument errorcode Action
```

where,

`errorcode` – specifies an error response code

`action` – specifies the custom error message and the location of the error document

To display an error message for unauthorized access using the `ErrorDocument` directive, enter the code as shown in Code Snippet 9, in the `httpd.conf` file.

Code Snippet 9:

```
ErrorDocument 401 /errors/401.html
```

This code in the `httpd.conf` file will display the `401.html` page whenever an authorization error occurs.

You will now design a customized error Web page to display an error message when the requested file is not found. The page will also provide additional information to view the Web page contents.

To create a custom error page, perform the following steps:

1. Create a new directory named error under the /usr/local/apache2/htdocs directory.

2. Open the gedit text editor.

3. Enter the code as shown in Example 1:

   Example 1:

```
<html>
  <head>
    <title> We’re sorry, the page you requested could not be found </
title>
  </head>
  <body>
    <font face = “arial” size=”-1”>
    <h1 align = “left”> Custom Error Page </h1>
    <hr align = “left” width=”100%”>
    <p><b> We’re sorry, but there is no page matching your request.
    </b></p>
    <p>
```

```
The page you are looking for might have been removed, had its name
changed, or is temporarily unavailable. It is also possible that you
typed the address incorrectly. </p>
    <p>You can do one of the following...</p>
    <ul>
    <li>
      Open the <a href="/"> home page </a>, and then look for links to
the information you want.
    </li>
    <li>
      Click the <A href = "javascript:history.back(1)"> Back </A> link
to try another link.
    </li>
    <li>
      If you typed the page address in the Address bar, make sure that
it is spelled correctly
    </li>
  </ul>
  </p>
    <hr align="left" width="100%">
    </font>
  </body>
</html>
```

4. Save the file as `404.html` in the `/usr/local/apache2/htdocs/error/` directory.

5. Open the `httpd.conf` file using the gedit text editor.

6. Enter the following code:

```
ErrorDocument 404 /error/404.html
```

5. Save the `httpd.conf` file and restart Apache Web server.

6. Open the **Mozilla Firefox** Web browser.

7. Access a nonexistent html page, such as:

```
http://localhost/useraccount.html
```

Figure 6.3 displays the custom error page.

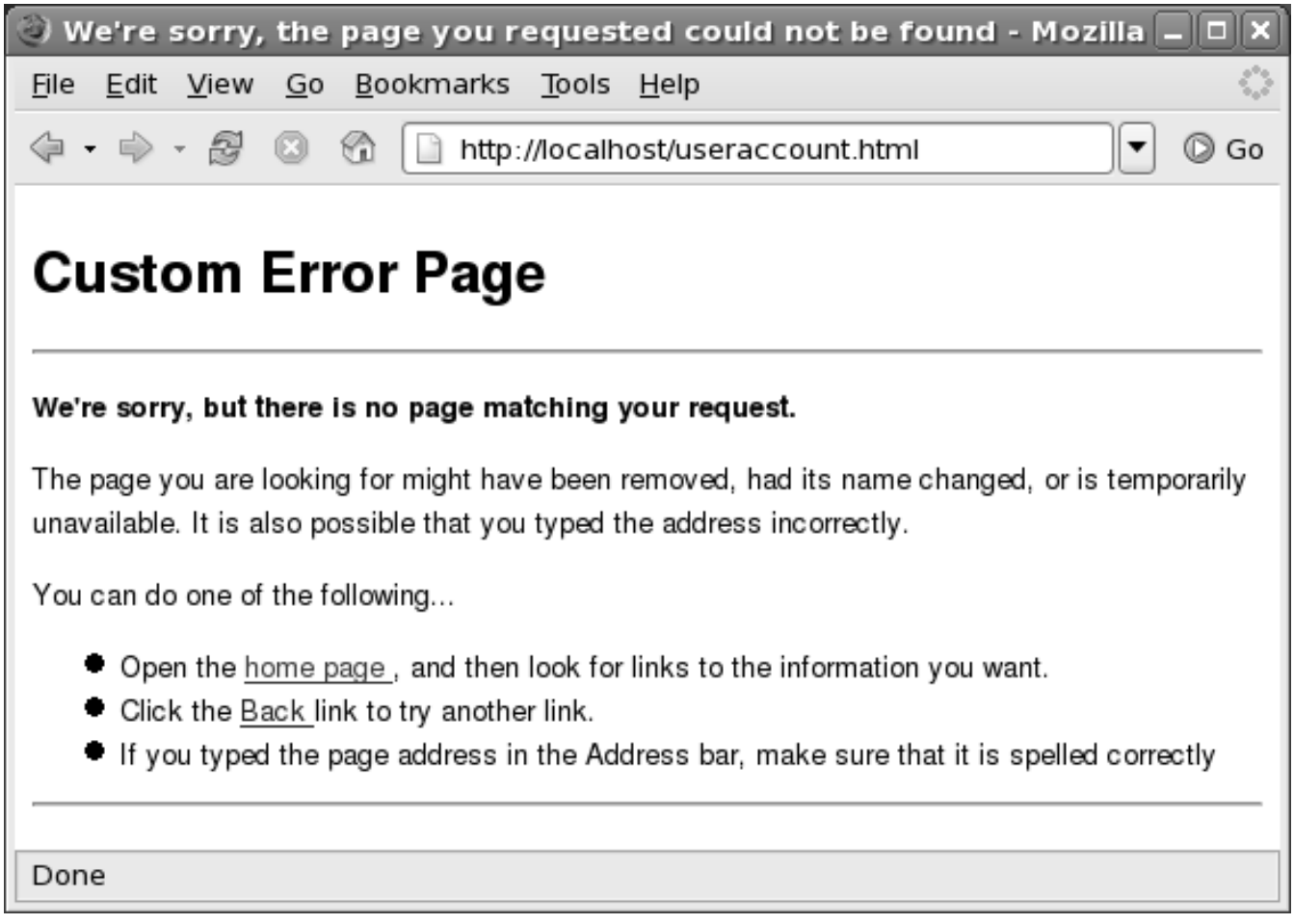


Figure 6.3: Custom Error Page

Concepts

Summary

- Logs monitor the server activity and performance and report any errors encountered during the server operation.
- The two main log files in Apache server are `error _ log` and `access _ log`.
- The `error _ log` file contains information about the errors that are generated during the operation of the server.
- The `access _ log` file records information about the requests received by the server.
- The `ErrorLog` directive specifies the location for the error log.
- The `LogLevel` directive specifies the type of error to be logged in the error log.
- The `TransferLog` directive specifies the location for the `access _ log` file.
- The `LogFormat` directive specifies the format for the `access _ log` file.
- Apache can be configured to display custom error pages when an error is encountered.
- The `ErrorDocument` directive specifies the custom error page associated with a particular error code.

Check Your Progress

1. Which of the following is the default location of the log files in Apache Web server?

   a. `/usr/local/apache2/htdocs`

   b. `/usr/local/apache2`

   c. `/usr/local/apache2/logs`

   d. `/usr/local/apache2/conf`

2. The _____ log level can be used to log alerts and emergency messages.

   a. crit

   b. emerg

   c. alert

   d. notice

3. Which option in the `LogFormat` directive specifies the remote user in authenticated requests?

   a. `%r`

   b. `%u`

   c. `%U`

   d. `%s`

4. Which of the following log files in Apache contains information about the requests made by the Web browser to the Web server?

   a. `referer_log`

   b. `access _log`

   c. `error_log`

   d. `user agent_log`

Check Your Progress

5. The _____ directive defines the location of an error page.

   a. `ErrorLog`

   b. `ErrorDocument`

   c. `CustomLog`

   d. `Transferlog`

Session

7 Apache Web Server Administration (Lab)

Lab Guide

Objectives

At the end of this session, the student will be able to:

- *Explain the process of configuring logs in Apache.*
- *Explain the process of viewing log files in Apache.*
- *Explain the process of creating a custom error page.*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Configure Logs in Apache

Apache records all information requested by the Web browser in the log files. Log files track information and performance of the server. Apache maintains two types of logs and they are as follows:

- Error log - Contains information on errors encountered during the working of the Apache server
- Access log - Records the requests processed by the Apache server

To configure Apache server for logging errors, perform the following steps:

1. **Logon to Linux as `root` in the GNOME environment.**
2. **Open the Linux Terminal.**
3. **In order to configure Apache Web server to record information in log files, you will create a directory named `mylogs` in the `/usr/local/apache2` directory. Before creating the directory, you must browse to the apache2 directory. To browse to the apache2 directory, enter the following command at the command prompt:**

```
cd /usr/local/apache2
```

Figure 7.1 displays the output of the command.

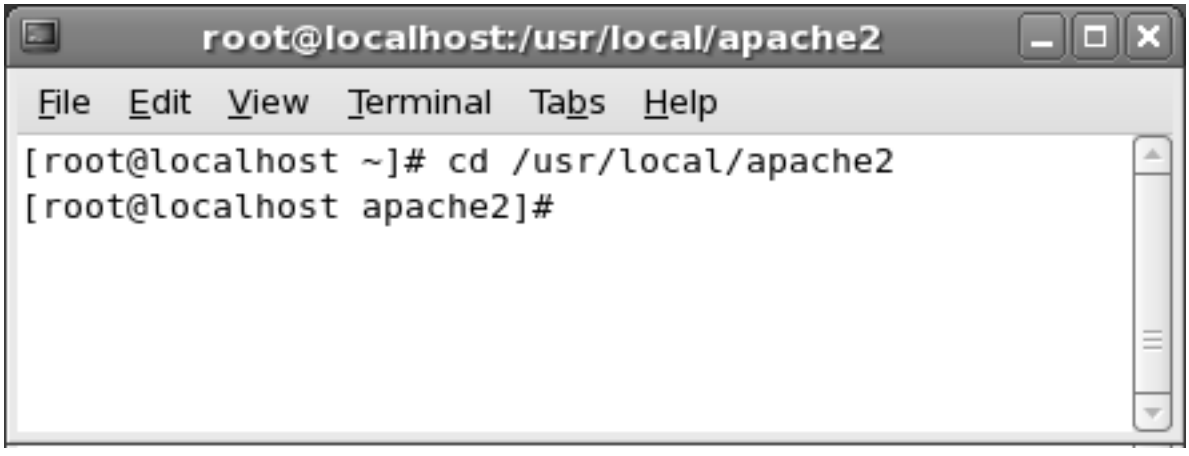


Figure 7.1: Browse to /usr/local/apache2 Directory

4. **To create the `mylogs` sub-directory in the `/usr/local/apache2` directory, enter the following command at the command prompt:**

```
mkdir mylogs
```

Figure 7.2 displays the output of the command.

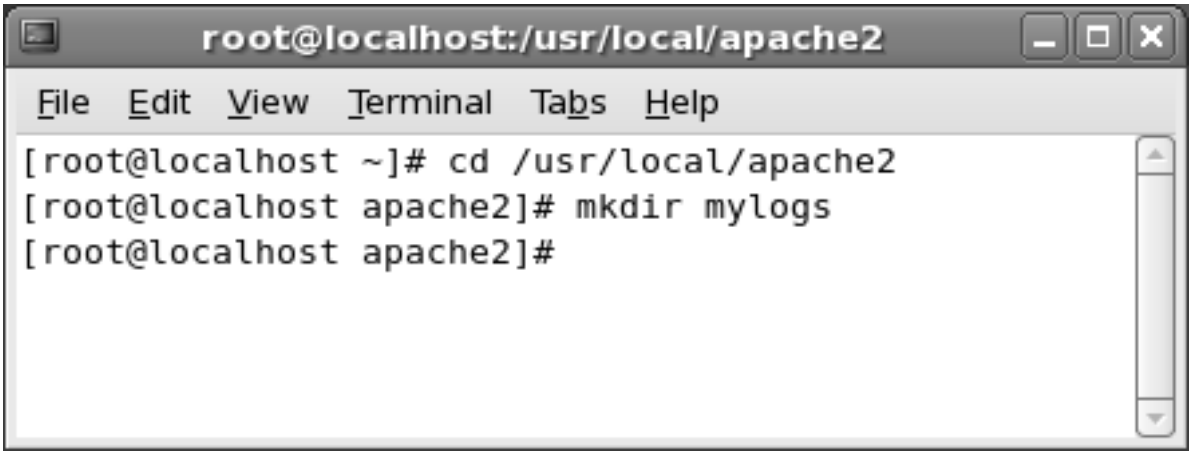


Figure 7.2: Creating the mylogs Directory

5. **To browse to the `mylogs` directory, enter the following command at the command prompt:**

```
cd mylogs
```

Figure 7.3 displays the output of the command.

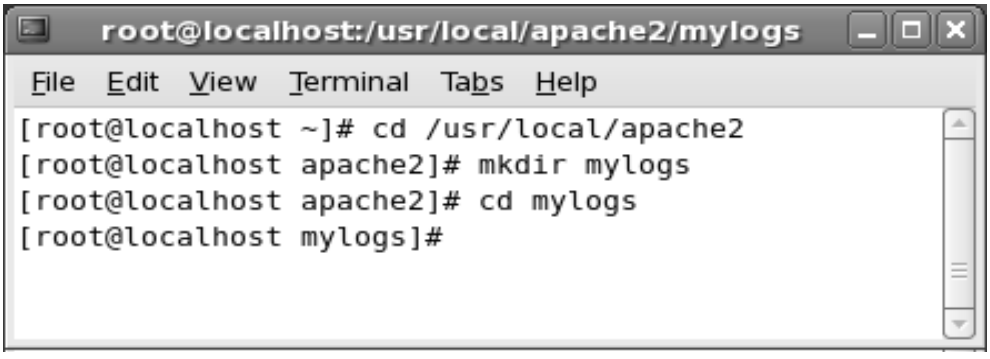


Figure 7.3: Browse to mylogs Directory

6. **To create a new file named `error_log` in the `mylogs` directory, enter the following command at the command prompt:**

```
vi error_log
```

Figure 7.4 displays the output of the command.

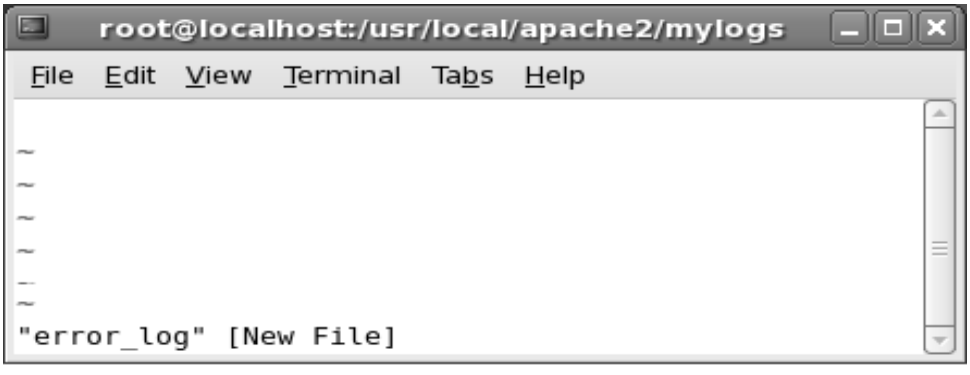


Figure 7.4: Creating error_log File

7. **Press the 'Esc' and `:wq` to exit the `vi` editor.**

8. **To browse to the conf directory, enter the following command at the command prompt:**

```
cd /usr/local/apache2/conf
```

Figure 7.5 displays the output of the command.

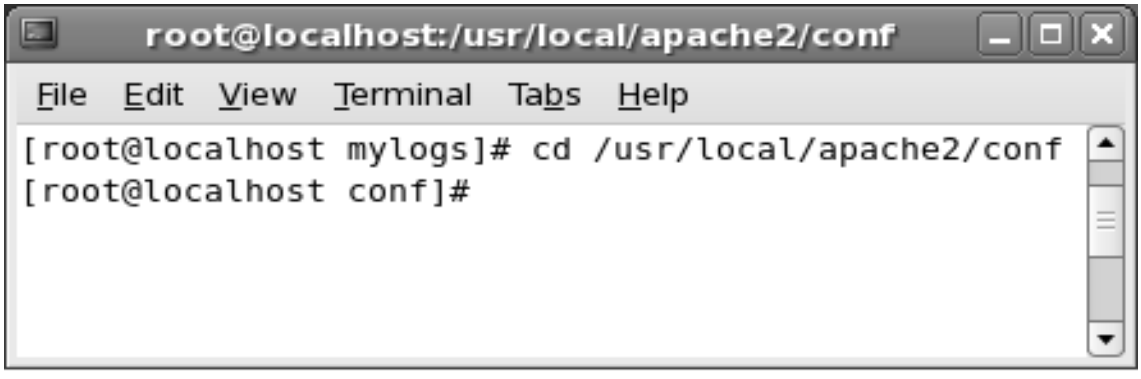


Figure 7.5: Browsing to the conf Directory

9. **To open the `httpd.conf` file in the `vi` editor, enter the following command at the command prompt:**

```
vi httpd.conf
```

Figure 7.6 displays the `httpd.conf` file.

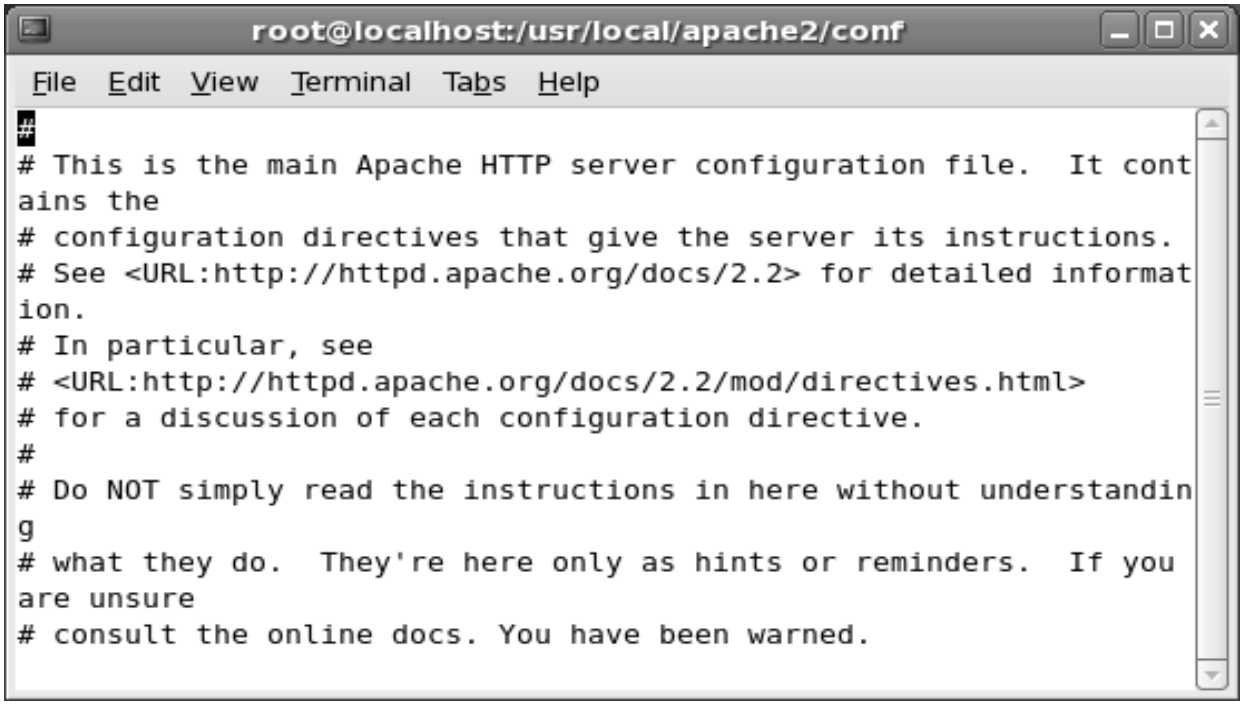


Figure 7.6: httpd.conf File

10. **Locate the `ErrorLog` directive in the `httpd.conf` file.**

11. **Press 'I' to enter into the 'Insert mode'.**

12. **Enter the following code in the `ErrorLog` directive of the `httpd.conf` file.**

```
Errorlog mylogs/error_log
```

Figure 7.7 displays the configuration of the error log directive in the `httpd.conf` file.

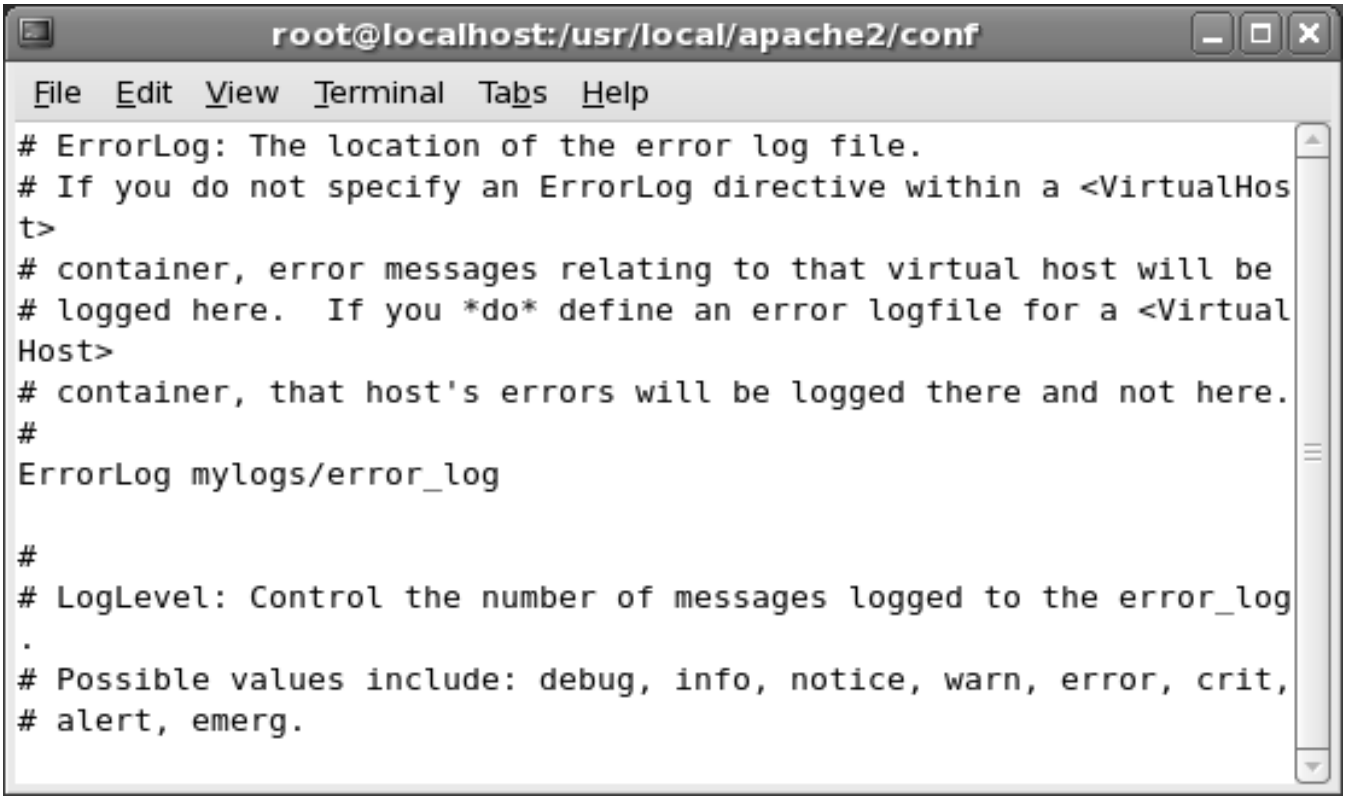


Figure 7.7: Configuring the ErrorLog Directive

The `ErrorLog` directive defines the location of the error log file to the Apache Web server. The server records the errors encountered during its operation, into the `error_log` file.

13. **To define the error level to be recorded in the error log file, enter the following code in the `LogLevel` directive of the `httpd.conf` file.**

```
LogLevel info
```

Figure 7.8 displays the `LogLevel` directive in the `httpd.conf` file.

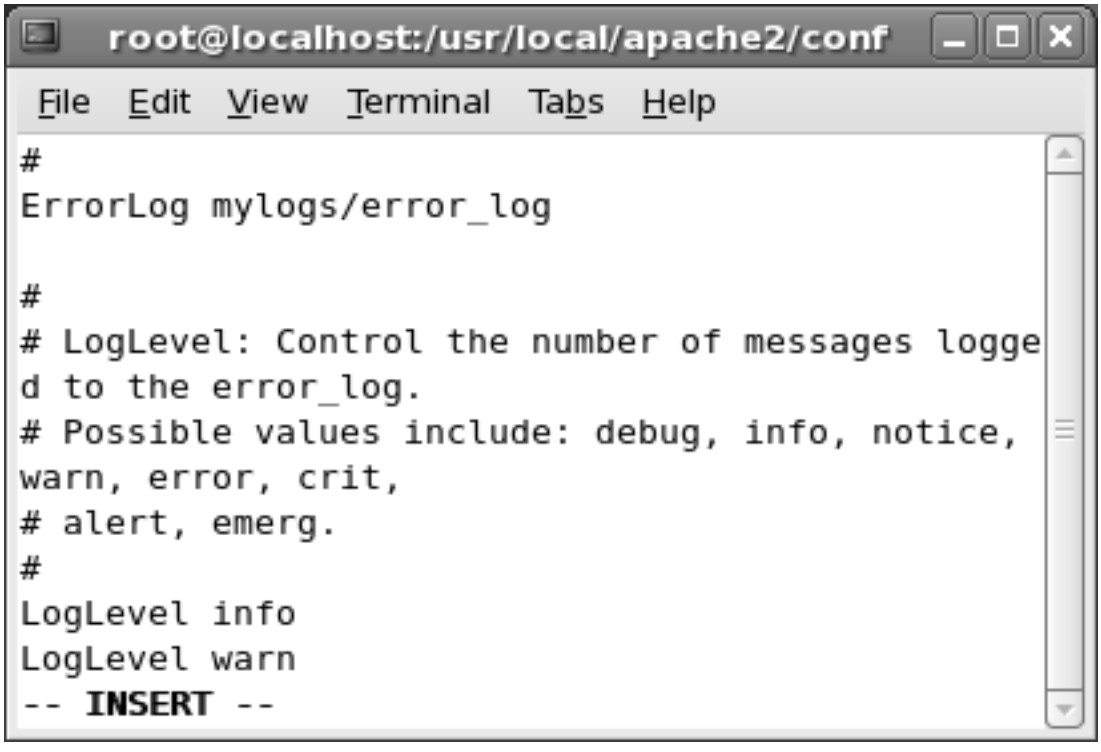


Figure 7.8: Configuring the LogLevel Directive

The `LogLevel` directive instructs the server to log informational messages along with higher-level messages into the `error_log`.

14. **Press 'Esc' and `:wq` to save changes and exit the `vi` editor.**

15. **Restart the Apache Web server.**

16. **Open the Mozilla Web browser and access the following pages:**

    `http://localhost/useraccount.html`
    `http://localhost/loginfile11.html`

The two pages `useraccount.html` and `loginfile11.html` are not present in the document `root` of the server. Hence, Apache server returns the `404` error message. This message indicates that the requested file is not found at the specified location. This information is recorded into the error log.

To view the information recorded in the `error_log`, perform the following steps:

1. **Use the File Browser application and browse to the `mylogs` directory.**
2. **Right-click the `error_log` file and select Open with Text Editor.**

Figure 7.9 displays the `error_log` file.

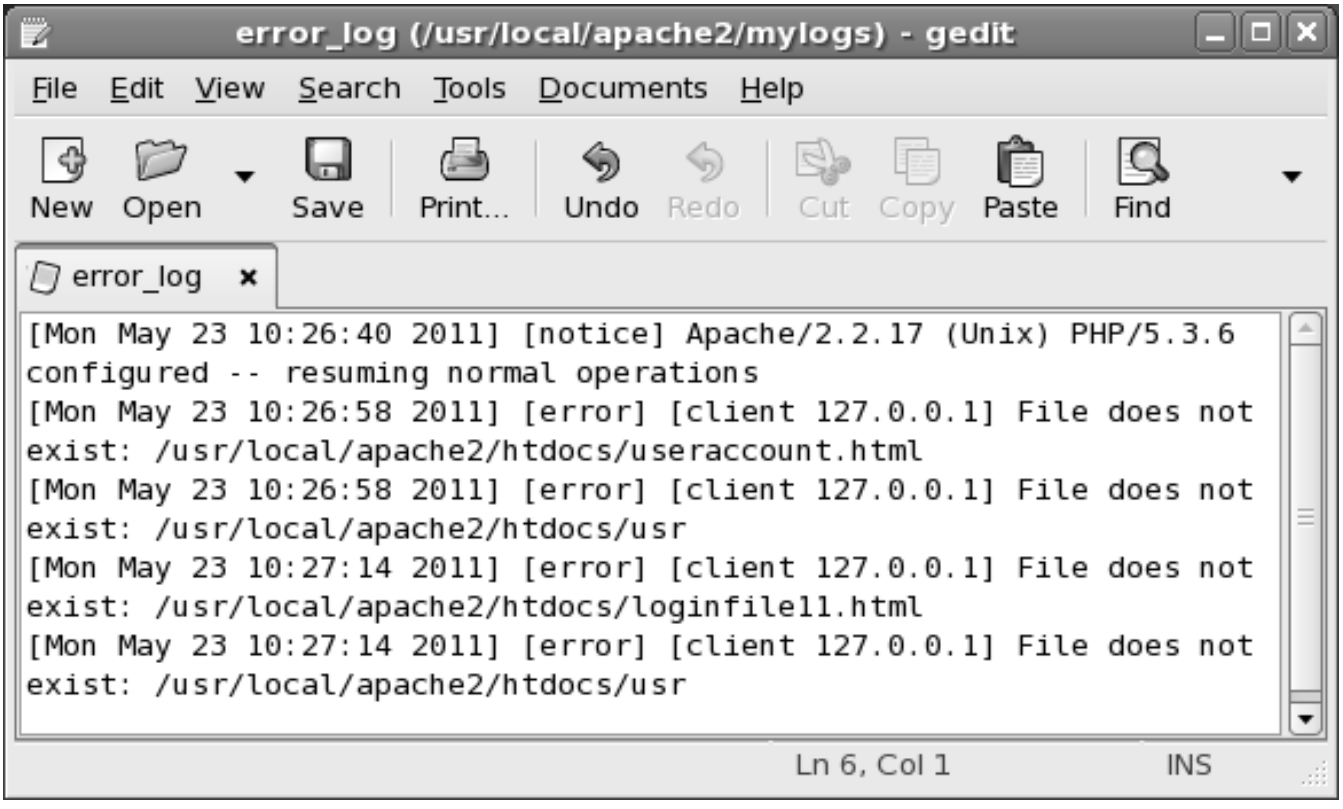


Figure 7.9: Apache error_log

The `error_log` displays the errors encountered by the server.

Configuring Apache for Access Log

The `LogFormat` and `CustomLog` directives enable you to configure Apache server to record information related to document or file access. The `CustomLog` directive enables you to record information to the log file on satisfying specified request conditions using environment variables. The `LogFormat` directive defines the format of the information to be recorded in the log file.

To configure Apache for access log, perform the following steps:

1. **Browse to the `mylogs` directory.**

2. **To create the `access_log` file, enter the following command at the command prompt:**

```
vi  access_log
```

3. **Press 'Esc' and `:wq` to save the file and exit the `vi` editor.**

4. **Similarly, create two more files, `agent_log` and `referer_log` in the `mylogs` directory using the `vi` editor.**

5. **Open the `httpd.conf` file.**

6. **Enter the following code in the `httpd.conf` file.**

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined

LogFormat "%h %l %u %t \"%r\" %>s %b" common

LogFormat "%{Referer}i → %U" referer

LogFormat "%{User-agent}i" agent
```

Figure 7.10 displays the `LogFormat` directive.

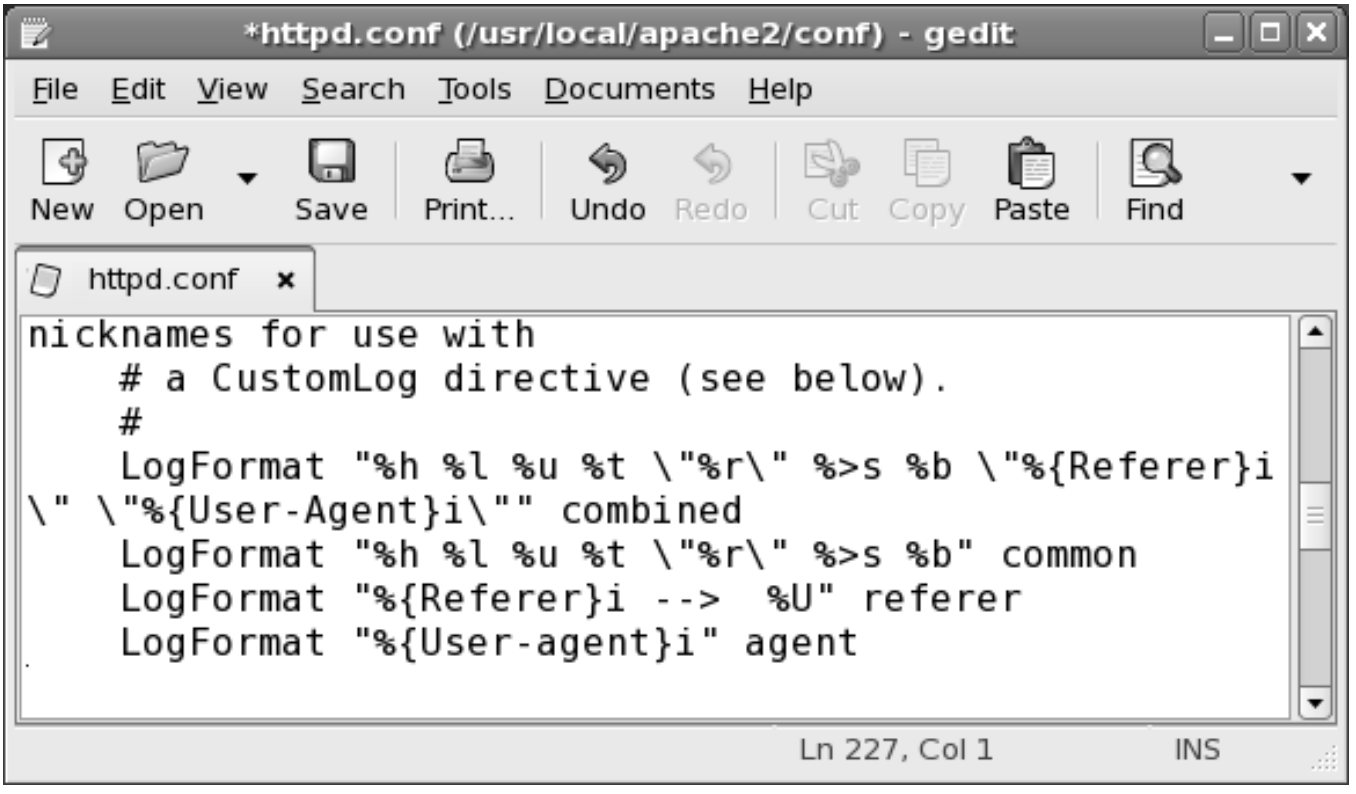


Figure 7.10: Configuring the LogFormat Directive

7. **Locate the `CustomLog` directive in the `httpd.conf` file and set the values as follows:**

```
CustomLog mylogs/access_log common

CustomLog mylogs/referer_log referer

CustomLog mylogs/agent_log agent
```

Figure 7.11 displays the `CustomLog` directive.

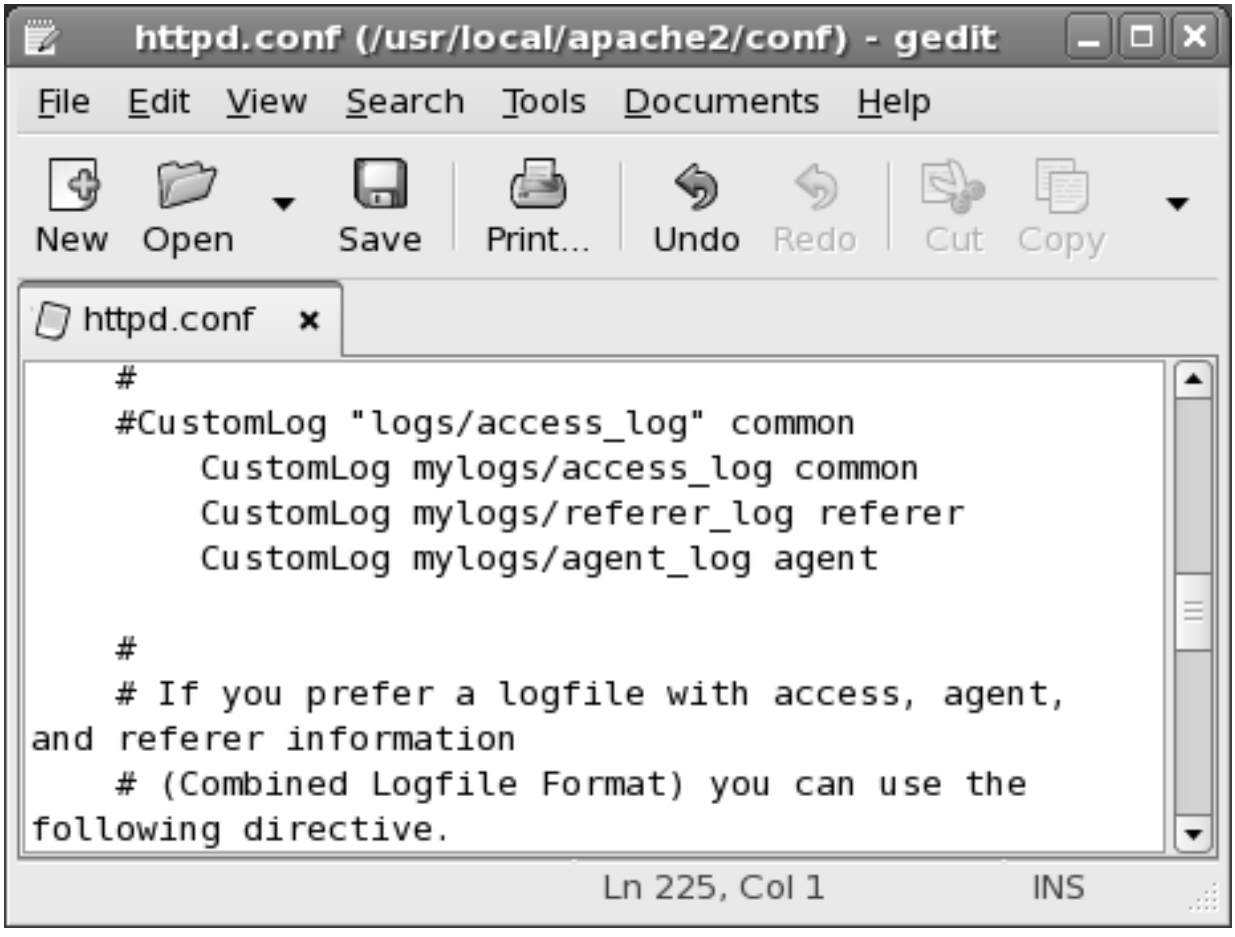


Figure 7.11: Configuring CustomLog Directive

8. **Save the changes.**

9. **Restart Apache Web server.**

10. **Open the Mozilla Firefox Web browser.**

Lab Guide

11. **Access the following Web pages:**

    `http://localhost/logfile11.html`
    `http://localhost/useraccount.html`
    `http://localhost/forum.html`

12. **Browse to the `mylogs` directory.**

13. **Right-click the `access_log` file and select Open with Text Editor.**

Figure 7.12 displays the content of the `access_log` file.

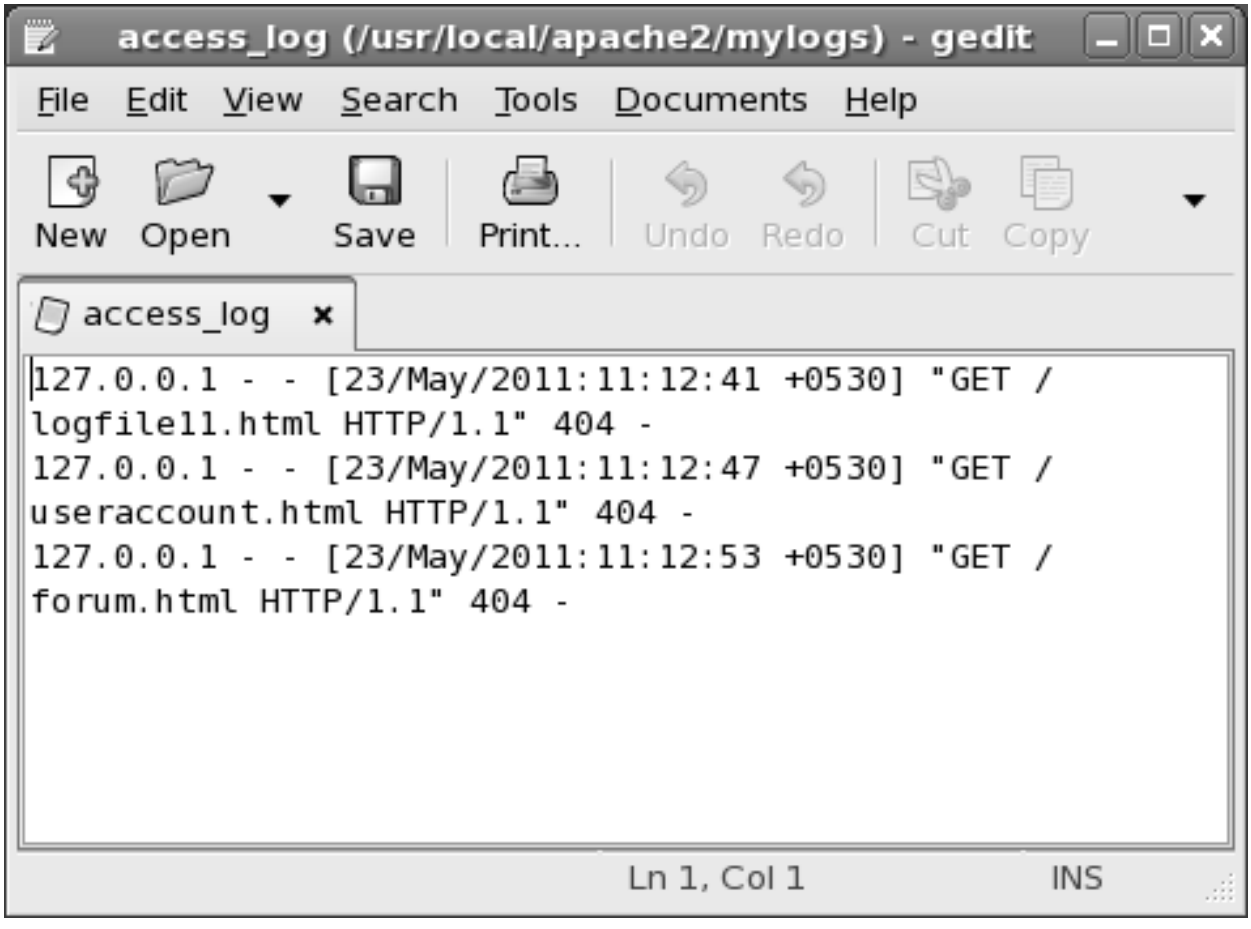


Figure 7.12: Apache access_log

The access log stores the information about the requests received by the server along with the client IP address and the time of the request.

14. **To view the `agent_log` file, browse to the `mylogs` directory and enter the following command at the command prompt:**

```
vi agent_log
```

Figure 7.13 displays the content of the `agent_log` file.

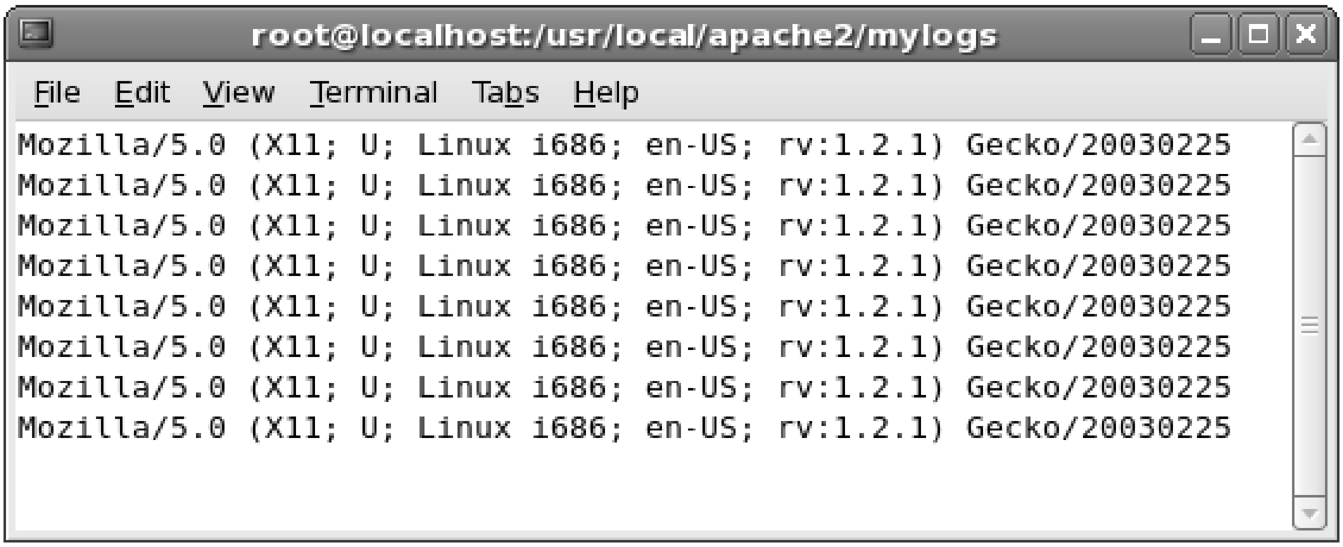


Figure 7.13: Apache agent_log

The agent log stores the information about the Web browser used for accessing pages.

15. **Press 'Esc' and `wq:` to save and exit the `vi` editor.**

16. **To view the `referrer_log` file, enter the following command at the command prompt:**

```
vi referer_log
```

Figure 7.14 displays the content of the `referer_log` file.

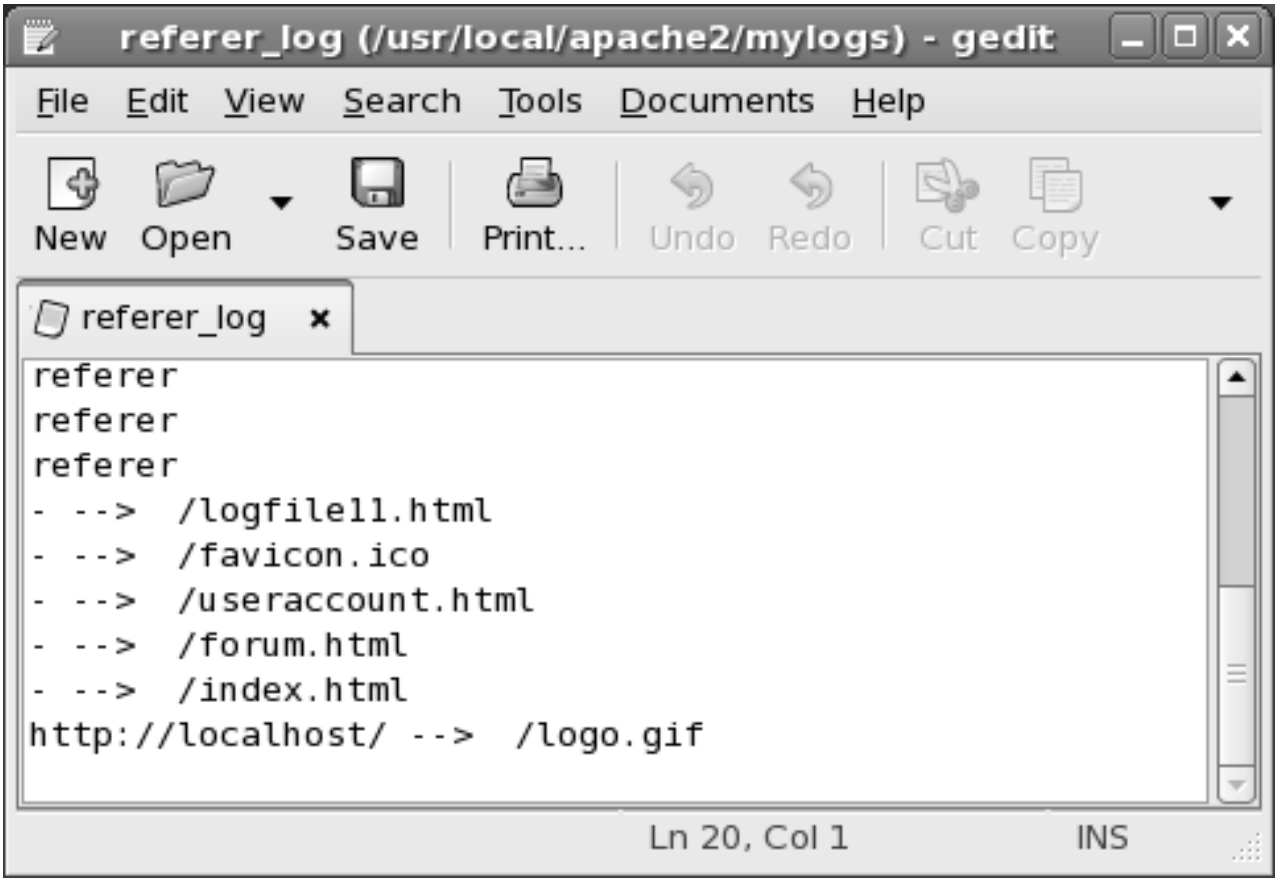


Figure 7.14: Apache referer_log

The `referer_log` file stores information about the page that the client refers for the request.

Creating a Custom Error Page

The `ErrorDocument` directive customizes the response of the server to the errors that occur. A custom error page is created using the html tags. It includes navigational links, search feature and information related to the error that occurred.

To create a Custom Error Page, perform the following steps:

1. **Create a new directory named error under the `/usr/local/apache2/htdocs` directory.**
2. **Create a new file using the gedit text editor.**

3. **Enter the following HTML code:**

```
<html>
 <head>
    <title> We're sorry, the page you requested could not be found
</title>
     </head>
 <body>
    <font face = "arial" size="-1">
        <h1 align = "left"> Mysite.com </h1>
        <hr align = "left" width="100%">
 <p> <b> We're sorry, but there is no page matching your request.</b>
</p>
    <p>
The page you are looking for might have been removed, had its name
changed, or is temporarily unavailable. It is also possible that you
typed the address incorrectly. </p>
        <p>You can do one of the following...</p>
        <ul>
            <li>
        Open the <a href="/"> home page </a>, and then look for links
to the information you want.
                    </li>
            <li>
        Click the<A href = "javascript:history.back(1)"> Back </A>
       link to try another link.
            </li>
            <li>
 If you typed the page address in the Address bar, make sure that it is
spelled correctly
                    </li>
            </ul>
        </p>
        <hr align="left" width="100%">
        <p>
            &copy;  Mysite.com  - 2004
               </p>
    </font>
</body>
</html>
```

Figure 7.15 displays the HTML code for the error page.

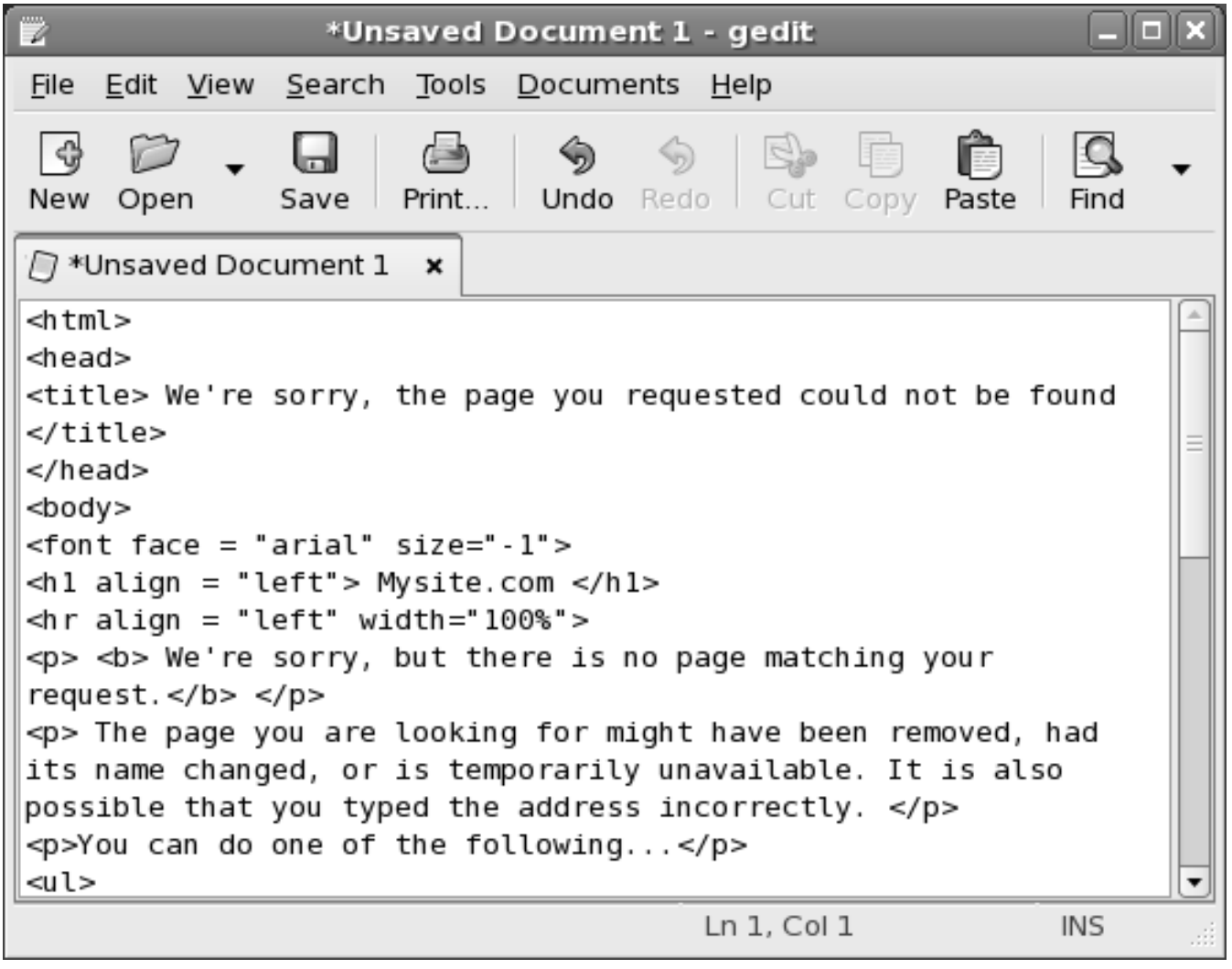


Figure 7.15: HTML code for Custom Error Page

4. **Save the file as `404.html` in the `/usr/local/apache2/htdocs/error` directory.**

5. **Open the `httpd.conf` file and enter the following code:**

```
ErrorDocument 404 /error/404.html
```

6. **Save the file and restart Apache Web server.**

7. **Open the Mozilla Firefox Web browser and access a nonexistent html page such as:**

```
http://localhost/useraccount.html
```

Figure 7.16 displays the Custom Error Page.

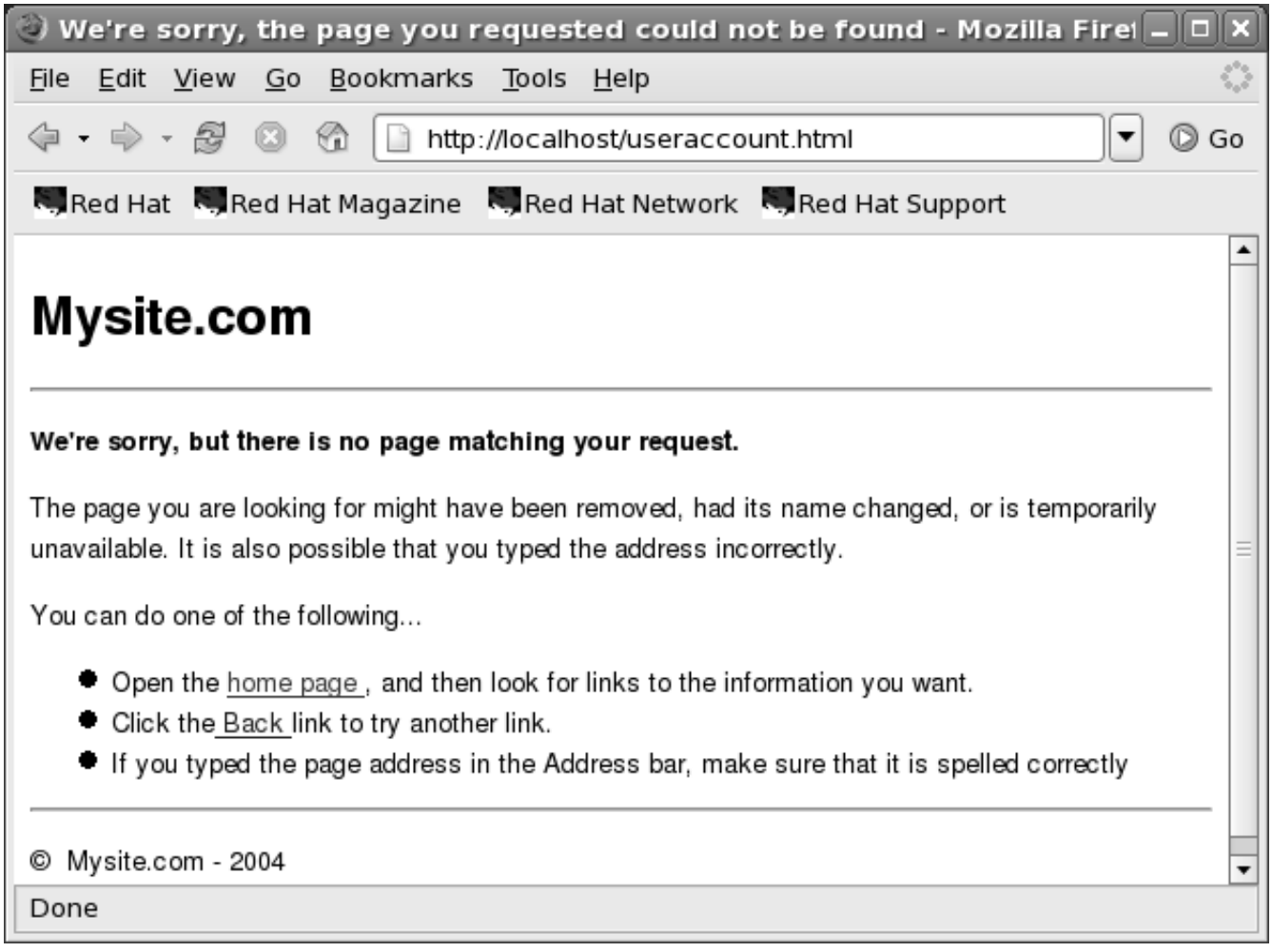


Figure 7.16: Custom Error Page

Do It Yourself

1. Configure the Apache to log critical errors.
2. Configure the access log to contain the following:
    a. The ip address of the client
    b. The date and time of the request
    c. The status code
3. Create a custom error page for error code 400 (bad request).

Onlinevarsity

your e-way to learning

TechnoWise

Are you a

TECHIE GEEK

looking for updates?

Logon to

www.onlinevarsity.com

Session

8 Security

Concepts

Objectives

At the end of this session, the student will be able to:

- *Describe security threats.*
- *Explain security in Apache Web server.*
- *Explain the process of authenticating a user and a host.*
- *Explain the use of a proxy.*
- *Explain the use of a firewall.*

8.1 Introduction

Security refers to controlling access to documents or Web sites. It involves assigning passwords that gives user access to a specific document, program, or system. Security can also be implemented using firewalls that prevent unauthorized access to the network.

In this session, you will learn to identify security threats, implement security in Apache Web server, and create user and host based authentication. In addition, you will learn to define firewall and proxies, configure firewall, and explain the firewall architecture models.

8.2 Security Threats

Security is an important issue while using Internet Web servers. The main objective of implementing Internet security is to protect the Web server from external attacks. It enables access based on the username and the password.

The following reasons explain the need to implement security in a Web server:

- **Insecure Architectures** - enables access to unauthorized users.
- **Broadcast Networks** - allows intruders and unauthorized users on local nodes. Hardware such as, hubs and routers are based on broadcast or non-switched principle. This principle states that when a node transmits data across the network to a recipient node, the hub or router sends a broadcast of the data packets until the recipient node receives and processes the data.
- **Centralized Servers** - allows a single point of access on the network.

The data in a centralized server can be manipulated. In such instances, a central server allows access to the entire network.

- **Unused Services and Open Ports** - enables access through ports and services created by default while installing the software or operating system. These services increase traffic on the server and workstation.

- **Inattentive Administration** - enables access when users do not monitor their servers and workstations, or use the default password. In such cases, hackers can break-in and misuse data in the server.

- **Inherently Insecure Services** - allows encryption of information over the network. Encryption requires a username and a password for authentication. For example, Telnet and FTP require a username and a password to authenticate the user. Packet switching software can easily hack these usernames and passwords.

- **Bad Passwords** - enable easy access to the system.

- **Vulnerable Client Applications** - allows network access for remote users. For example, if the server provides Telnet or FTP services on a public network, an attacker can capture the plain text usernames and passwords when they are transmitted over the network.

8.3 Security in Apache Web Server

An improperly configured Apache Web server contains security flaws. When the `root` login in Linux is used to start Apache server, it binds itself to port 80.

If you do not configure the user and the group, then all Apache server processes will run with root privileges. Running an Apache server with the root privileges can affect the server connected to the Internet. A badly written script allows a hacker, complete access to the server.

User Authentication requires a username and a password for enabling access. Apache server authenticates a HTTP request in three ways:

- **Access Control** - means controlling access to a Web resource. It restricts access to incoming HTTP request. Apache server controls access to a Web resource. Access can be granted or denied based on different conditions, such as the network address of the client.

- **Authentication** - is the process of verifying the correctness of the username and the password. It verifies a user by comparing its username and password from a list of known usernames and passwords. .

- **Authorization** - allows access to a resource protected by authentication. An authorized user is a valid user, and has permission to access the resource.

Concepts

The module `mod_auth` implements user authentication using text files. The module `mod_access` enables you to implement access control. The access control is based on the characteristics of a client request such as, the client hostname or the IP address.

The core module supports the authentication directives. Some of the directives that enable authentication in Apache server are as follows:

- **AuthName Directive** - The `AuthName` directive specifies the name of authorization realm for a directory, The authorization realm is the protected area of the Web site, that is sent to the client browser to allow the user to know which username and password to forward. The authentication modules in Apache server require this information to identify the restricted resource. By default, the `AuthName` directive accepts a single argument. However, if the argument contains a space, then it should be enclosed in double quotes. This directive depends on the `AuthType`, `Require`, `AuthUserFile`, and `AuthGroupFile` directives.

  The syntax for the `AuthName` directive is as follows:

```
AuthName auth-domain
```

  where,

  `AuthName` - specifies the name of the authentication information

  `auth-domain` - specifies the argument for the directive

  For example, to set the authentication realm to 'Security', enter the code, as shown in Code Snippet 1 in `httpd.conf` file.

  Code Snippet 1:

```
AuthName "Security"
```

  The directive in Code Snippet 1 displays 'Security' in the password dialog box on the Web browser.

- **AuthType Directive** - The `AuthType` directive defines the method to authenticate users for a specific directory. The authentication methods that are available are `Basic` and `Digest`. The `Basic` method requires authentication and authorization with a username and a password. The `Digest` method of authentication requires a password file to verify the username and the password to enable access to a resource. In digest authentication, the password is not transmitted across the network. Instead, an MD5 (Message-Digest) digest of the password is transmitted. This ensures the security and the integrity of the password.

The syntax for the AuthType directive is as follows:

```
AuthType Basic|Digest
```

- **Require Directive** - The Require directive specifies the authorized users who are allowed access to a resource. Apache server uses the authorization modules, such as mod_authz_user and mod_authz_groupfile to process access permissions.

  The syntax for the Require directive is as follows:

```
Require entity-name [entity name]
```

  where,

  Require - specifies access permissions

  entity-name - specifies the users who have access to the resource

  Table 8.1 lists the options of the Require directive.

Syntax	Permissions
Require user userid [userid]...	Allows the specified users to access the resource
Require group group-name [group-name] ...	Allows only the users mentioned in the named groups to access the resource
Require valid-user	All valid users can access the resource

Table 8.1: Options for the Require Directive

Note: You must configure the Require directive with directives, such as AuthName, AuthType, AuthUserFile, and AuthGroupFile.

- **AuthUserFile Directive** - The AuthUserFile directive defines name of the text file that contain username and password for user authentication. This file stores a list of usernames and corresponding passwords. This file is also known as the password file. Every line of the user file contains a username followed by a colon, and the encrypted password. If the user ID is included multiple times, mod_authn_file will use the first occurrence to verify the password.

  The syntax of the AuthUserFile directive is as follows:

```
AuthUserFile file-path
```

  where,

  file-path - specifies the path of the user file

Concepts

Note: Apache uses the path for the user file specified in the `ServerRoot` directive, if the file-path location is not specified in the `AuthUserFile` directive.

For example, to set the path to the password file named `passwd`, enter the code, as shown in Code Snippet 2, in `httpd.conf` file.

Code Snippet 2:

```
AuthUserFile /usr/local/apache2/passwords/passwd
```

- **AuthGroupFile Directive** - The `AuthGroupFile` directive defines the name of the text file that stores a list of user groups. Every line in the text file contains a group name followed by a colon and the usernames separated by spaces.

The syntax of `AuthGroupFile` directive is as follows:

```
AuthGroupFile file-path
```

where,

`file-path` - defines the path of the text file containing the user groups

For example, to define the path of the text file, enter the code, as shown in Code Snippet 3 in the `httpd.conf` file.

Code Snippet 3:

```
AuthGroupFile /usr/local/apache2/passwords/mygroups
```

8.3.1 Using Authentication Directives in .htaccess file

A `.htaccess` file contains one or more configuration directives. The directives in this file apply to the directory and all its sub-directories in which you save the file. It makes changes in the configuration of the server on per-directory basis.

The configuration directives included in `.htaccess` file are applicable to the directory and the subdirectories where the `.htaccess` file is located. Directives are applied to the directories in the order that they are found. Therefore, a `.htaccess` file in a particular directory can override directives from `.htaccess` file located higher up in the directory tree. By default, Apache server does not permit authentication directives in the `.htaccess` file as a security measure. To permit authentication directives in the `.htaccess` file, you must ensure that they are enabled.

For example, to enable authentication directives for a specific directory, enter the code, as shown in Code Snippet 4, in the `Directory` section of the configuration file, `httpd.conf`.

Code Snippet 4:

```
<Directory /usr/local/apache2/mydocs>
AllowOverride +AuthConfig
</Directory>
```

In the code, the configuration enables the use of all authentication directives.

8.3.2 Basic Authentication

Basic authentication is a simple and commonly used method for authenticating users. When a client requests a document that is protected using basic authentication, Apache sends a 401 Authentication Required header along with the name of the realm set by `AuthName` directive.

For example,

```
WWW-Authenticate: BASIC realm="File Security"
```

If the client browser supports basic authentication, it prompts the user for a username and a password. It repeats the request to the server, and adds the username, and password of the client in the Authentication header. For example, if the user enters a username as John, and password as pass123, the browser generates a new request including the Authorization header.

```
Authorization: BASIC t90jhismlvghdftHLkpw=
```

The string `t90jhismlvghdftHLkpw=` in the example evaluates to John: pass123.

To configure Apache server for Basic file authentication, enter the code, as shown in Code Snippet 5, in the `Location` section of the configuration file, `httpd.conf`.

Code Snippet 5:

```
<Location  /myfiles>
  AuthName "Protected Documents"
  AuthType Basic
  AuthUserFile /usr/local/apache2/passwords/passwdfile
  AuthGroupFile /usr/local/apache2/passwords/grpfile
  Require valid-user
</Location>
```

Concepts

In the code, Apache Web server is configured to verify the account credentials before permitting access to the `/myfiles` directory. The Apache Web server will use the password file named `passwd` as specified in the `AutheUserFile` directive.

Note: The explanation to generate a password file is described later in the session.

This file is used to compare and verify the username and the password for authentication. The group name for authentication is derived from the `AuthGroupFile` directive.

8.3.3 Digest Authentication

Digest authentication is an alternate method for protecting the Web content. It improves the security of passwords. This method uses the module `mod_auth_digest` to implement digest authentication. The module transmits a digest of the password instead of the password. Table 8.2 lists some of the directives used in Digest Authentication.

Directive	Function
`AuthDigestProvider`	Specifies the authentication provider for the location.
`AuthDigestAlgorithm`	Specifies the algorithm to use for calculating the challenge and response hash. Challenge hash is used to encrypt the authentication while response hash is used to verify or decrypt the authentication.
`AuthDigestShmemSize`	Defines the memory space to be allocated at startup. Apache server uses this memory space to track client activities.

Table 8.2: Directives used for Digest Authentication

8.4 Creating and Maintaining Password Files

Apache uses password files to verify the authenticity of the username and the password. This file must be stored outside the document directory because it contains sensitive information. The `htpasswd` utility is used to create and update the flat-files to store usernames and password for basic authentication. The utility can be used to encrypt and display password information for use in other types of data stores. If the utility cannot generate the password file, it returns an error message.

A password file is required to restrict access to a file using a password. This password file must be stored at a location that is inaccessible from the Internet so that an intruder cannot download it. The `htpasswd` utility is located in the `/usr/local/apache2/bin` directory by default.

The syntax for the `htpasswd` command is as follows:

```
htpasswd option passwdfile username
```

where,

`option` - specifies a list of available options. Table 8.3 lists the options of the `htpasswd` utility.

Option	Description
`-b`	obtains the password directly from the command line instead of prompting for it
`-c`	creates a password file and if the file already exists, the file gets overwritten and truncated
`-m`	uses MD5 to encrypt the passwords
`-d`	uses crypt() to encrypt the passwords
`-s`	uses SHA encryption
`-p`	uses plaintext format to encrypt passwords

Table 8.3: htpasswd Utility Options

`passwdfile` - specifies the name of the password file that stores the username and password

`username` - specifies the username to be created or updated in the `passwdfile`. The username is added to this file, if it does not exist. If it exists, the password is changed.

For example, to create a password file for the user john, enter the following command at the command prompt:

```
/usr/local/apache2/bin/htpasswd -c /usr/local/apache2/passwords john
```

Figure 8.1 displays the output of the command.

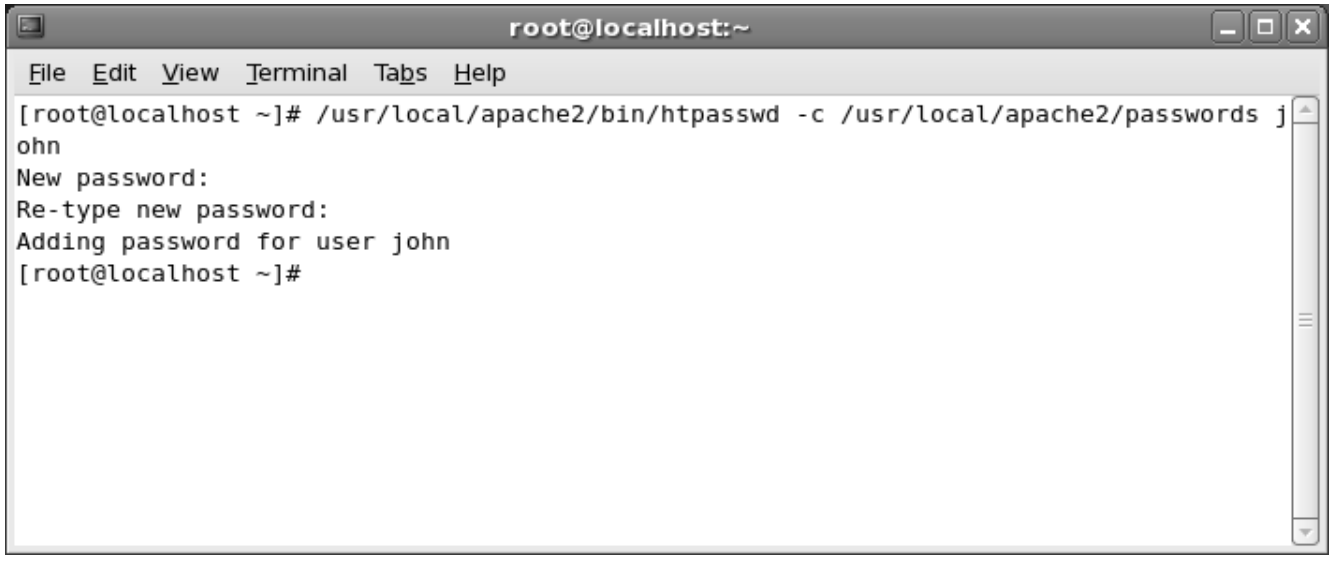


Figure 8.1: Creating a Password File

Concepts

In figure 8.1, the `-c` argument is used to create a new password file. The `htpasswd` utility prompts for a password at execution. After creating the password file, Apache server must be configured to request a password from the user. You must also include the names of the users who are provided access permission. You must specify these settings in either `httpd.conf` or in `.htaccess` file.

8.5 Working with Proxies

A proxy server acts as an intermediate server between a client browser and the HTTP server. The main objective of a proxy server is to filter client requests. It intercepts all the client requests and fulfills them if the resource requested by the client exists on the proxy. If the proxy is unable to fulfill the client request, it forwards the request to the remote HTTP server. You can configure Apache server to function as a proxy server.

8.5.1 Installing and Enabling Proxy Services

The proxy functionality of Apache server is stored in the `mod_proxy` module. It implements full support for the `HTTP/1.1` proxy protocol. There are four basic modules of proxy:

- **`mod _ proxy`** - specifies the proxy or the gateway for Apache server
- **`mod _ proxy _ connect`** - supports the `CONNECT HTTP` method. Apache server uses this module to pass SSL request through proxy servers
- **`mod _ proxy _ ftp`** - supports to proxy FTP requests using the `GET` method only
- **`mod _ proxy _ http`** - enables proxy for HTTP and HTTPS requests

The `mod_proxy` includes these protocol modules by default when Apache Web server is configured during configuration.

Some of the directives that enable you to install proxy in Apache are as follows:

- **<Proxy> Directive** - The Proxy directive encloses all the directives that apply only to the matching filtered content. Shell-style wildcards can be used to filter requests.

  The syntax for `<Proxy>` directive is as follows:

```
<Proxy wildcard-url> … </Proxy>
```

  For example, to allow access from a specific host, enter code, as shown in Code Snippet 6, in the configuration file, `httpd.conf`.

Code Snippet 6:

```
<Proxy *>
  Order Deny, Allow
  Deny from all
  Allow from mynetwork.myserver.com
</Proxy>
```

In the code, the `<Proxy>` directive matches the host to all matching proxy content. The `Order` directive initiates the two directives, `Deny` and `Allow`. The `Allow` directive permits access only to the specified IP addresses while the `Deny` directive restricts access to specific IP addresses.

The example allows access to the host only from `mynetwork.myserver.com` and prohibits access from remaining hosts.

- **ProxyRequests Directive -** The `ProxyRequests` directive enables or disables Apache server to function as a forward proxy server. A forward proxy server redirects the requests to the HTTP server. The syntax of the `ProxyRequests` directive is as follows:

```
ProxyRequests On|Off
```

The default value of the `ProxyRequests` directive is set to `Off`.

- **ProxyIOBufferSize Directive** - The `ProxyIOBufferSize` directive specifies the size of the internal data buffer used for temporary storage of data.

The syntax of the `ProxyIOBufferSize` directive is as follows:

```
ProxyIOBufferSize bytes
```

For example, to set the buffer size to 8100, enter the code as shown in Code Snippet 7, in the Proxy section of the `httpd.conf` file.

Code Snippet 7:

```
ProxyIOBufferSize 8100
```

Note: The default value of the buffer size is 8192.

- **ProxyDomain Directive -** The `ProxyDomain` directive specifies the default domain to which Apache proxy server belongs within an intranet. If the server receives a request to a host without a domain name, it will generate a redirection response to the same host along with the configured domain.

The syntax for the `ProxyDomain` directive is as follows:

```
ProxyDomain Domain
```

For example, to specify a default domain name, enter the code as shown in Code Snippet 8, in the Proxy section of the `httpd.conf` file.

Code Snippet 8:

```
ProxyDomain .MyServer.com
```

- **ProxyBlock Directive -** The `ProxyBlock` directive blocks HTTP, HTTPS, and FTP document requests to sites whose names contain matched words, hosts and/or domains specified in this directive. This directive specifies a list of these words, hosts, and/or domains, separated by spaces. It determines the IP addresses of the items listed, which may be host name at startup, and cache them for the matching test. This process will result in an increased startup time of Apache Web server.

  The syntax for `ProxyBlock` directive is as follows:

```
ProxyBlock *|word|host|domain [word|host|domain] …
```

  For example, to block request for a specific domain, enter the code as shown in Code Snippet 9, in the `httpd.conf` file.

  Code Snippet 9:

```
ProxyBlock myserver.com yourcompany.com
```

  The example blocks the Web sites `myserver.com`, and `yourcompany.com`.

- **ProxyMaxForwards Directive -** The `ProxyMaxForwards` directive defines the maximum number of proxies through which Apache server can pass through a client request. Apache server will redirect the client request if the request does not contain the `Max-Forwards` header.

  The syntax of `ProxyMaxForwards` directive is as follows:

```
ProxyMaxForwards number
```

  For example, to set the maximum number of proxies to 20, enter the code as shown in Code Snippet 10, in the Proxy section of the `httpd.conf` file.

Code Snippet 10:

```
ProxyMaxForwards 20
```

Note: The default value of the directive is -1.

8.5.2 Configuring Apache as a Proxy Server

You can configure Apache server as a proxy server for FTP or HTTP services. Apache server can be configured for two types of proxies:

- **Forward Proxy** - specifies an intermediate machine between the client and a remote server. A forward proxy server receives the request from a client and forwards it to the remote server. It is also capable of caching data. This feature reduces load on the network between the proxy and the remote server. The Forward proxy is used for the following reasons:
  - **Security** - provides an authenticated gateway through firewalls to prevent direct Internet access to the clients
  - **Caching** - logs frequently accessed sites, graphics, and other elements when accessed to the Internet
  - **Filtering** - enables access permissions to the clients for sites and information
- **Reverse Proxy** - enables controlled access from the Internet to the servers behind a firewall. Client Web browsers interpret a reverse proxy like a regular Web server and do not require additional configuration. The client requests for content in the namespace of the reverse proxy and then the reverse proxy processes the request and returns the content similar to a Web server.

  Uses of reverse proxy:
  - Provides Internet access to a server behind the firewall
  - Used for load balancing among backend servers
  - Caches for a time-consuming backend server
  - Gathers several servers in the same URL space

To configure Apache as a proxy server, enter the directives as shown in Code Snippet 11, in the `httpd.conf` file.

Concepts

Code Snippet 11:

```
Listen 8080
User  httpd
Group httpd
ProxyRequests On
```

You must control the access to proxy after configuration.

For example, to secure proxy servers, enter the code, as shown in Code Snippet 12, in the `httpd.conf` file.

Code Snippet 12:

```
<Proxy 192.127.12.10>
  Order Deny, Allow
  Deny from all
  Allow from 127.0.0.1
</Proxy>
```

The example allows access to the proxy from 127.0.0.1.

8.6 Working with Firewalls

Firewalls provide protection to networks against outside attackers by protecting the network from unnecessary Internet traffic. Apache server configures firewalls to block data from certain locations and allow only relevant and necessary data. This is important for those using cables or DSL modems. They can analyze multiple packets and incoming protocols and perform conditional evaluations.

The two types of firewalls are as follows:

- **Network-Level Firewalls: Packet Filters** - provides routers with packet filtering capability. Router is a device used to create a permanent Internet connection. They work by controlling traffic based on the IP address, blocking data packets based on destination address or port information in the packet's header.

- **Application-Proxy Firewalls/Application Gateways** - establishes connection between outside clients and internal network. The IP packets are not forwarded. Instead, a type of translation occurs where the gateway acts as an interpreter and filters IP packets.

8.6.1 Configuring Firewall

Firewalls exist between the computer and the network. It determines which resource on the computer can access the network. A properly configured firewall increases the security of the system. You can enable and configure a firewall in Red Hat Enterprise Linux.

To configure the firewall, perform the following steps:

1. Select **System → Administration → Security Level and Firewall**. The **Security Level Configuration** dialog box appears.

   Figure 8.2 displays the **Security Level Configuration** dialog box.

Figure 8.2: Security Level Configuration Dialog Box

The options available in the Security Level Configuration dialog box are as follows:

- **Firewall** - enables or disables the firewall
- **Trusted services** - lists the services that are allowed access by the firewall

2. In the Trusted services pane, check the FTP, Mail (SMTP), SSH, Secure WWW (HTTPS), Telnet, and WWW (HTTP) check boxes.

3. Click **Apply**. Red Hat Enterprise Linux prompts for confirmation.

   Figure 8.3 displays the confirmation dialog box.

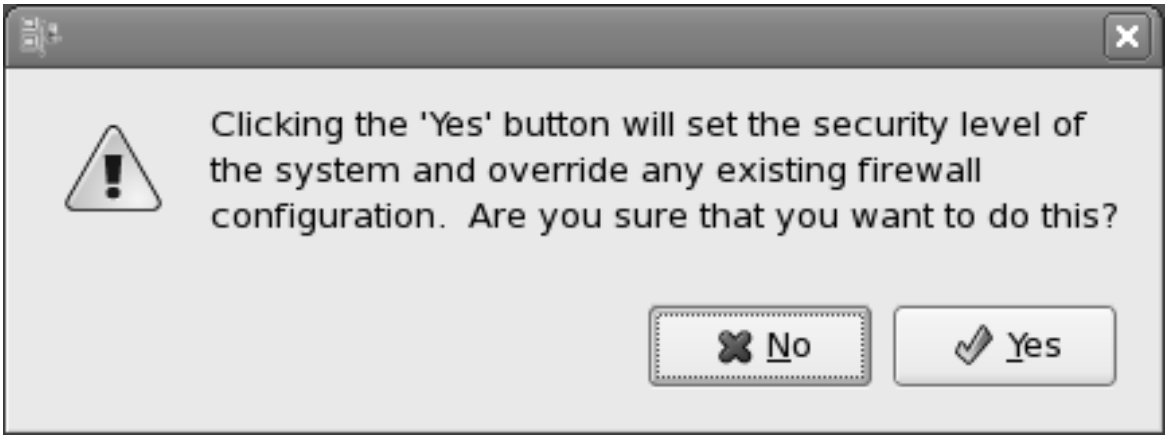


Figure 8.3: Enabling Firewall to Allow Access to Services

4. Click **Yes**.

5. Click **OK**.

8.6.2 Firewall Architecture Models

Firewall components can be combined together with different firewall architectures. The following are some of the firewall architectures:

- **Screening Router Architecture** - specifies the simplest form of firewall architecture. The architecture consists of a local network of computers, a screening router, and an external network, such as the Internet. The screening router functions as the firewall and allows data packets acceptable by the network to pass through it. This is an example of packet filtering. It is a low cost firewall system and does not provide an in-depth security.

Advantages:

- It is transparent.
- Compared to other architectures, it is easier to use and less expensive.

Disadvantages:

- It cannot handle all types of traffic.
- The configuration of this architecture is difficult.
- It possesses limited or no logging capabilities.
- It does not contain user authentication.
- The internal network structure is quite accessible.

Figure 8.4 displays the Screening Router Architecture.

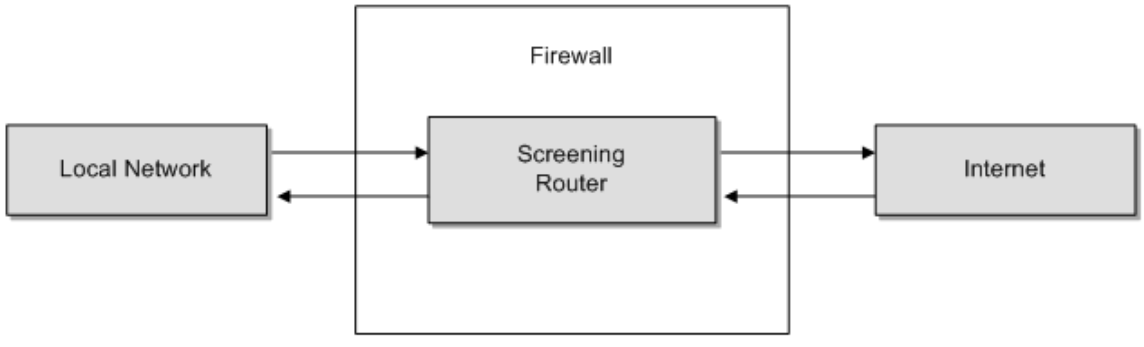


Figure 8.4: Screening Router Architecture

- **Dual-Homed Host Architecture** - provides a firewall system that consists of a Dual-Homed Host machine. A Dual-Homed Host machine has multiple IP addresses, each for a specific port. The firewall machine has the local network connected to one port and Internet on a different port. The communication between the local network and the Internet is permitted in two ways:

  - **Account on the Dual-Homed Host machine** - enables the user to logon to the Dual-Homed Host machine to access the Internet.
  - **Proxy Programs** - executes on the Dual-Homed Host machine to enable access to the Internet. Users do not need to log on to the firewall machine to communicate using proxy software.

Concepts

Advantages:

- This architecture is more secure than the Screening Router Architecture.
- The internal network structure is not accessible.

Disadvantages:

- It is not user friendly.
- It involves change in user behavior.
- It needs multiple proxies and proxies are not always available.

Figure 8.5 displays the Dual-Homed Host Architecture.

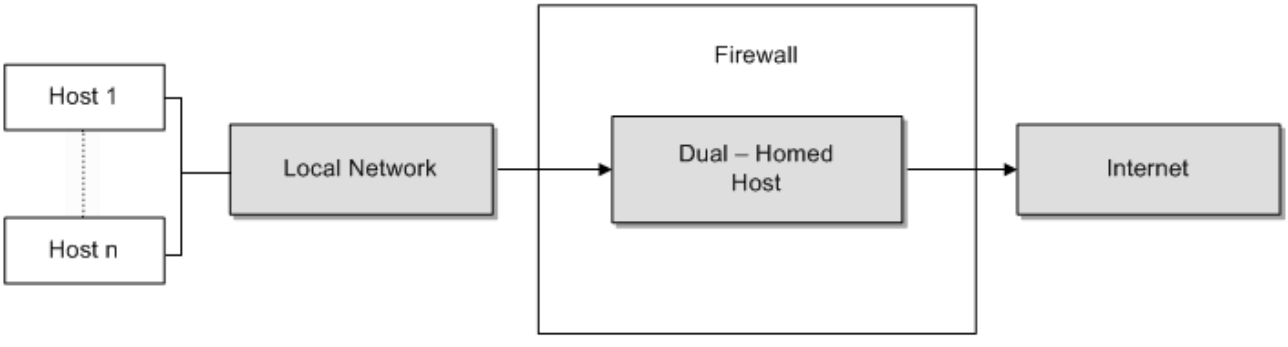


Figure 8.5: Dual-Homed Host Architecture

➢ **Screened Host Architecture** - consists of the screening router and the screened host. The Screening Router is located between the local network and the Internet to disable direct communication. The screened host is connected to the local network. An external network must connect to the screened host in order to connect to the local network.

This architecture is more flexible than Dual-Homed architecture because of packet filtering by the screening router, and services given by the screened host. The disadvantage of this architecture is that if an intruder bypasses the screened host, it can pose a security threat to the local network.

Advantages:

- Has a multiple authentication

- Has a transparent outbound access
- Has restricted inbound access

Disadvantages:

- Internal structure not hidden
- Single point of failure (router)
- No protection from compromised gateway

Figure 8.6 displays the Screened Host Architecture.

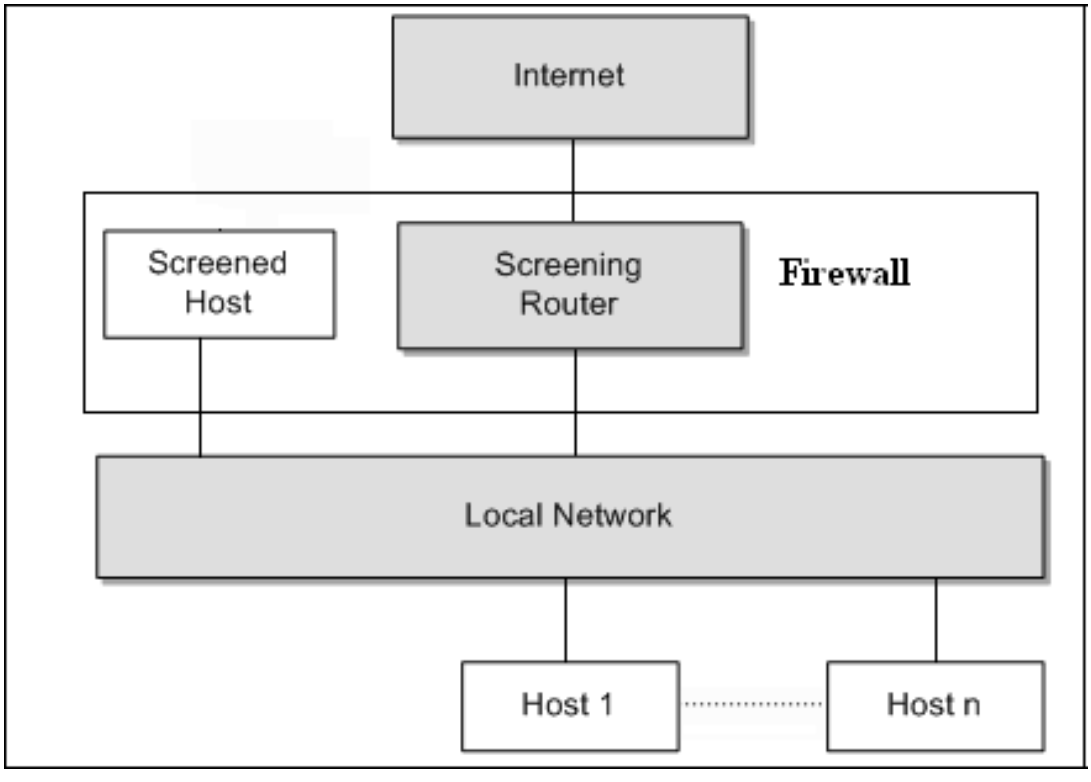


Figure 8.6: Screened Host Architecture

In figure 8.6, the screened host is connected to the local network. The screened host is accessible from the Internet. It will execute proxy programs to enable services.

The other hosts on the local network can access the Internet only after establishing a connection to the screened host.

- **Screened Subnet Architecture** - consists of a screened host connected to a network called as perimeter network (or perimeter net). The perimeter network is an additional network between the external and the internal networks. If an intruder successfully manages to access the outer boundaries of your firewall, then the perimeter net offers an additional layer of security for your internal systems. This router protects the perimeter net and the internal network from unauthorized Internet users. The internal router is connected between the perimeter net and the internal network. It performs the function of packet filtering and enables the data to flow from the internal network to the Internet. This architecture provides a better security as compared to the earlier given architectures since an intruder has to bypass both the routers and the screened host to break into the internal network.

  Advantages:

  - The working of this architecture is clear to the end users.
  - It is an adaptable form of architecture.
  - The internal network structure is not accessible.
  - It offers services to the outside network without changing the inside network.

  Disadvantages:

  - All security functions provided by gateway, a single point of security failure.

Figure 8.7 displays an external router connected between the Internet and the perimeter network in the screened host architecture.

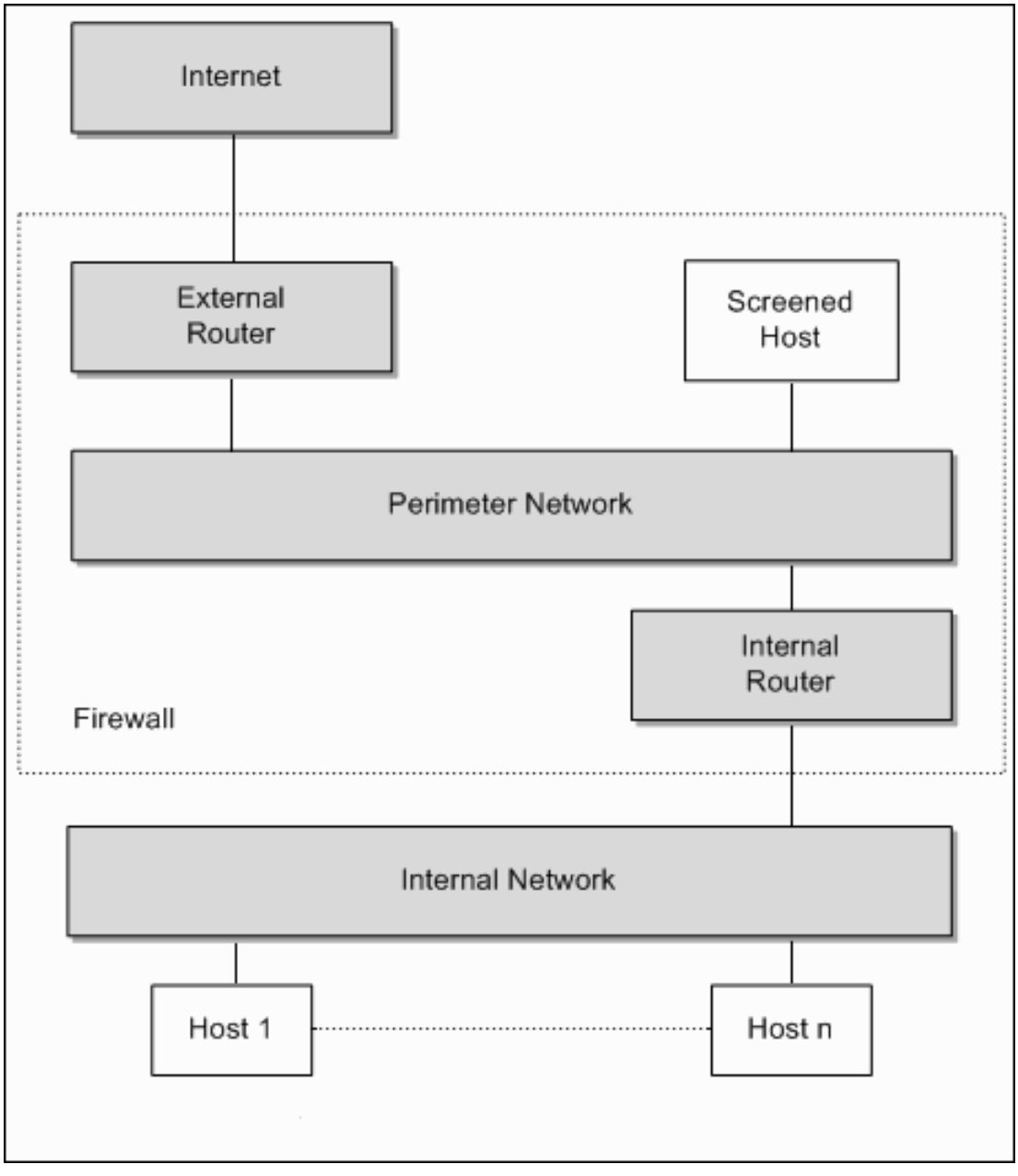


Figure 8.7: Screened Subnet Architecture

Concepts

Summary

- User authentication requires username and password for permitting access to a resource.
- The `.htaccess` file contains one or more configuration directives. The directives present in this file are applied to directory in which the file is saved and to all its sub-directories.
- Apache supports `basic` and `digest` authentication methods. Basic authentication is a simple and commonly used method for authenticating users. Digest authentication disables transmitting clear text passwords across the Internet. Digest authentication is implemented using the module `mod _ auth _ digest`.
- Password files are used to verify and authenticate the username and the password.
- Proxies are intermediate servers connected between a client and a remote server.
- Firewalls can be configured to block data from specific locations and allow only relevant and necessary data. Firewalls provide protection against outside attackers by protecting the network from unnecessary Internet traffic.
- In the Screening Router firewall architecture, it functions as a firewall allowing data packets acceptable to the internal network to pass through it.
- The Dual-Homed Host firewall architecture consists of a Dual-Homed Host machine that has multiple IP addresses, one for each port.
- The Screened Host firewall architecture consists of a screening router and a screening host.
- The Screened Subnet firewall architecture consists of two routers and a screened host that together function as a firewall to protect the internal network from unauthorized access.

Check Your Progress

1. The directives in _____ file apply to directory in which the file is saved and to all its sub-directories.

   a. `httpd.conf`

   b. `htaccess.conf`

   c. `.htaccess.conf`

   d. `.htaccess`

2. Which of the following argument of the `httpd` utility creates a new password file?

   a. `-p`

   b. `-s`

   c. `-d`

   d. `-c`

3. _____ provide protection against outside attackers by protecting the network from unnecessary Internet traffic.

   a. Proxies

   b. Password Files

   c. Firewalls

   d. Routers

Check Your Progress

4. _____ are intermediate servers that stand between a client and a remote server.

   a. Caching proxies

   b. Firewalls

   c. Packet Filters

   d. Proxies

5. Which type of firewall model architecture consists of a perimeter network?

   a. Screening Router Architecture

   b. Screened Subnet Architecture

   c. Dual-Homed Host Architecture

   d. Screened Host Architecture

Onlinevarsity

your e-way to learning

GROWTH

RESEARCH

OBSERVATION

UPDATES

PARTICIPATION

www.onlinevarsity.com

Session

9 Security (Lab)

Lab Guide

Objectives

At the end of this session, the student will be able to:

- *Create and maintain password files.*
- *Create user based authentication.*
- *Configure host based authentication.*
- *Configure firewall.*

The steps given in the session are detailed, comprehensive, and carefully thought through. This has been done so that the learning objectives are met and the understanding of the tool is complete. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Creating and Maintaining Password Files

Password files are used to authenticate the username and password from a list of known usernames and passwords. The `htpasswd` utility of Apache is used to create the password file.

1. **Open the Linux terminal.**
2. **To create a new password file for the user tim, enter the following command at the command prompt:**

   ```
   /usr/local/apache2/bin/htpasswd -c /usr/local/apache2/passwords tim
   ```

   The `htpasswd` command prompts for a password.
3. **Enter the password.**

Figure 9.1 displays the creation of the password file.

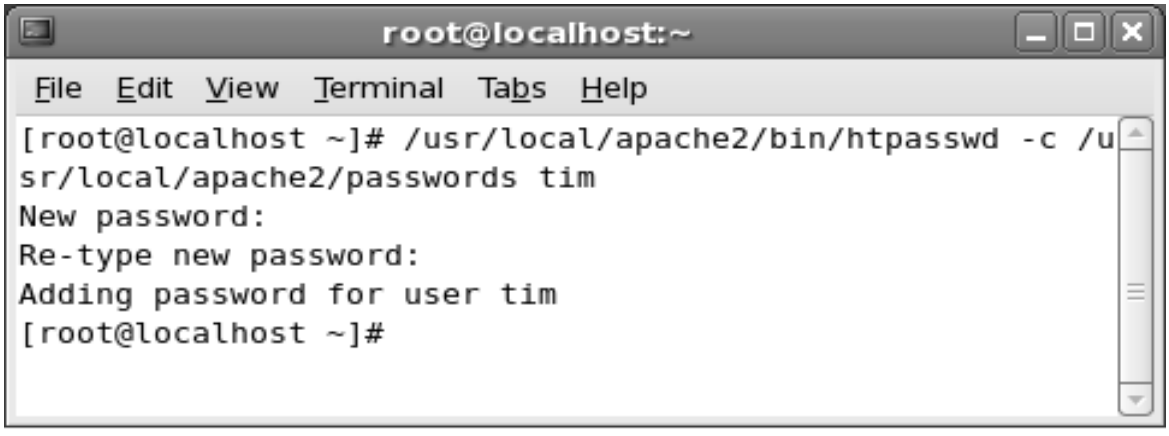


Figure 9.1: Creating a Password File

The `htpasswd` command creates a file named passwords in the `/usr/local/apache2` directory.

Creating User Based Authentication

User authentication requires the user to enter username and password for permitting access to a document. User authentication works using the procedures of authentication and authorization.

To create a user based authentication, perform the following steps:

1. **Using the GUI in Linux, create a directory named Mysite under the `/usr/local/apache2/htdocs` directory.**

2. **Open the gedit text editor.**

3. **Enter the following code:**

```
<html>
<body>
<h1>This is the mysite home page!</h1>
</body>
</html>
```

4. **Save the file as `index.html` in the `/usr/local/apache2/htdocs/Mysite` directory.**

5. **Assign read, write and execute permissions to the `index.html` file.**

6. **Create a directory named password under the `/usr/local/apache2` directory.**

7. **Copy the passwords file from the `/usr/local/apache2` directory to the `/usr/local/apache2/passwd` directory.**

8. **Open the `httpd.conf` file using the gedit text editor.**

9. **Enter the following code in the Directory section of the configuration file.**

```
<Directory /usr/local/apache2/htdocs/Mysite>
 AuthType Basic
 AuthName "Test Files"
 AuthUserFile /usr/local/apache2/passwd/passwords
 Require user tim
 Order allow,deny
 Allow from all
 Satisfy all
</Directory>
```

10. **Save the changes and restart Apache Web server.**

11. **Open the Mozilla Firefox Web browser.**

12. **Enter `http://localhost/Mysite` in the address bar.**

Figure 9.2 displays the password prompt.

Figure 9.2: Password Prompt

13. **Enter tim in the username box.**
14. **Type password in the password box.**
15. **Click OK. The homepage of Mysite is displayed.**

Figure 9.3 displays the homepage of Mysite.

Figure 9.3: Displaying the Homepage after Authentication

Session 9

Security (Lab)

To restrict access to files in the `/usr/local/apache2/htdocs/secret` directory to username tim and password tim, perform the following steps:

1. **Using the GUI in Linux, create a directory named secret under the `/usr/local/apache2/htdocs` directory.**

2. **Open the `httpd.conf` file using the gedit text editor.**

3. **Enter the following code in the Directory section of the `httpd.conf` file:**

```
<Directory /usr/local/apache2/htdocs/secret>
    AuthType Basic
    AuthName "Restricted Files"
    AuthUserFile /usr/local/apache2/passwd/passwords
    Require user tim
    Order deny,allow
    Deny from all
    Allow from foo.com
</Directory>
```

4. **Save the changes and restart Apache Web server.**

5. **Open the Mozilla Firefox Web browser.**

6. **Enter `http://localhost/secret` in the address bar and press Enter. The Forbidden message is displayed.**

Figure 9.4 displays the permission message.

Figure 9.4: Restricting Access to Files in a Directory

Configure Host Based Authentication

Host Based Authentication directly uses the incoming HTTP request and gives access to the document. It controls the access to a Web resource.

To configure Host Based Authentication, perform the following steps:

1. **Open the httpd.conf file using the gedit text editor.**

2. **Enter the following code in the Directory section of the configuration file.**

```
<Directory usr/local/apache2/htdocs>
    Limit <GET POST>
    Order deny,allow
    deny from host.Mysite.com
    allow from 127.0.0.1
    </Limit>
</Directory>
```

3. **Save the changes and restart Apache Web server.**

4. **Open the `index.html` file from the `/usr/local/apache2/htdocs` directory using the gedit text editor.**

5. **Delete the existing code and enter the following code:**

```
<html>
<body>
<h1>It Works!</h1>
</body>
</html>
```

6. **Save the changes and restart Apache Web server.**

7. **Open the Mozilla Firefox Web browser and enter `http://localhost/Mysite` in the address bar. An error message is displayed.**

8. **Enter 127.0.0.1 in the address bar and press Enter.**

Figure 9.5 displays the Apache index page.

Figure 9.5: Apache Index Page

Configuring Firewall

Firewalls are configured to block data from certain locations and allow access to only relevant and necessary data. It determines which resource on the computer can access the network.

To configure firewall settings, perform the following steps:

1. **Login to Linux as root.**
2. **Select System → Administration → Security Level and Firewall. The Security Level Configuration dialog box is displayed.**

Figure 9.6 displays the Security Level Configuration Dialog Box.

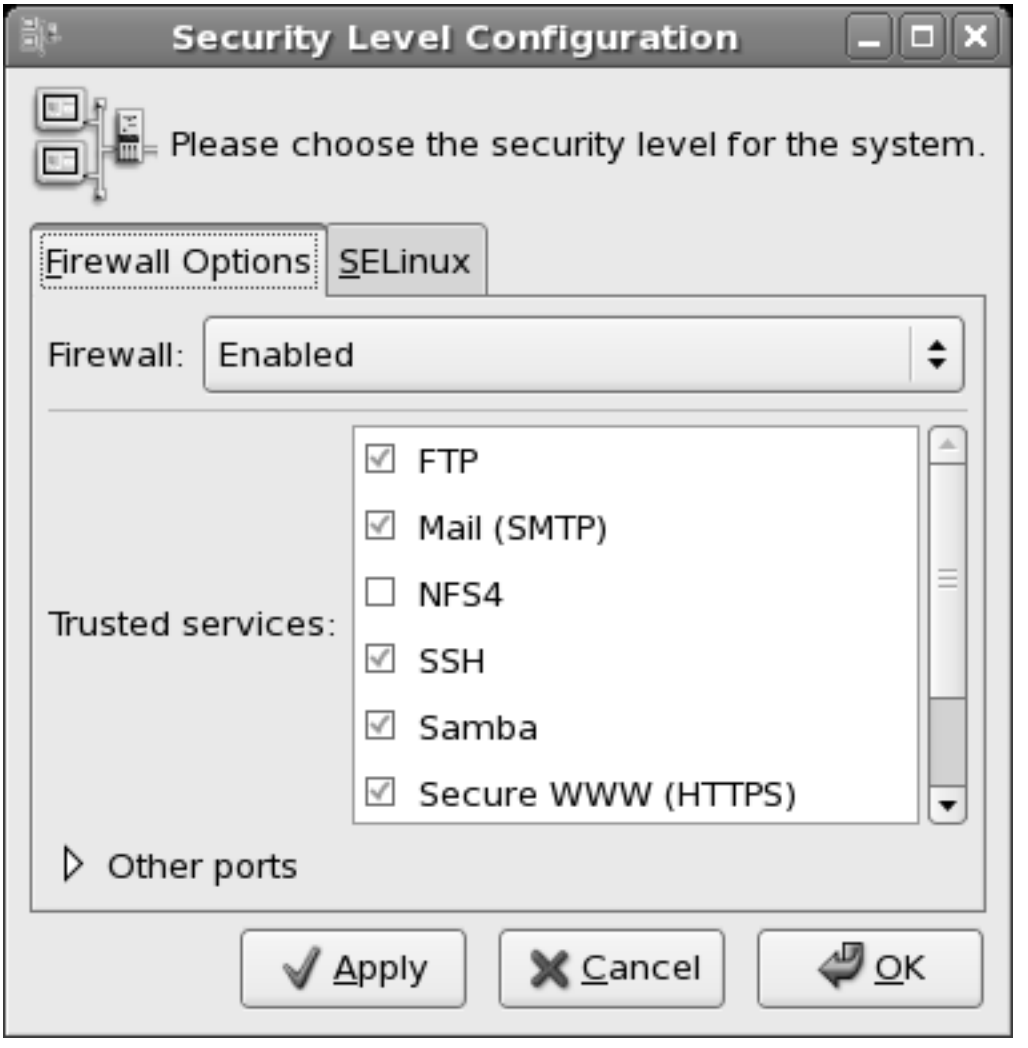


Figure 9.6: Security Level Configuration Dialog Box

3.	**Select Enabled from the Firewall list.**
4.	**In the Trusted Services pane, check the FTP, Mail (SMTP), SSH, Samba, and Secure WWW (HTTPS) check boxes.**
5.	**Click Apply. A confirmation message box is displayed.**

Figure 9.7 displays the security level confirmation message dialog box.

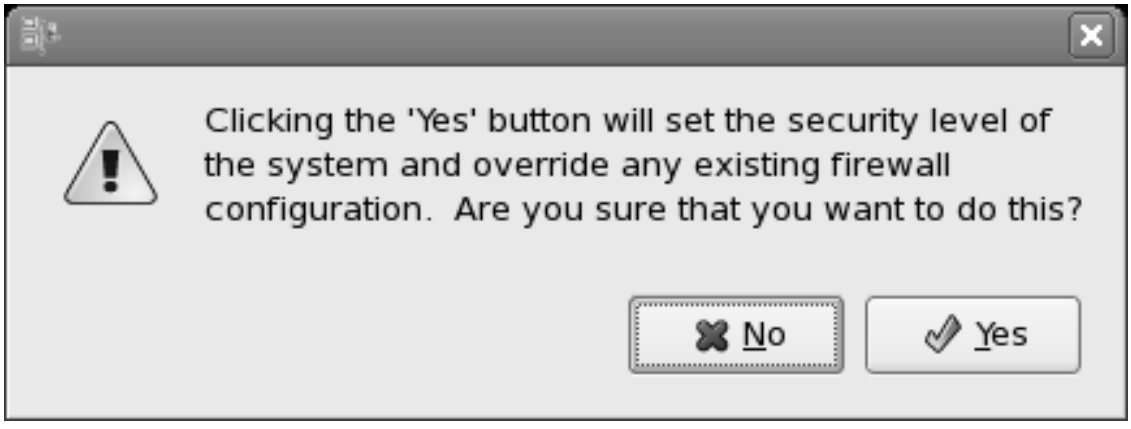


Figure 9.7: Security Level Confirmation Message Dialog Box

6. **Click Yes. The firewall is configured.**

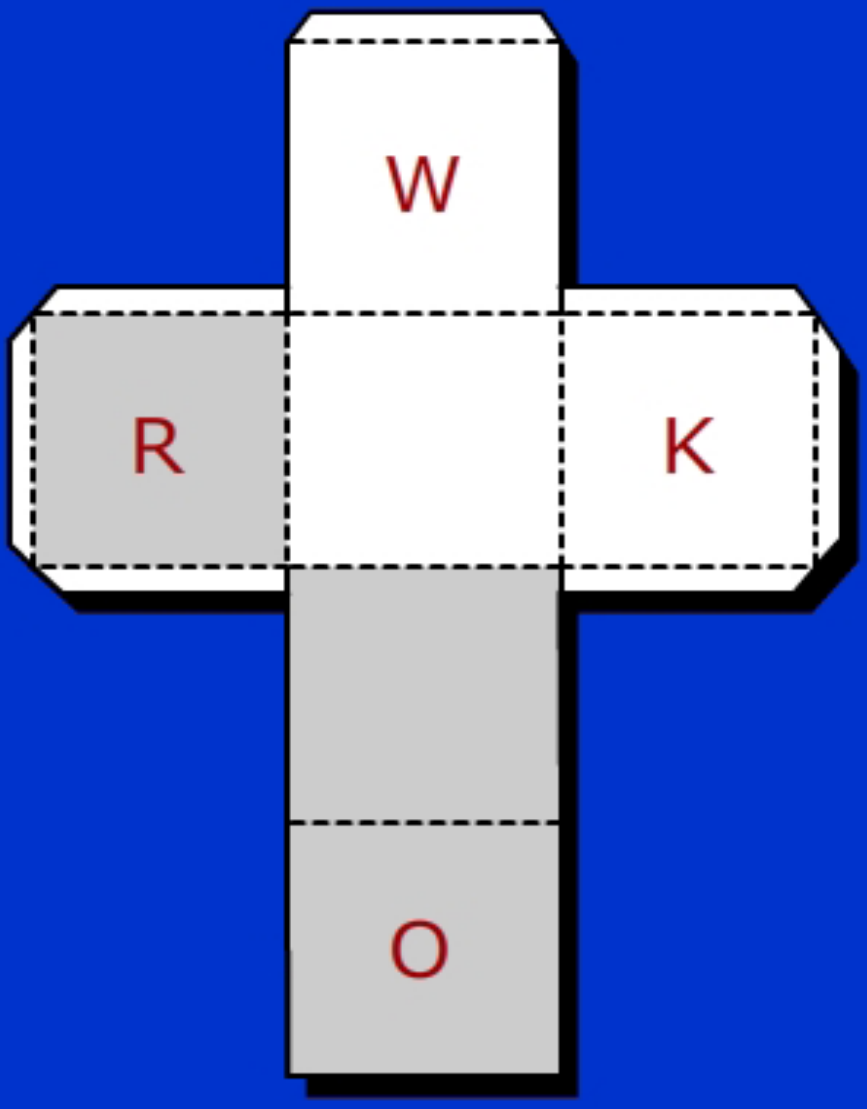
Do It Yourself

1. Create a password file with the username as `smith` and password as `root`.

2. Disable the firewall.

Onlinevarsity
your e-way to learning

WORK



ASSIGNMENT:

Form the cube to read '**WORK**'.

"Practice does not make perfect. Only perfect practice makes perfect."
- Vince Lombardi

Practice the Practicals for Perfection @

www.onlinevarsity.com

Session

10 Secure Sockets Layer

Concepts

Objectives

At the end of this session, the student will be able to:

- *Explain the working of SSL protocol.*
- *Explain the use of SSL module.*
- *Explain the installation process of a private key.*
- *Explain the creation of a certificate-signing request and a temporary certificate.*

10.1 Introduction

The Secure Sockets Layer (SSL) protocol is used for secure communication over the Internet. The SSL protocol disables intruders from intercepting the information transmitted over the Internet. SSL uses encryption and decryption techniques for this purpose. Originally designed by Netscape, the SSL protocol is currently one of the most secure communication protocol. SSL defines an encrypted link between a Web server and a browser to ascertain that the transmitted data remains confidential. It is an industry standard and a large number of Web sites utilize it to establish secure online transactions.

In this session, you will learn about the SSL protocol and the cryptographic techniques that it uses. You will also learn about the SSL directives in Apache server. In addition, you will learn how to install a private key in Apache server. You will also learn how to create a certificate signing request and a temporary certificate for Apache server.

10.2 Secure Sockets Layer

All Web servers use common methods to authenticate users based on an unencrypted username and password. Unencrypted information is sent as a plain text and can be hacked. Apache server implements these schemes as modules, combining the functions of user authentication and access authorization.

However, this traditional Web security model has two weaknesses. They are as follows:

- **Eavesdropping** - In a traditional Web security model, there are chances that someone with malicious intent and capable of snooping on network packets can access sensitive information, such as passwords.
- **Credibility** - Based on the user ID and password provided, the client cannot be completely sure if the Web server is legitimate.

Man-in-the-middle attack are attempts where intruders mask the identity and appear as the legitimate host in every respect. The user unknowingly perceives the intruder as the actual recipient and divulges all the information. With the growth in e-commerce, man-in-the-middle attack are likely to be more common.

In 1990s, Netscape Communications developed a scheme called SSL to address the security vulnerabilities in the traditional Web security model.

The SSL is a protocol that ensures secure data transmission over the Internet. The SSL protocol runs above the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol and below the application layer protocol, HTTP. The TCP/IP protocol monitors the transport and routing of data over the Internet.

The SSL protocol provides a private channel of communication between the Web server and the browser, and also assures the clients that they are connected to the appropriate Web server. SSL achieves this by comparing the URL encoded in the certificate and presented by the remote Web server to the URL that the browser uses to locate the server. Both these URLs must match.

The main objectives of an SSL are as follows:

- Authenticating the client and server communication with each other
- Ensuring data security and integrity

SSL uses the public-and-private key encryption system from the Rivest, Shamir, and Adleman (RSA) algorithm—an algorithm used for public key cryptography. SSL also includes the use of a certificate. Therefore, to understand the working of SSL, you need to understand the concept of cryptographic algorithms, message digest functions, digital signatures, and certificates.

The primary objective of SSL is to secure data during transport between the client and the server. The data must be encrypted and should be readable only by the recipient.

10.2.1 Cryptography

SSL is a protocol that ensures secure transactions between a Web browser and a Web server through cryptography. Mostly, only the server end is authenticated, which implies that the client is certain about the server's identity, but this does not imply vice versa. You can use SSL certificates or other similar methods for client authentication.

SSL protects confidential information through the use of cryptography. Cryptography is the process of converting data into scrambled code, transmitting it across a given network, and deciphering it at the other end. The cryptographic algorithms enable you to transform information into an unreadable format before sending it over the network. The encrypted message is known as cipher text and the receiver can decrypt it by using a secret key at the receiving end.

Concepts

Figure 10.1 displays a Cryptographic algorithm.

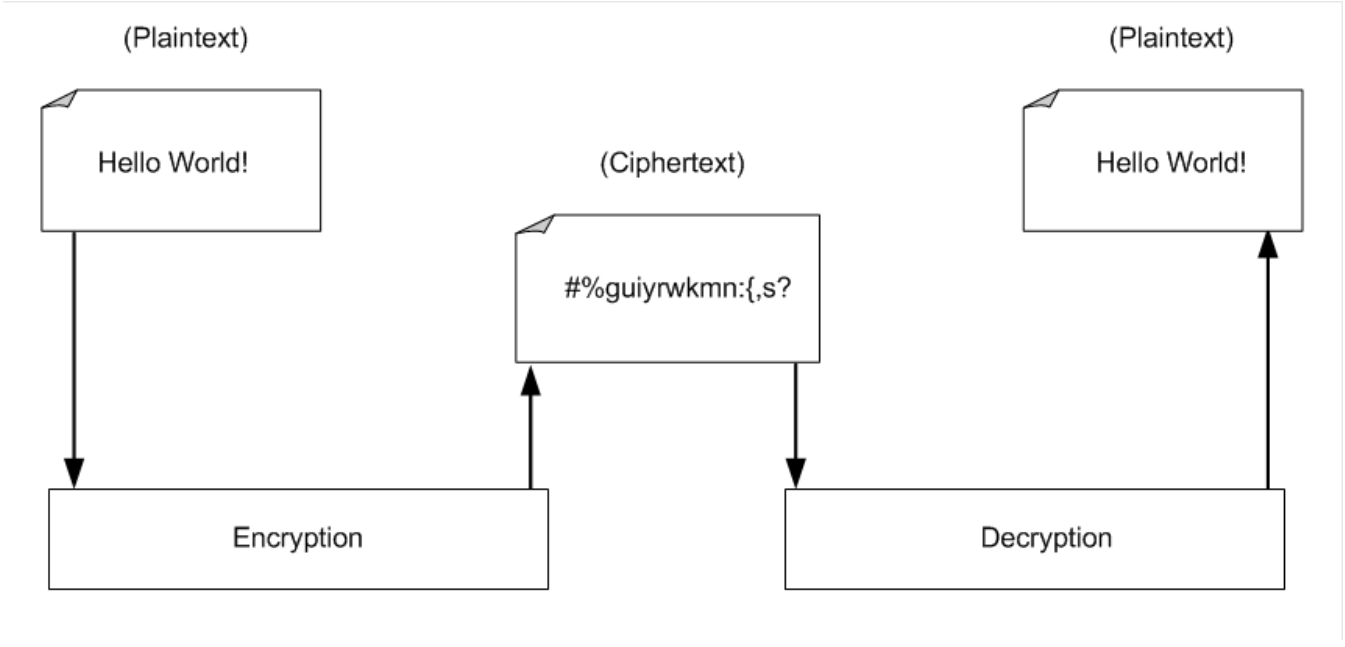


Figure 10.1: Cryptographic Algorithm

Cryptography uses two main styles or forms of encrypting data: symmetrical and asymmetrical.

- **Symmetric cryptography** - Symmetric algorithms use the same key for encryption and decryption of message or a communication system. In other words, the sender and the receiver share a common key. It is also known as secret-key cryptography, shared-key cryptography, private-key cryptography, or conventional cryptography. This key is a secret piece of information and is used for encryption and decryption. The symmetric system can be used only if the key is available to both the parties involved in the communication. If the key is hacked, then the encrypted message can be decrypted, modified, and again encrypted using the same key.

  Advantages of symmetric key encryption:

  - It is a fast mode of encryption.
  - It can be used with large amounts of data.

  Disadvantages of symmetric key encryption:

  - It can be hacked and decoded unless you have carefully planned the coding.
  - It is mandatory for both the participants to know the single key being used.

Figure 10.2 displays a symmetric key system.

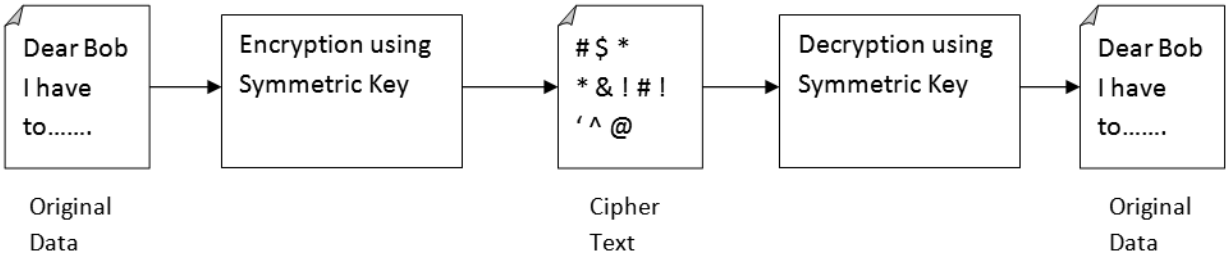


Figure 10.2: Symmetric Key System

- **Asymmetric cryptography** - It is also known as public key cryptography. Asymmetric cryptography uses two different keys for encryption and decryption. In this system, each machine involved in the communication must hold a pair of keys: a public key and a private key. The public key is published on the network, but the private key is applicable only to the machine. The public or the private keys can be used for encryption. A message encrypted with a public key can be decrypted using only the private key corresponding to that public key. In general, to transmit data, you must encrypt the data using the receiver's public key and the receiver decrypts the data using its private key.

Figure 10.3 displays an asymmetric key system.

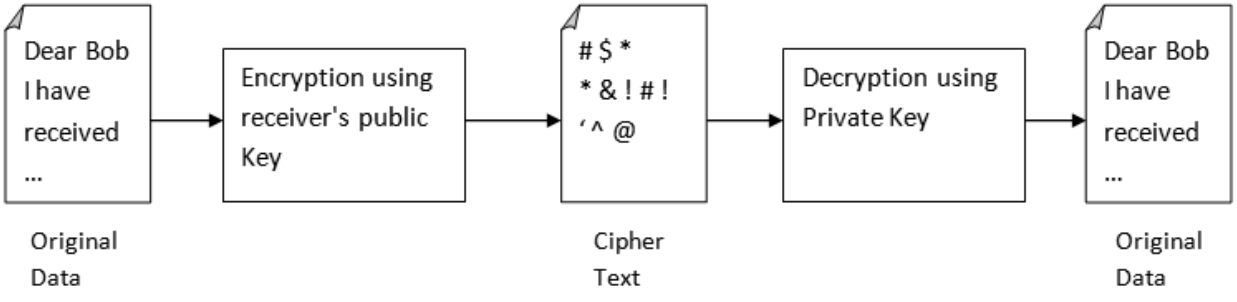


Figure 10.3: Asymmetric Key System

10.2.2 Message Digest

Message digests provide a reliable method to ensure message integrity. A message digest function converts a plain text message into a short, fixed-length string. This encrypted string is known as hash. As you cannot obtain the original text from the hash, it is also known as 'one-way function'.

Any change to the text message modifies the corresponding message hash. These attributes of a message digest permit it to act as a digital fingerprint of the original message.

In cryptography, Message-Digest algorithm 5 (MD5) is a cryptographic hash function that contains a 128-bit (16-byte) hash value. A number of security applications have used MD5 to verify the integrity of files. MD5 hash is expressed as a 32-digit hexadecimal number.

The integrity of data is verified by transmitting the summary along with the data. The receiver calculates a summary of its own from the received message and compares it with the received summary. If the two summaries match exactly, then it implies that the message was received intact. Digital signatures can be used for this purpose.

Figure 10.4 displays a message digest system.

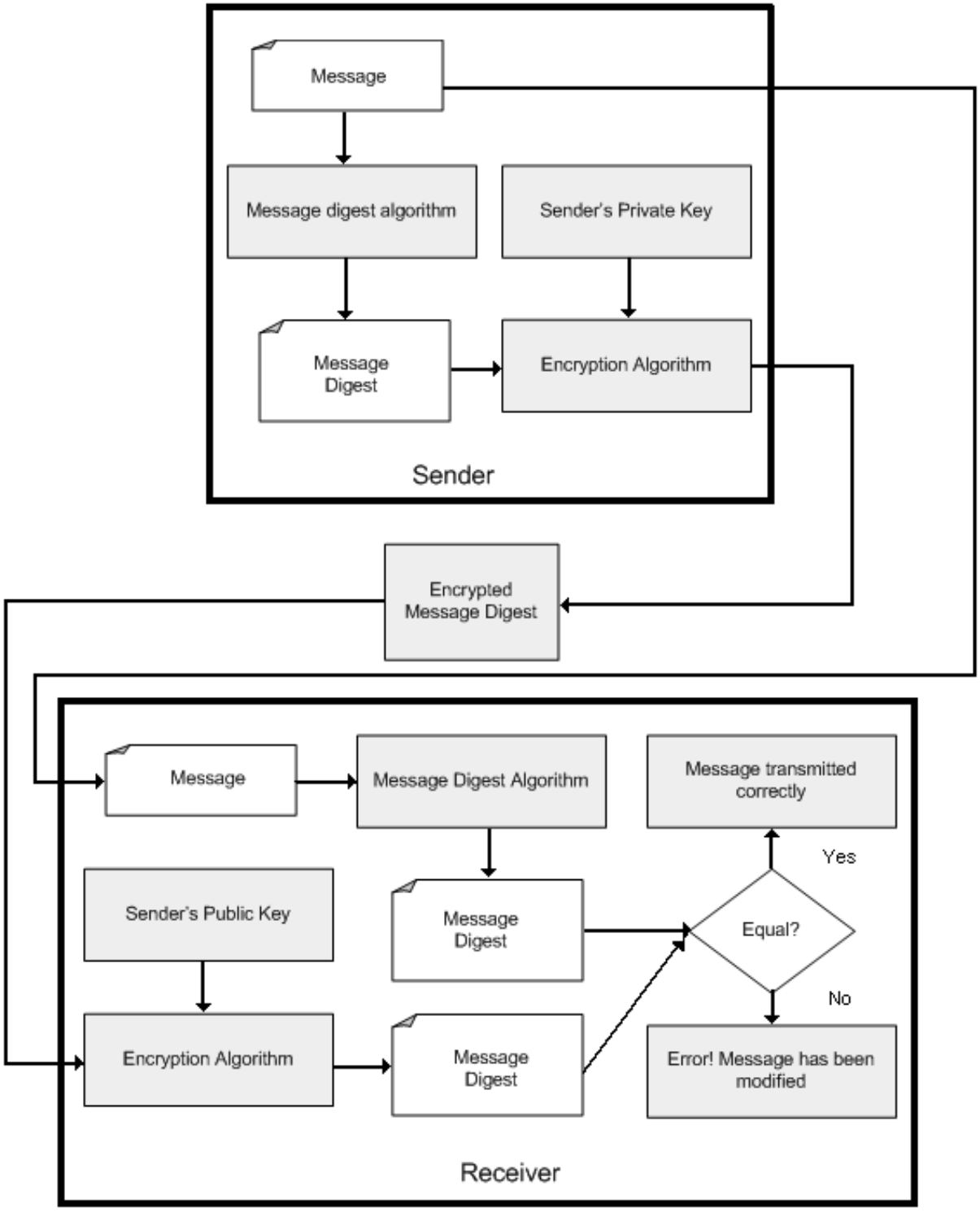


Figure 10.4: Message Digest System

In figure 10.4, the message to be transmitted is encrypted using the encryption, and the message digest algorithm, and the encrypted key. At the recipient end, the encryption algorithm is used to generate the message digest of the message. This message digest is then compared with the message digest received from the sender. If the two message digests are identical, it implies that the message has not been tampered and has been transmitted correctly.

10.2.3 Digital Signatures

Monetary or financial transactions on the Internet require authentication from the sender. A digital signature is created by encrypting the message digest with the sender's private key. A digital signature is used to authenticate a digital message or document. A valid digital signature assures the recipient about the credibility of the user and the integrity of the message. Digital signatures use a form of asymmetric cryptography.

Digital signatures are used to implement electronic signatures but not all electronic signatures use digital signatures.

Digital signatures should have the following properties:

- If you generate a signature from a fixed message and a fixed private key, you should authenticate it by using the corresponding public key.
- You should be unable to generate a valid signature for an entity that does not possess the private key.

A digital signature scheme typically consists of three algorithms:

- **Key Generation Algorithm** - selects a private key from a set of possible private keys and transmits a corresponding public key for the chosen private key
- **Signing Algorithm** - generates a signature for a given message and private key
- **Signature Verifying Algorithm** - authenticates a given message using the public key and the signature

Some common reasons for including a digital signature to messages are as follows:

- **Authentication** - Digital signatures can be used to validate the source of messages. When a user owns a digital signature, it confirms that the message was sent by that user.
- I**ntegrity** - Digital signatures maintain the integrity of a message. Any attempt to modify a digitally signed message nullifies the digital signature.

- **Non-repudiation** - If a user has digitally signed a piece of information, at any point of time, the user cannot deny having signed it.

Figure 10.5 displays a digital signature system.

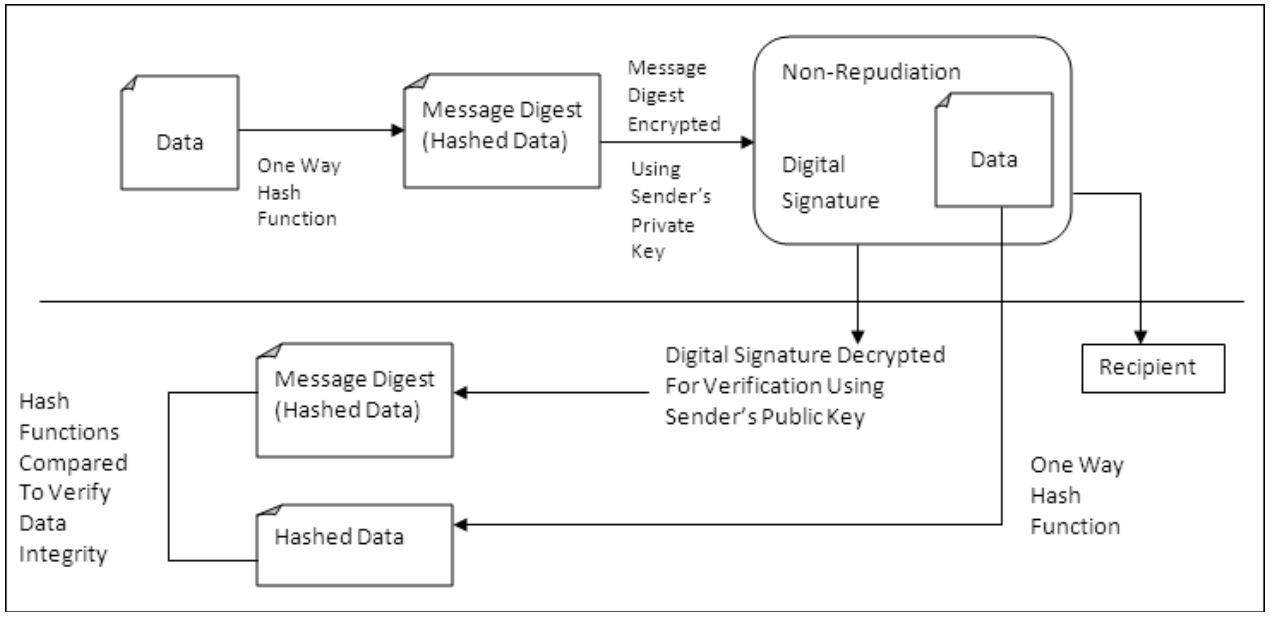


Figure 10.5: Digital Signature System

In figure 10.5, the first half of the image displays the encryption process, where the data or message is encrypted using the private key. The second half of the image displays the signature verification or decryption process where the key is decrypted and the authenticity of the message and the sender is verified.

A digital certificate works as follows:

1. The sender transmits the data, that is encrypted with the key generated by the Message Digest, and the corresponding hash is generated.
2. The encrypted data is validated using digital signature, which ensures the following key features, authentication, integrity, and non-repudiation.
3. The hashed data and digital signature is sent to the receiver along with the sender's public key. Both the data are compared to ensure that the integrity of the data is maintained.

10.2.4 Certificates

When a computer communicates with another computer on a network, both the machines must authenticate themselves to each other. For example, if John sends a private message to the bank server, he must ensure the server's identity.

Concepts

Similarly, the bank server must ensure that the received and signed message belongs to John. Certificates are used to solve this problem of identification. A digital certificate establishes credentials for any online transactions. A certificate generally contains information, such as the distinguished name, public key of the applicant, signature of the Certification Authority (CA), and the period of validity. CA is a third-party trusted agency that issues this digital certificate.

10.3 SSL Protocol

SSL is a protocol that ensures privacy and integrity between two communicating applications using TCP/IP. The SSL protocol runs above TCP/IP and below higher-level protocols, such as HTTP or IMAP.

SSL permits the following activities:

- Authentication of an SSL-enabled server to an SSL-enabled client
- Authentication of the client to the server
- Establishment of an encrypted connection between the client and the server

With the help of these capabilities, SSL addresses the following fundamental issues related to communication over the Internet:

- SSL server authentication allows a user to validate the server's identity. Similarly, SSL client authentication permits the server to verify the user's identity. SSL server and client authentication can be used during online transactions, such as, transmission of confidential information.
- Confidentiality is the key feature of an encrypted SSL connection. With the help of public or private key algorithms, the sender encrypts the data and the receiver decrypts the data, thus securing data. In addition, an encrypted SSL connection is protected with a mechanism that automatically determines if the data has been modified during the transit.

SSL further consists of two sub-protocols:

- **SSL Record Protocol** - The SSL record protocol specifies the format for data transmission and handles the encryption for all messaging. SSL records consist of the encapsulated data, digital signature, message type, version, and length. SSL records are 8 bytes long. Encrypted messages also include padding and pad length in the frame, since the record length is fixed.

  The SSL record protocol provides the following two services for SSL connections:

  - **Confidentiality** - Uses symmetric encryption to encrypt the application data with a shared secret key defined by the handshake protocol

- **Message Integrity** - Uses a message authentication code (MAC) with shared secret key

SSL record protocol works as follows:

1. Accepts the message to be transmitted
2. Breaks up the data into smaller blocks
3. Compresses the data (this step is optional)
4. Applies a Media Access Control address (MAC address)
5. Encrypts the data
6. Adds a header
7. Transmits the resulting unit in a TCP segment
8. Decrypts, verifies, decompresses, and reassembles the received data and delivers it to the calling application

Figure 10.6 depicts the working of the SSL record protocol.

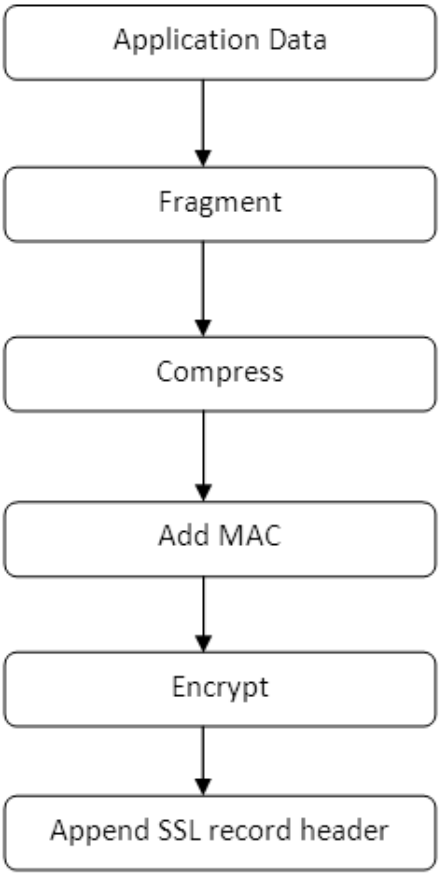


Figure 10.6: Flowchart for SSL Record Protocol

- **SSL Handshake Protocol** - The SSL handshake protocol is a process where the client and server exchange information and negotiate services. The SSL handshake protocol uses the SSL record protocol to exchange messages between an SSL-enabled client and SSL-enabled server when an SSL connection is established. An HTTP-based SSL connection is always initiated by the client using a URL starting with `https://` instead of `http://`. At the beginning of an SSL session, an SSL handshake is performed, which produces the cryptographic parameters of the session.

Figure 10.7 displays the processing of an SSL handshake.

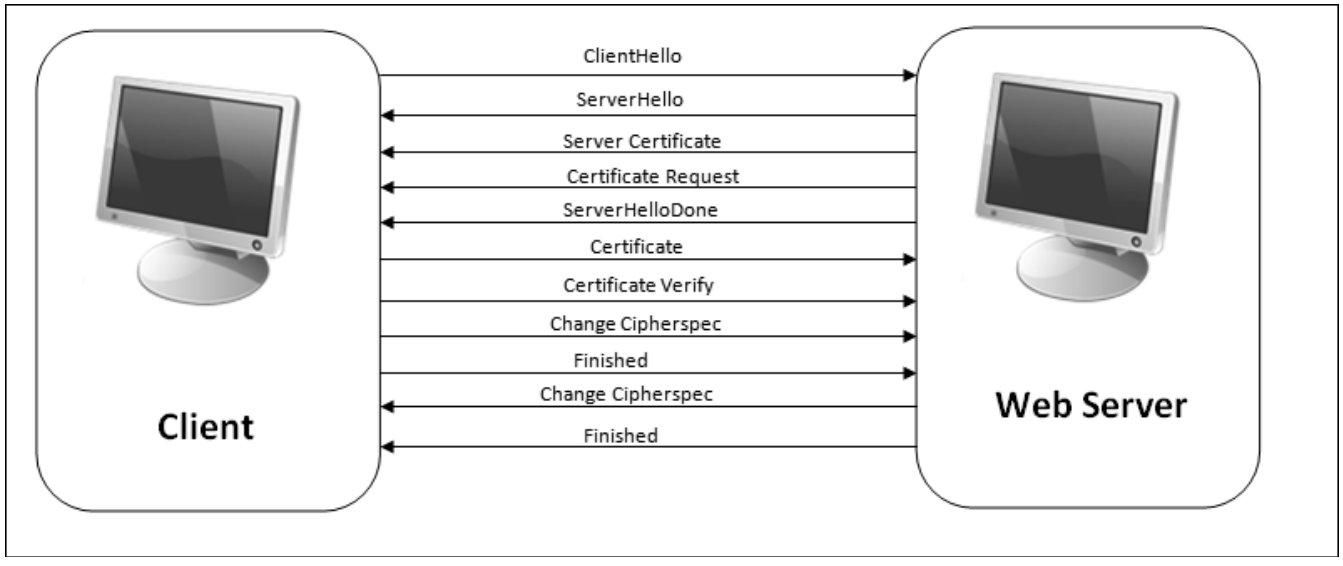


Figure 10.7: SSL Handshake

The SSL handshake sequence between the client and the server is as follows:

1. The client establishes the connection by sending a `ClientHello` signal. It includes the cryptographic capabilities of the client sorted in client preference order, such as the version of SSL, the cipher suites, and the data compression methods supported by the client. The message also contains a 32-byte random value to be used for key generation.

2. The server responds with a `ServerHello` signal, which includes the cipher suite and the data compression method selected by the server and the session ID. The server also sends its digital certificate containing its public key to the client.

3. The SSL server sends the server certificate to the client for verification. When the server requires authenticating the client using a digital certificate, the server sends a client certificate request. This request contains the types of certificates supported and the names of CAs.

4. The server again sends a `ServerHelloDone` signal back to the client.

5. The SSL client verifies the signature on the digital certificate issued by the server. It also checks whether the `CipherSuite` used by the server is acceptable.

6. The SSL client transmits the byte string to the server. This is done to enable the server and the client to generate the key for digitally signing message data.

7. The SSL server verifies the signature on the client certificate.

8. In response to the client certificate request from the SSL server, the SSL client transmits a string encrypted with the private key and the digital certificate. The string transmitted by the client to the server is a random byte string.

9. The client issues the `ChangeCipherSpec` message to indicate that all the messages sent thereafter must be encrypted using the established key.

10. The SSL client encrypts a 'finished' message with the secret key and sends it to the SSL server, in order to culminate the client part of the handshake.

11. The server sends its own `ChangeCipherSpec` message to the client, which has the same purpose as that of the client's `ChangeCipherSpec` message.

12. The SSL server sends the SSL client a 'finished' message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.

13. For the duration of the SSL session, the SSL server and SSL client can now exchange messages that are symmetrically encrypted with the shared secret key.

10.4 Using the SSL Module

Using the `mod_ssl` module, you can configure Apache server to implement SSL protocol. You must build and install the OpenSSL libraries, that provide the encryption on which `mod_ssl` is based. The OpenSSL library is available for free at `http://www.openssl.org/`. The default port for an SSL connection is port 443.

To configure Apache server for accepting requests on port 443, enter the following code as shown in Code Snippet 1 in the `httpd.conf` file.

Code Snippet 1:

```
Listen 443
```

Note: The `Listen` directive enables incoming requests on a particular port or on a specific address-and-port combination.

If you specify only a port number, the server listens to the given port on all interfaces. If you specify an IP address along with the port, Apache Web Server responds to all of the requests on the given port and interface.

You can define multiple `Listen` directives and the Apache Web server will process all the requests from the defined list of addresses and ports.

Some of the directives provided in the `mod_ssl` module are as follows:

- **SSLCertificateFile Directive** - The `SSLCertificateFile` directive specifies the location of the server certificate.

  The syntax for this directive is as follows:

```
SSLCertificateFile filepath
```

  where,

  `filepath` - defines the location of the server certificate

  For example, to specify the location of the server certificate, enter the following code as shown in Code Snippet 2 in the `httpd.conf` file.

  Code Snippet 2:

```
SSLCertificateFile /usr/local/apache2/conf/ssl.crt/server.crt
```

- **SSLCertificateKeyFile Directive** - The `SSLCertificateKeyFile` directive specifies the location of the private key file of the server. This directive is used if the private key is not stored in the server certificate.

Note: When the `SSLCertificateFile` directive is used and the file contains both the certificate and the private key, do not use this directive.

  The syntax for this directive is as follows:

```
SSLCertificateKeyFile filepath
```

  where,

  `filepath` - defines the location of the private key file

  For example, to specify the location of the private key file of the server, enter the following code as shown in Code Snippet 3 in the `httpd.conf` file.

  Code Snippet 3:

```
SSLCertificateKeyFile usr/local/apache2/conf/ssl.key/server.key
```

- **SSLCertificateChainFile Directive** - The `SSLCertificateChainFile` directive specifies the location of the certificate chain file. This file consists of a list of accepted CA, ranging from the issuing CA certificate of the server certificate to the root CA certificate. Such a file just joins the various PEM-encoded CA certificate files, usually in certificate chain order. The server accepts only those certificates that are signed by this list of CAs.

  The syntax for the `SSLCertificateChainFile` is as follows:

```
SSLCertificateChainFile filepath
```

  where,

  `filepath` - specifies the location of the chain list file

  For example, to specify the location of the `SSLCertificateChainFile`, enter the following code as shown in Code Snippet 4 in the `httpd.conf` file.

  Code Snippet 4:

```
SSLCertificateChainFile usr/local/apache2/conf/ssl.crt/ca.crt
```

- **SSLEngine Directive** - The `SSLEngine` directive enables or disables the SSL protocol engine. It is generally used inside a `<Virtual Host>` section to enable SSL for a specific virtual host.

  The syntax for this directive is as follows:

```
SSLEngine on|off
```

  For example, to enable the SSL protocol, enter the following code as shown in Code Snippet 5 in the `httpd.conf` file.

Code Snippet 5:

```
<VirtualHost _default_:443>
  SSLEngine on
  …
</VirtualHost>
```

By default, the `SSLEngine` directive is off.

- **SSLProtocol Directive** - The `SSLProtocol` directive controls which versions of the SSL protocol will be accepted in new connections.

The syntax for this directive is as follows:

```
SSLProtocol   [+ | -] protocol
```

where,

`protocol` – specifies a particular protocol

The different protocol values are as follows:

- **SSLv2** - Original SSL protocol designed by Netscape and represents the SSL protocol version 2
- **SSLv3** - Represents the SSL protocol version 3 and is supported by most browsers
- **TLSv1** - Represents the Transport Layer Security protocol, version 1.0 and is the successor to SSLv3, but not yet supported by any popular browsers
- **All** - Represents all these protocols
- **+** - Includes the specified protocol
- **-** - Excludes the specified protocol

For example, to include all the SSL protocol versions except version 3, enter the following code as shown in Code Snippet 6 in the `httpd.conf` file.

Code Snippet 6:

```
SSLProtocol all -SSLv3
```

- **SSLVerifyClient Directive** - The `SSLVerifyClient` directive specifies the certificate verification level for client authentication.

The syntax for this directive is as follows:

```
SSLVerifyClient level
```

where,

`level`- represents the various certification level and can have the following values:

- **none** - denotes that no client certificate is required
- **optional** - implies that a client can present a valid certificate
- **require** - implies that a client must present a valid certificate
- **optional_no_ca** - implies that a client can present a valid certificate, but it need not be verified successfully

10.5 Installing a Private Key

A private key must be installed in order to enable the SSL protocol. A private key contains information about the public key also. Therefore, public key is not required to be generated separately. The OpenSSL utility is used to render a private key for Apache server. You can create two types of private keys namely, RSA and Digital Signature Algorithm (DSA).

To generate a RSA key for encryption and signing, enter the following command as shown in Code Snippet 7 at the command prompt.

Code Snippet 7:

```
openssl genrsa -des3 -out privkey.pem 2048
```

This command creates a 2048-bit RSA key and stores it in the file `privkey.pem`.

Figure 10.8 displays the output of the command.

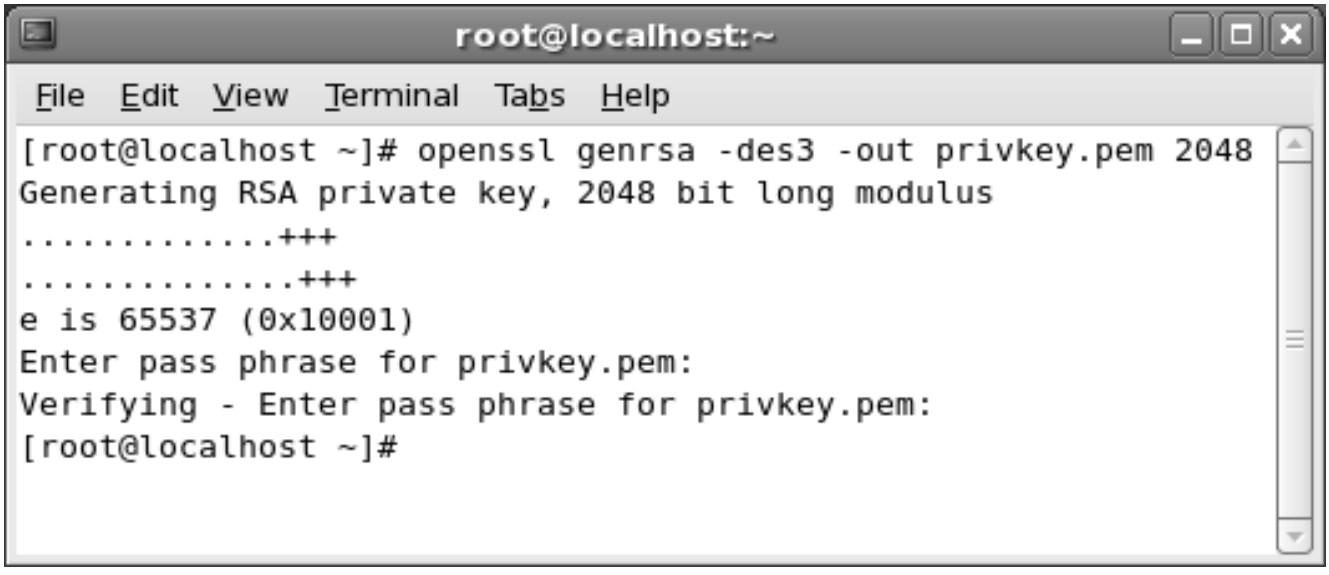


Figure 10.8: Generation of RSA Key

As shown in figure 10.8, once you enter the preceding command, you are prompted to enter the pass phrase (the protecting password).

Note: If the flag, `-des3`, is not included in the command, the key will not be protected by a password.

To generate a DSA key for signing only, enter the following command as shown in Code Snippet 8 at the command prompt.

Code Snippet 8:

```
openssl dsaparam -out dsaparam.pem 2048
```

where,

`openssl` - specifies the program for using cryptography functions

`dsaparam` - generates and manipulates DSA parameter files

`out` - generates the output to a file

`dsaparam.pem` - specifies the name of the output file

`2048` - specifies the length of the key

This command generates a 2048 bit long DSA key without using any encrypted key. Figure 10.9 displays the output of the command.

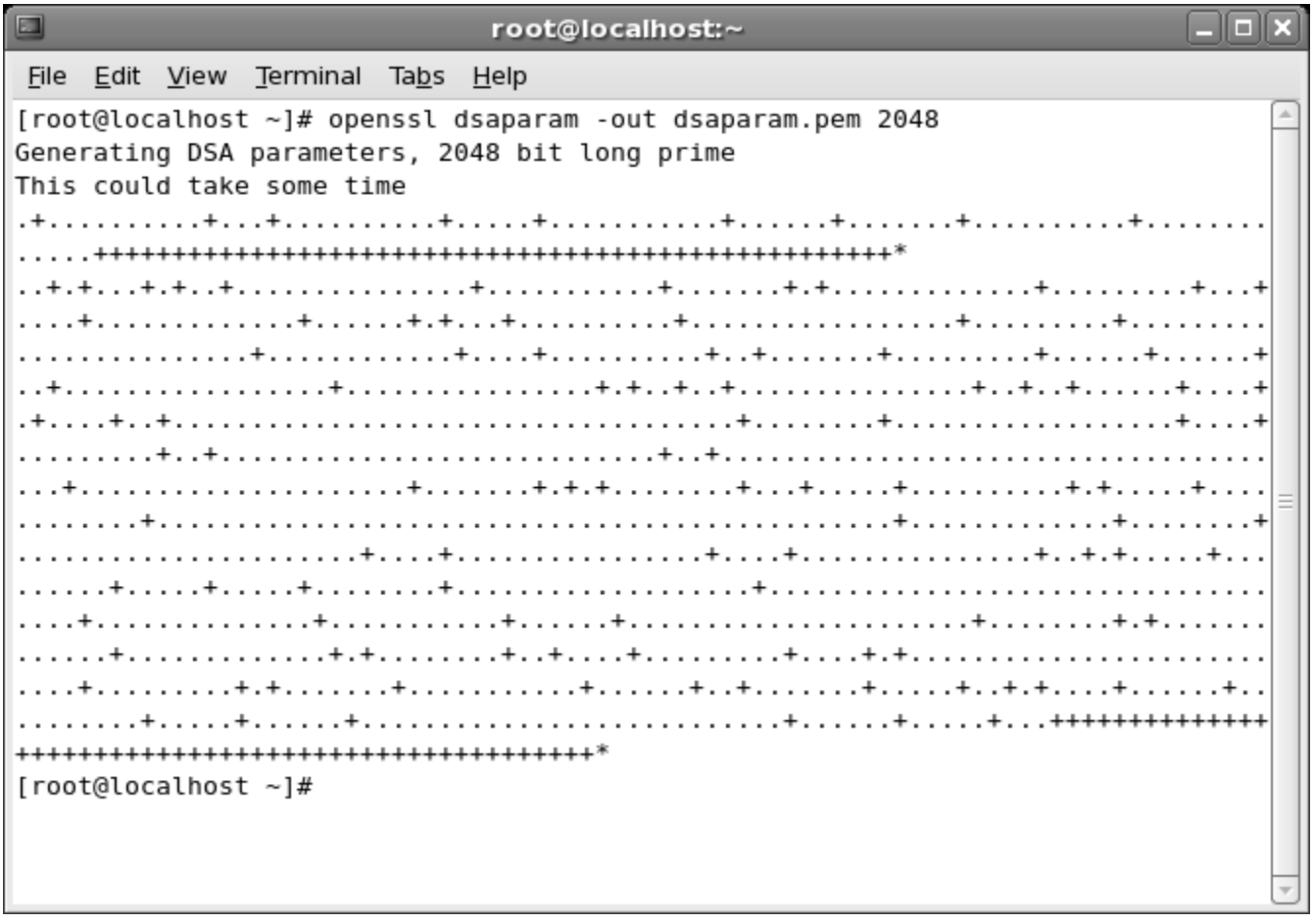


Figure 10.9: Generation of DSA Key for Signing

After generating the DSA key, enter the following command as shown in Code Snippet 9 at the command prompt to generate a key using DES3 encryption.

Code Snippet 9:

```
openssl gendsa -des3 -out privkey.pem dsaparam.pem
```

where,

`openssl` - specifies the program for using cryptography functions

`gendsa` - generates a DSA private key from a DSA parameter file

`-des3` - encrypts the private key with the triple DES cipher

`out` - generates the output to a file

`privkey.pem` - specifies the name of the input file

`dsaparam.pem` - specifies the name of the output file

This command creates a 2048-bit DSA key with encryption and stores it in the file `dsaparam.pem`.

Figure 10.10 displays the output of the command.

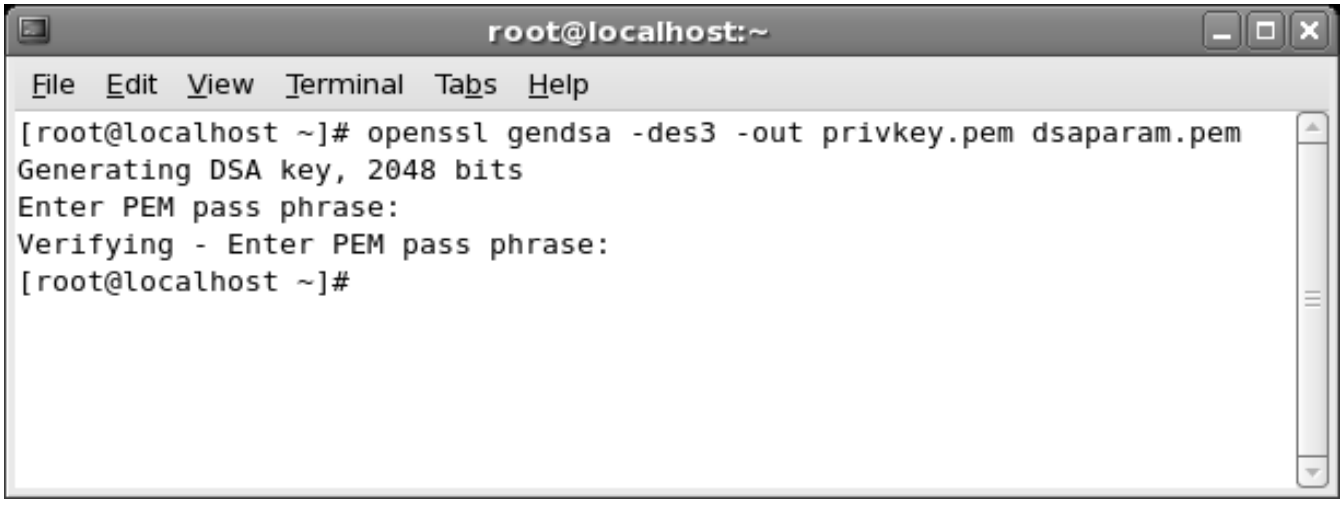


Figure 10.10: Generation of DSA Key

When you install a private key, you must specify the `passphrase` (the protecting password), each time you start Apache server. The `passphrase` maintains the integrity of the certificate, thereby eliminating the risk of misuse. If you want to remove the `passphrase`, then you must ensure that the file permissions are restricted to essential users only. To avoid entering a password each time Apache server starts, prefer creating an unencrypted private key.

Note: The average size of an RSA or a DSA key is 2048 bits. However, for better security, 2048 bits or an even higher size is recommended for these keys. The maximum size is 4096 bits.

To create an unencrypted private key, enter the following command as shown in Code Snippet 10 at the command prompt.

Concepts

Code Snippet 10:

```
openssl genrsa 2048 > www.apachesite.com.key
```

where,

`openssl` - specifies the program for using cryptography functions

`genrsa` - generates an RSA private key

`2048` - specifies the bit size or length of the key

`www.apachesite.com.key` - generates the output to the file

Figure 10.11 displays the output of the command.

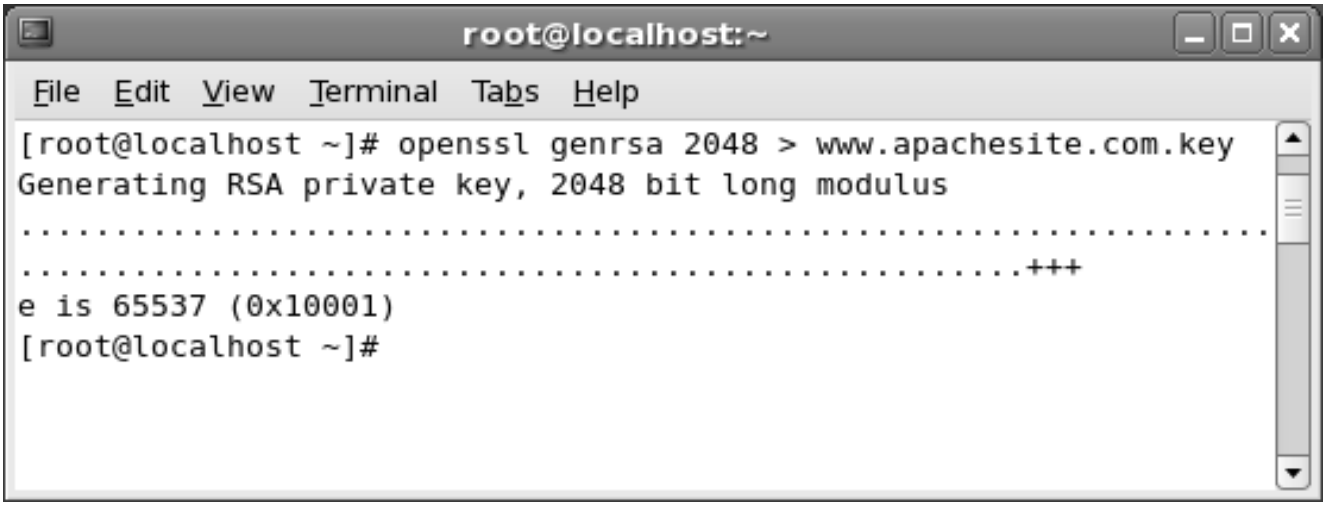


Figure 10.11: Creation of an Unencrypted Private Key

10.6 Creating Certificates

In an online transaction, a certificate validates the authenticity of the two systems that communicate with each other. A certificate is required to validate a private key. To verify the authenticity of a certificate, it has to be digitally signed. A digitally signed certificate cannot be modified. You can create and sign certificate requests by using the OpenSSL utility.

- **Certificate Signing Request -** A Certificate Signing Request (also CSR or certification request) is an application sent from an applicant to a CA to acquire a digital identity certificate.

  The applicant generates a key pair that includes the public key and the private key.

The CSR is then generated that contain information, such as owner name, e-mail address, certificate usage, duration of validity, and public key. The corresponding private key is not included in the CSR. However, the private key is used to digitally sign the request. After verifying the credentials, the CA sends a digitally signed identity certificate to the client along with its private key.

To generate a CSR, enter the following command as shown in Code Snippet 11 at the command prompt.

Code Snippet 11:

```
openssl req -new -key www.mysite.com.key -out www.mysite.com.csr
```

where,

`req` - generates and processes certificate requests

`-new -key www.mysite.com.key` - creates a new certificate and a new private key for www.mysite.com.key

`-out www.mysite.com.csr` - defines the output filename to write to or the standard output

The command reads the CSR, signs it with the key, and generates a certificate.

Concepts

Figure 10.12 displays the output of the command.

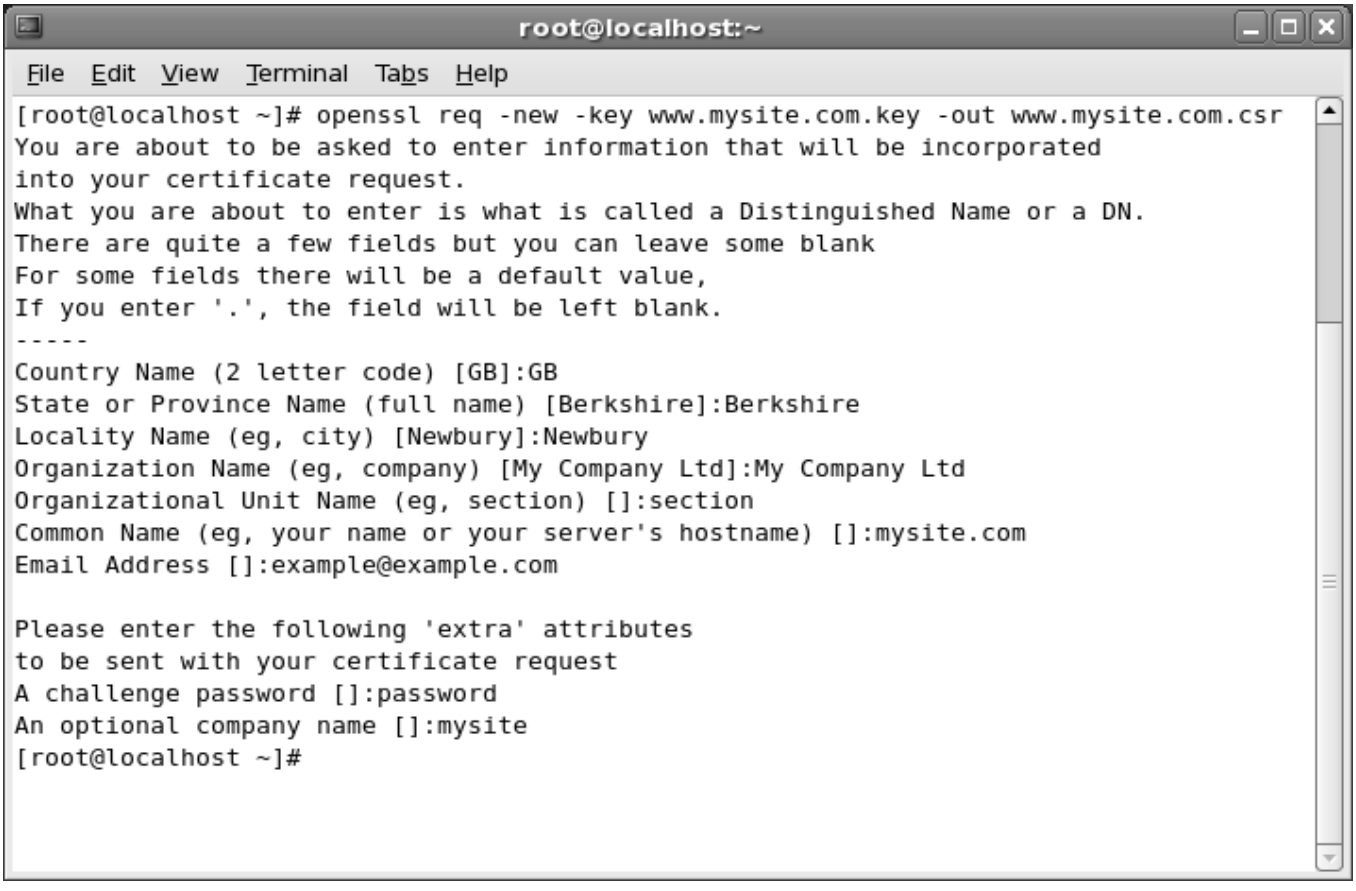


Figure 10.12: Creating a CSR

- **Temporary Certificate** - A self-signed certificate can be created before receiving the digitally signed certificate from CA. This temporary certificate will generate an error in the client browser specifying that the signing CA is unknown and not trusted.

  To create a temporary certificate, after generating a CSR, enter the command as shown in Code Snippet 12 at the command prompt.

  Code Snippet 12:

```
openssl req -x509 -key www.mysite.com.key -in \www.mysite.com.csr -out
www.mysite.com.crt
```

  where,

  `openssl` - specifies the program for using cryptography functions

`req` - generates and processes certificate requests

`-x509` - specifies the certificate signing and display utility

`www.mysite.com.key` - specifies the key to use for creating the certificate

`in` – specifies the name of the input file

`www.mysite.com.csr` - specifies the name of the certificate request

`out` - generates the output to a file

`www.mysite.com.crt` - specifies the name of the output file

Figure 10.13 displays the output of the command.

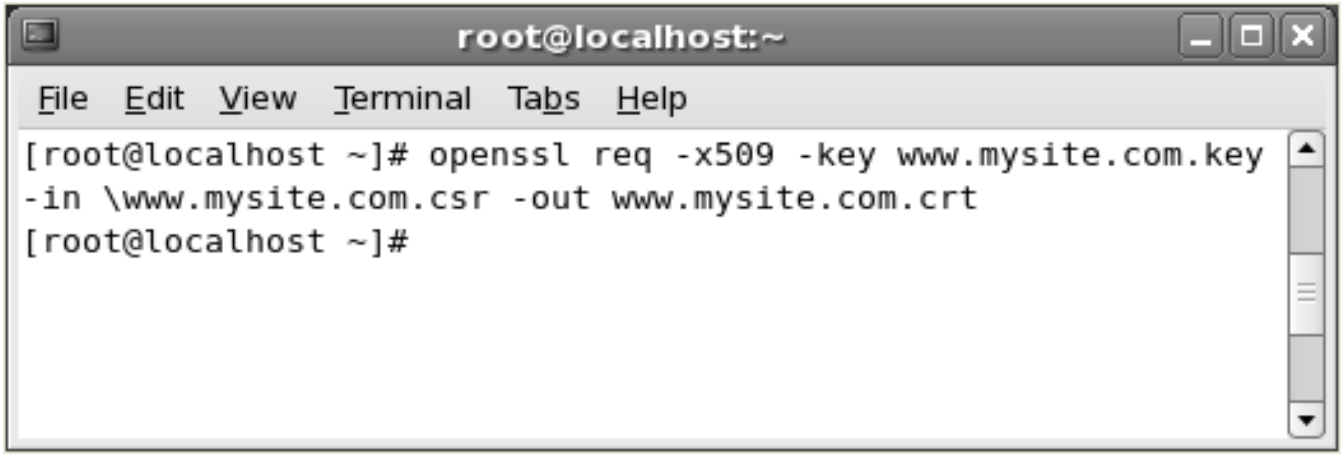


Figure 10.13: Generating a Temporary Certificate in the Command Prompt

To view the contents of temporary certificate file, enter the following command as shown in Code Snippet 13 at the command prompt.

Code Snippet 13:

```
vi www.mysite.com.crt
```

Figure 10.14 displays the output of the command.

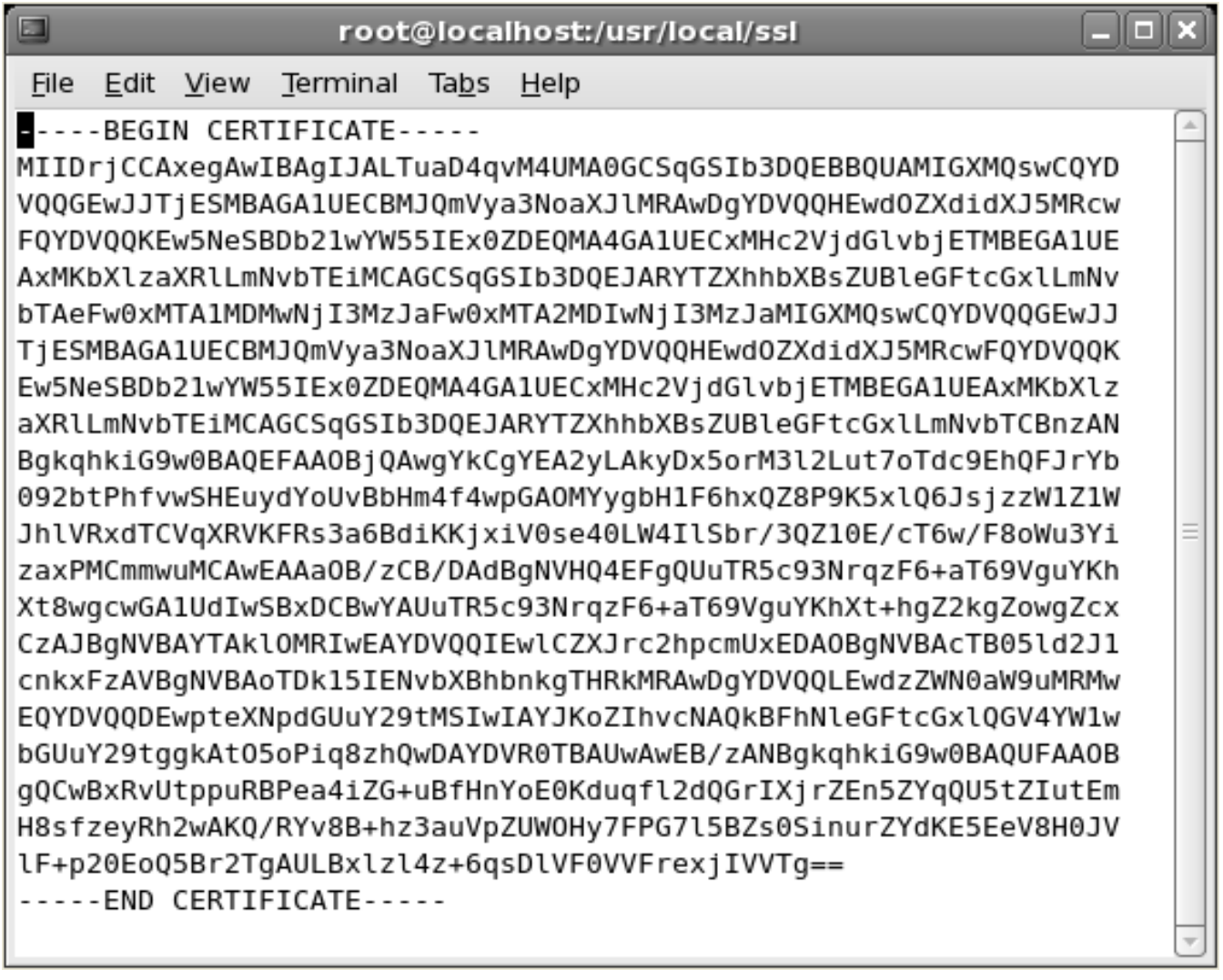


Figure 10.14: Generating a Temporary Certificate

The openssl utility generates the temporary certificate in the `/usr/local/ssl` directory. After generating the certificate, you must test the SSL connection. To test the SSL connection, enter the following command as shown in Code Snippet 14 at the command prompt.

Code Snippet 14:

```
openssl s_client – connect localhost: 443 - state
```

The command displays a list of negotiations between openssl and Apache server that can assist in analyzing problems or debugging.

Figure 10.15 displays the output of the command.

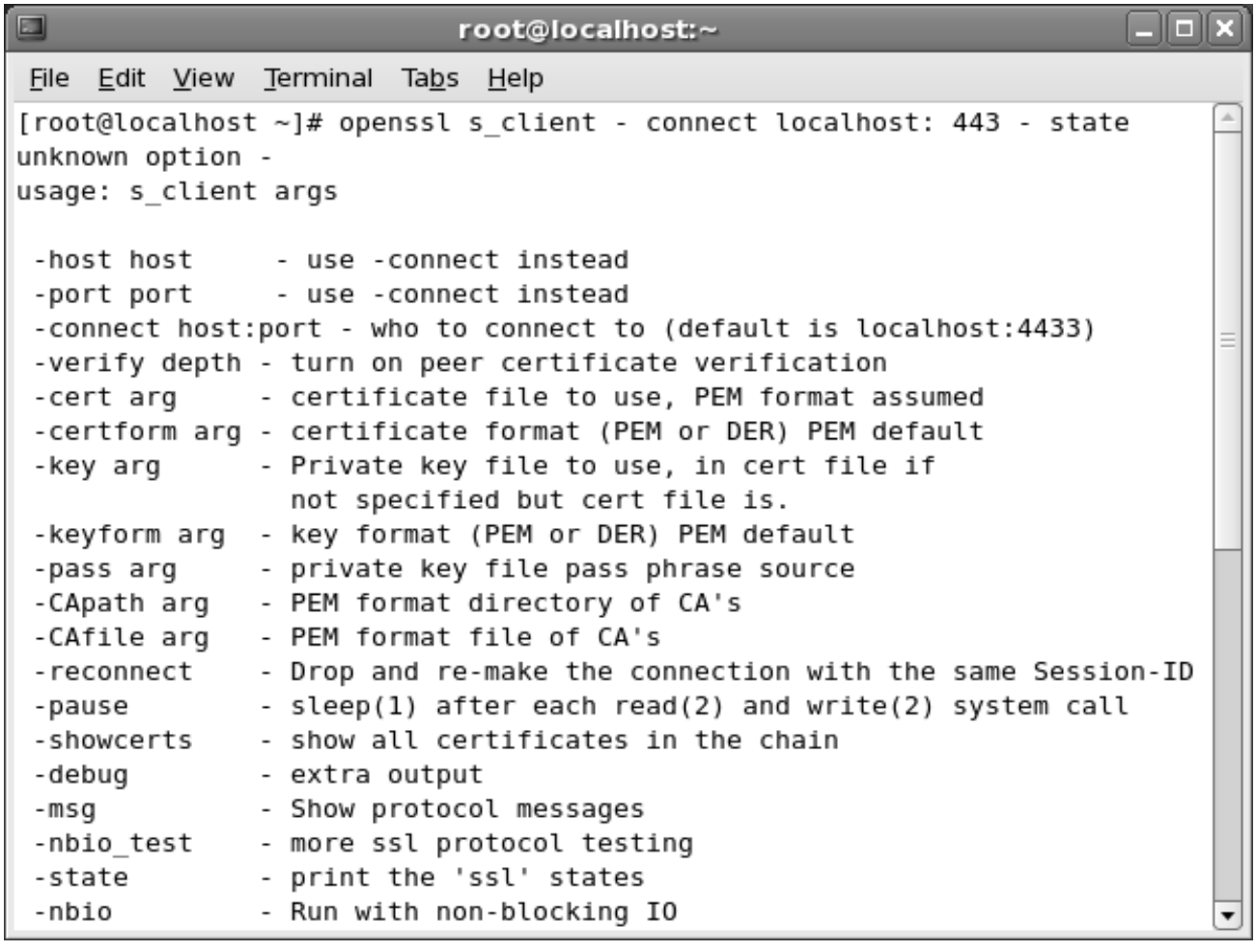


Figure 10.15: Testing the SSL connection

10.7 Acquiring a Signed Certificate

When a client requests a page from a Web server and the server does not have its certificate signed by a recognized CA, the client browser issues a warning message about the site. To prevent this warning message from appearing, the CSR must be signed from a recognized CA.

Verisign and Thawte are two recognized companies that issue digital certificates. To apply for Verisign certificates, you can refer to `http://www.verisign.com/server/`. To apply for `Thawte` certificates, you can refer to `http://www.thawte.com/certs/server/`.

The following information is required to receive a digitally signed certificate:

- Proof of ownership for the organization name mentioned in the CSR
- Proof of ownership of the domain name specified in the CSR
- Proof of authorization of the requesting party
- CSR file

After authentication and verification, the Certification Authority issues a digitally signed certificate.

Summary

- The SSL protocol establishes a secure connection between a client and a Web server.
- Encryption is the method of converting a source message into an unreadable format before transmission. The encrypted message is also known as cipher text.
- The method of converting an encrypted message back into its original form is called decryption.
- In symmetric key system, the sender and the receiver use a common key for encryption and decryption.
- The asymmetric key system uses a pair of keys—a public key known to everyone, and a private key known only to the recipient of the message.
- A message digest is formed by transforming the original message with the help of a one-way hash function.
- A digital signature is created by encrypting the message digest with a private key.
- A certificate is an electronic document used for the identification of a person, a server, or a company.
- Certificates are issued by a third-party trusted agency called as Certification Authority (CA). A certificate generally contains information, such as the distinguished name and public key of the applicant.

Check Your Progress

1. Which of the following encryption technique uses a public key for data encryption and a private key for decryption?

   a. Symmetric key encryption

   b. Asymmetric key encryption

   c. Message Digest

   d. One-way hash

2. Which of the following component is generated by encrypting the one-way hash with a private key?

   a. Certificate

   b. Digital Certificate

   c. Message Digest

   d. CSR

3. Which directive is used to enable the SSL protocol?

   a. `SSLProtocol`

   b. `SSLEngine`

   c. `SSLCertificateFile`

   d. `SSLDisable`

Check Your Progress

4. Which utility provides the required cryptographic support for SSL in Apache Web Server?

   a. `mod_ssl`

   b. `httpd`

   c. `openssl`

   d. `mod_alias`

5. On installing a private key, you must generate a _____.

   a. Certificate

   b. Certificate Signing Request

   c. Public Key

   d. Message Digest

6. Which command will you use to generate a DSA key using the OpenSSL utility?

   a. `openssl genrsa -des3 -out privkey.pem 2048`

   b. `chmod 400 www.apachesite.com.key`

   c. `openssl gendsa -des3 -out privkey.pem dsaparam.pem`

   d. `openssl genrsa 1024 > www.apachesite.com.key`

Session

11 Secure Sockets Layer (Lab)

Objectives

At the end of this session, the student will be able to:

- *Install the OpenSSL utility.*
- *Install the private key.*
- *Configure Apache for the SSL protocol.*
- *Create a Certificate Signing Request for the Apache server.*
- *Create a Temporary Certificate for the server.*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Install the OpenSSL Utility

The Secure Sockets Layer (SSL) protocol enables you to establish a secure connection between a client and a Web server. The OpenSSL utility provides the cryptographic support that is necessary for implementing the SSL protocol in Apache server. The OpenSSL utility is available for download at `http://www.openssl.org`. It is a free utility.

To install the OpenSSL utility, perform the following steps:

1. **Logon to Linux as `root` in the GNOME environment.**
2. **Download the `openssl-1.0.0d.tar.gz` file from `http://www.openssl.org/source/` and save it in the `home` folder.**
3. **Right-click the `openssl-1.0.0d.tar.gz` file and select Extract Here. The contents are extracted in a folder named `openssl-1.0.0d` under the current directory.**
4. **Right-click the `openssl-1.0.0d` folder and select Open In Terminal. The Terminal Window is displayed.**

Figure 11.1 displays the Terminal Window.

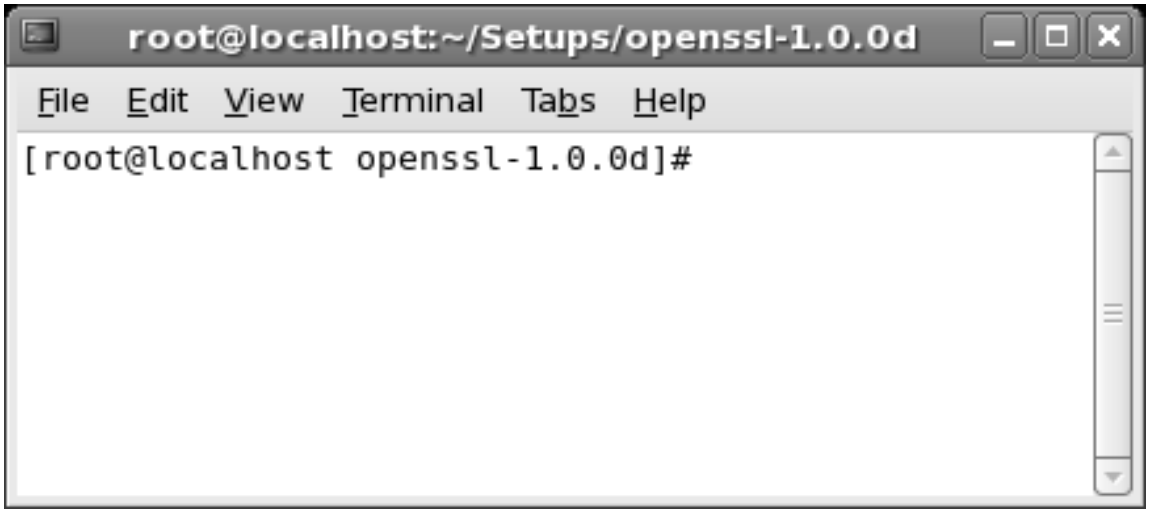


Figure 11.1: Terminal Window

5. **To configure the build process for the OpenSSL utility, enter the following command at the command prompt:**

```
./config
```

Figure 11.2 displays the output of the command.

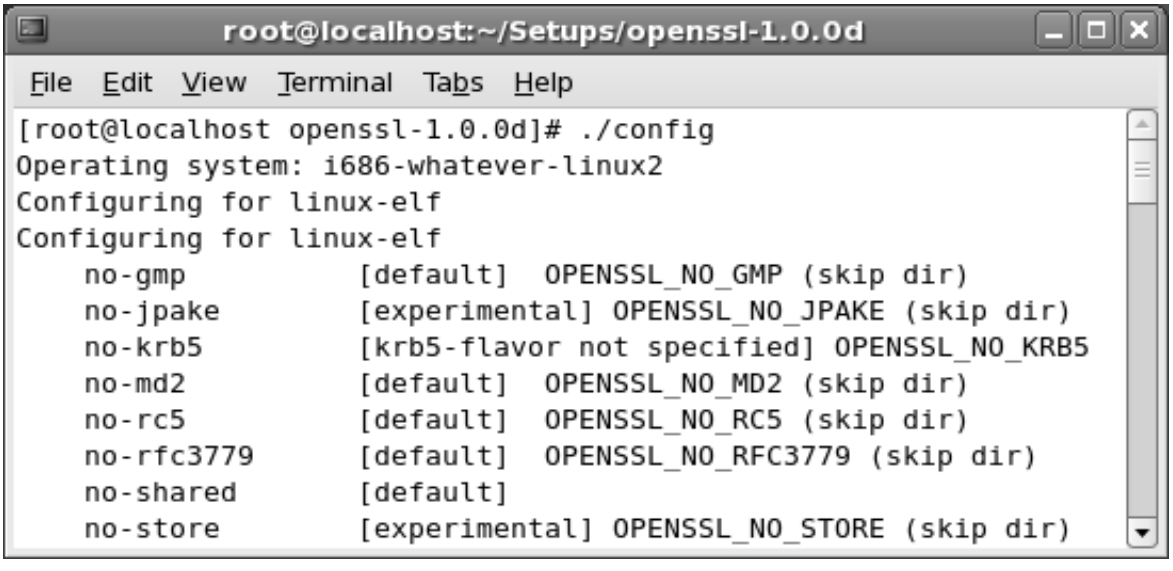


Figure 11.2: config Command

6. **To build the OpenSSL utility, enter the following command at the command prompt:**

```
make
```

Figure 11.3 displays the output of the command.

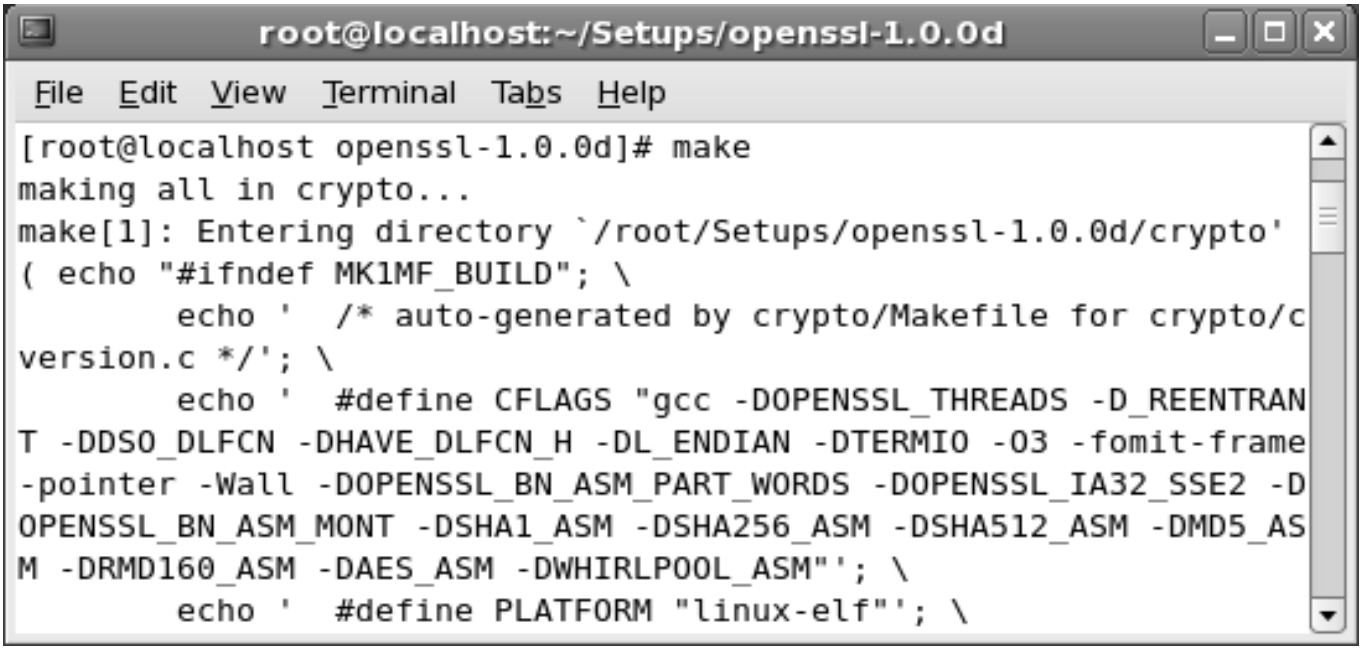


Figure 11.3: make Command

7. **To test the OpenSSL utility, enter the following command at the command prompt:**

```
make test
```

8. **To install the OpenSSL utility, enter the following command at the command prompt:**

```
make install
```

Figure 11.4 displays the output of the command.

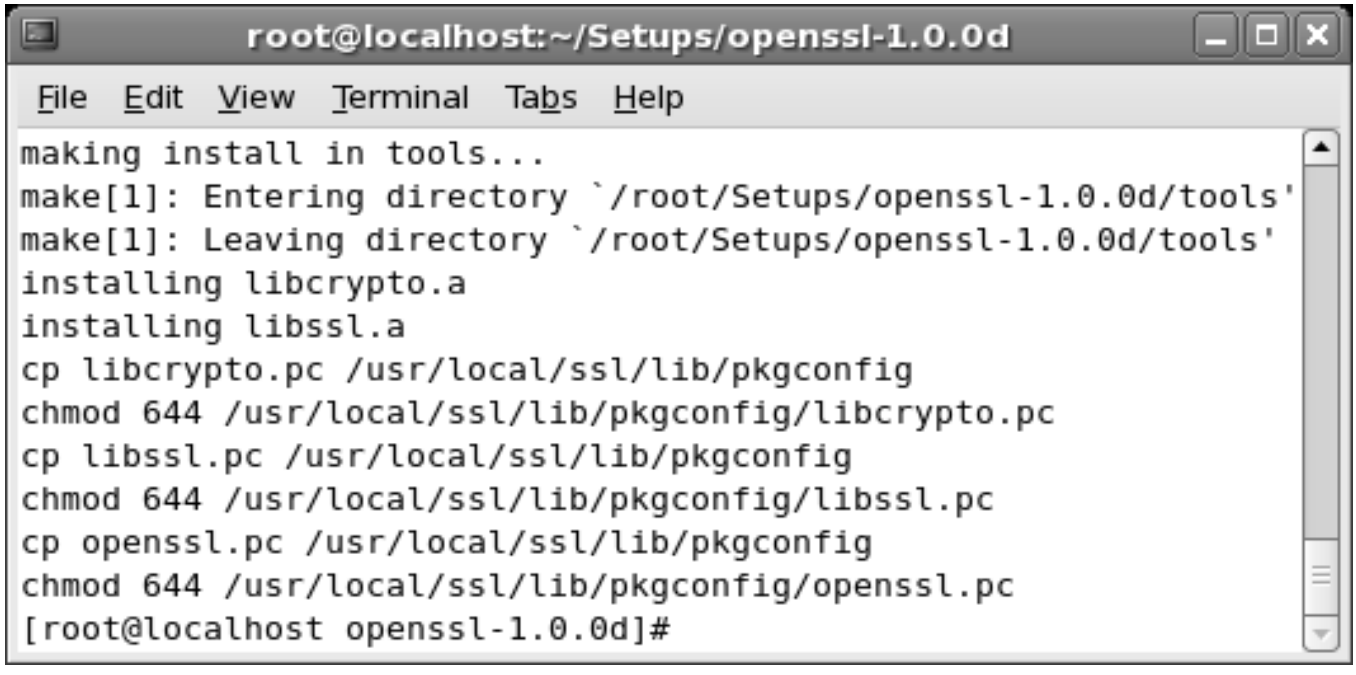


Figure 11.4: make install Command

After installing the OpenSSL utility, you must configure Apache server to include the SSL security. The source code of Apache is required to configure SSL.

9. **Right-click the `httpd-2.2.17` folder and select Open In Terminal. The Terminal window is displayed.**

10. **To configure `mod_ssl`, enter the following command at the command prompt:**

```
./configure --enable-ssl
```

Figure 11.5 displays the output of the command.

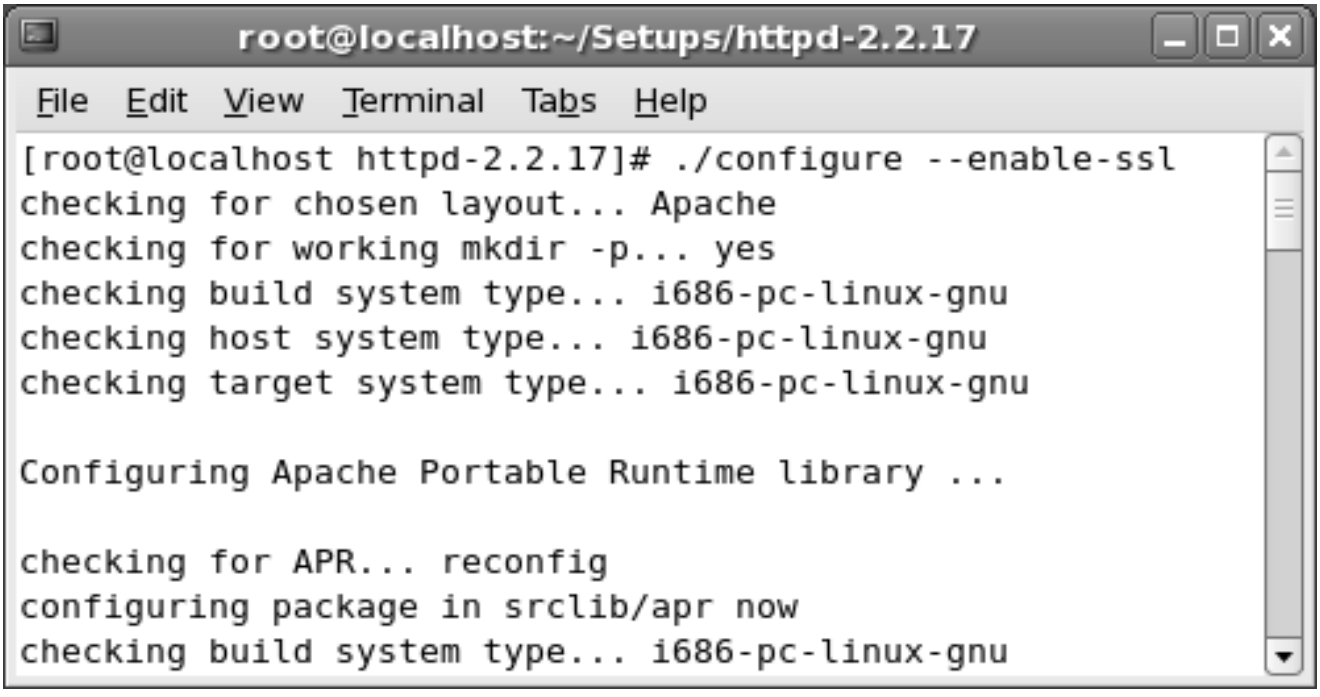


Figure 11.5: Enabling SSL in Apache

Note: The configure command compiles and installs required files for the SSL utility to the respective directories. Hence reinstallation of Apache Web server is not required.

Installing a Private Key

Public key cryptography uses two different keys for encryption and decryption. In this system, each entity must possess a pair of keys, a public key and a private key. The public key is transmitted on the network, but the private key is valid only to the machine where the private key is generated. The public or the private keys can be used for encryption. A message encrypted with a public key can be decrypted using the corresponding private key. To transmit data securely, you must encrypt the data using the receiver's public key and the receiver must decrypt the data using the private key.

To install a private key, perform the following steps:

1. **To browse to the ssl directory, enter the following command at the command prompt:**

```
cd /usr/local/ssl
```

Figure 11.6 displays the output of the command.

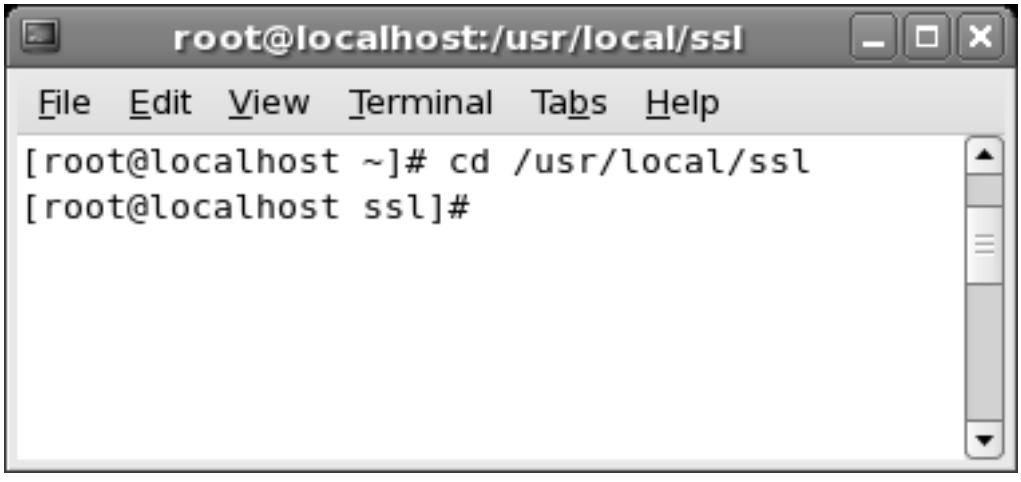


Figure 11.6: ssl Directory

2. **To create an unencrypted private key for the server, enter the following command at the command prompt:**

```
openssl genrsa 1024  > www.mysite.com.key
```

Figure 11.7 displays the output of the command.

Figure 11.7: Creating Unencrypted Private Key

Note: The private key is generated in the `/usr/local/ssl` folder.

3. **To view the generated private key file, enter the following command at the command prompt:**

```
vi www.mysite.com.key
```

Figure 11.8 displays the private key file.

Figure 11.8: Contents of the Private Key File

Creating a Certificate Signing Request for the Apache Web Server

John has launched a new Web site, www.mysite.com, and he wants to acquire a Certificate Signing Request (CSR) from a Certification Authority (CA). A certificate signing request is the process of acquiring a digital identity certificate from certificate authority. John sends a request to CA along with its public and private key, but the private key is not disclosed to the CA. A certificate generally contains information, such as the distinguished name, public key of the applicant, signature of the CA, and the period of validity. The CA sends an identity certificate that is digitally signed with the private key of the CA after authentication and verification.

1. **To create a Certificate Signing Request (CSR), enter the following command at the command prompt:**

```
openssl req -new -key www.mysite.com.key -out www.mysite.com.csr
```

2. **Enter the required information.**

Figure 11.9 displays the information required for CSR.

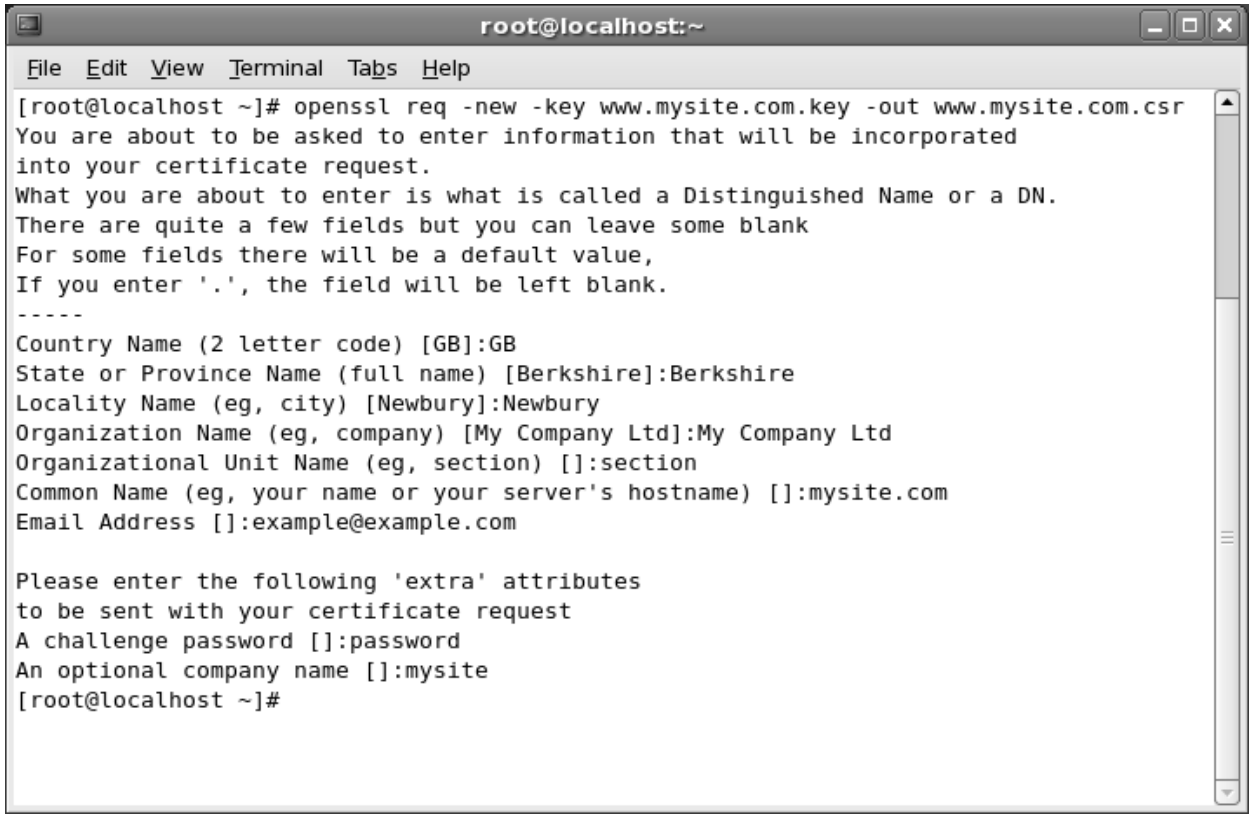


Figure 11.9: Creating a CSR

3. **To view the CSR file, `www.mysite.com.csr`, enter the following command at the command prompt:**

```
vi www.mysite.com.csr
```

Figure 11.10 displays the CSR file.

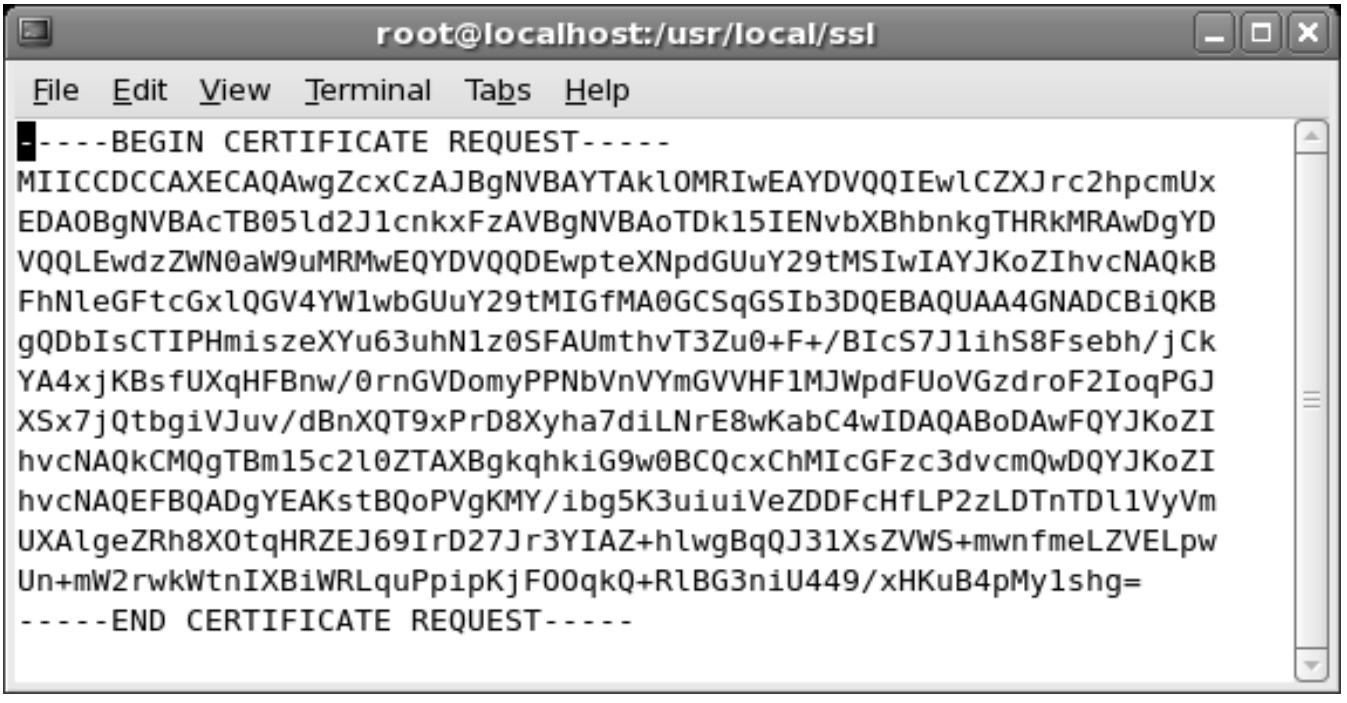


Figure 11.10: CSR File

4.	**Enter `:q` to exit the `vi` editor.**

Create a Temporary Certificate for the Server

A temporary certificate assigns a digital signature to the data that is transmitted over the Internet. While creating a temporary certificate, a public and private key is generated. The private key is not disclosed to the CA. The private key is used to digitally sign the request. The CA sends an identity certificate that is digitally signed with the private key of the certificate authority after authentication and verification. The temporary certificate is generated in the `/usr/local/ssl` directory.

1.	**To create a temporary certificate for the above generated private key, enter the following command at the command prompt:** `openssl req -x509 -key www.mysite.com.key -in \www.mysite.com.csr -out www.mysite.com.crt`

Figure 11.11 displays the output of the command.

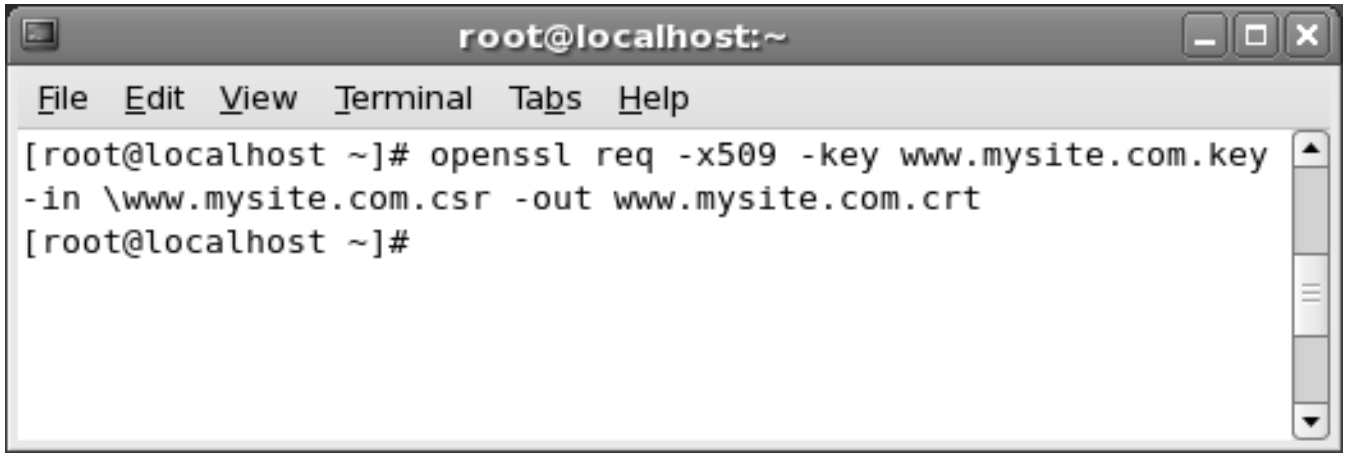


Figure 11.11: Creating a Temporary Certificate

2. **To view the contents of temporary certificate file, enter the following command at the command prompt:**

```
vi www.mysite.com.crt
```

Figure 11.12 displays the output of the command.

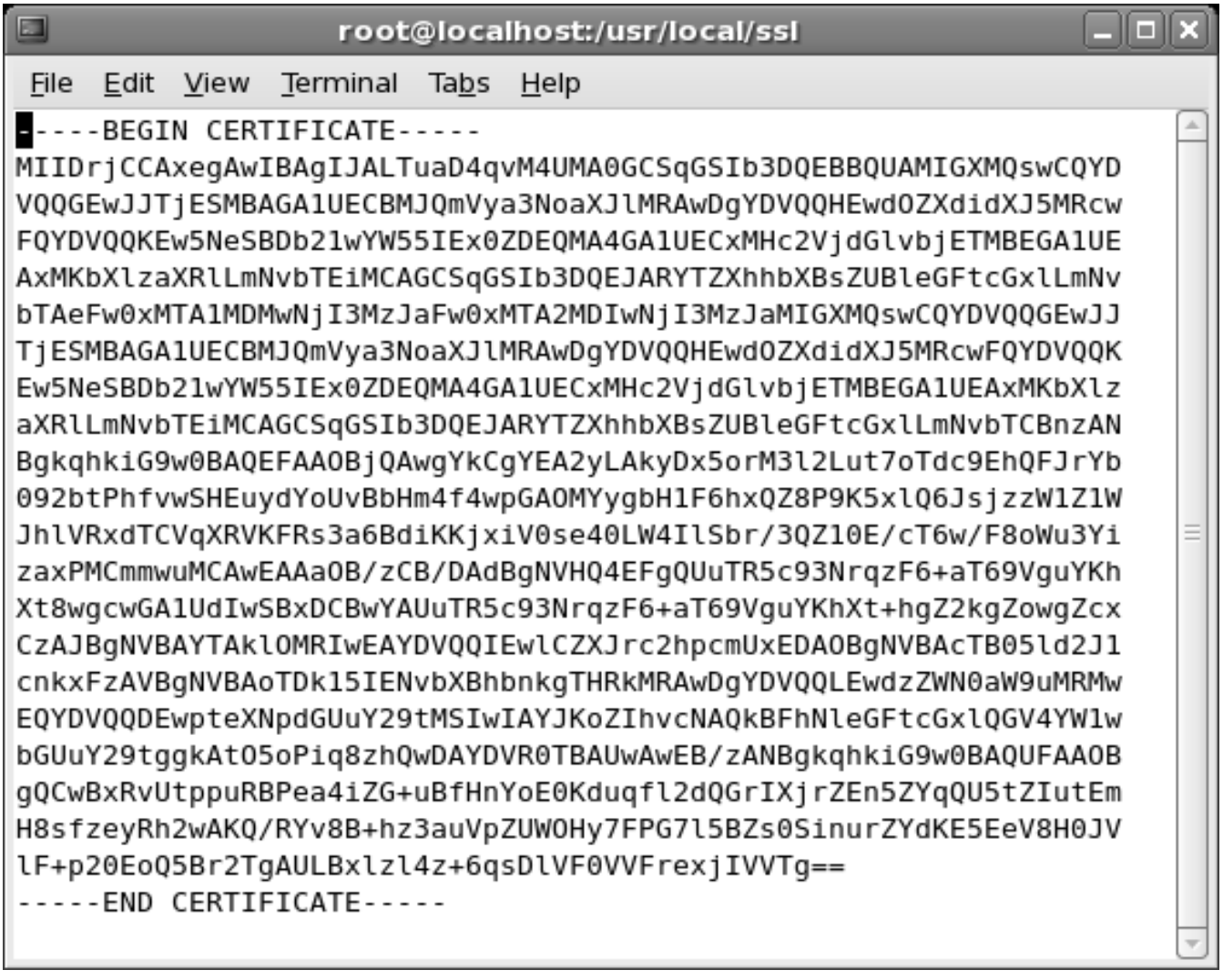


Figure 11.12: Temporary Certificate

Configuring mod_ssl in Apache

The SSL directives must be included in the `httpd.conf` file to implement the SSL protocol.

To include the SSL directives in the `httpd.conf` file, perform the following steps:

1. **To browse to the `conf` directory, enter the following command at the command prompt:**

   ```
   cd /usr/local/apache2/conf
   ```

2. **To open the `httpd.conf` file, enter the following command at the command prompt:**

   ```
   vi httpd.conf
   ```

3. **Enter the following SSL directives in the `httpd.conf` file:**

   ```
   LoadModule ssl_module modules/mod_ssl.so
   SSLEngine ON
   SSLCertificateFile   /usr/local/ssl/www.mysite.com.key
   SSLCertificateKeyFile  /usr/local/ssl/www.mysite.com.crt
   ```

Figure 11.13 displays the `httpd.conf` file.

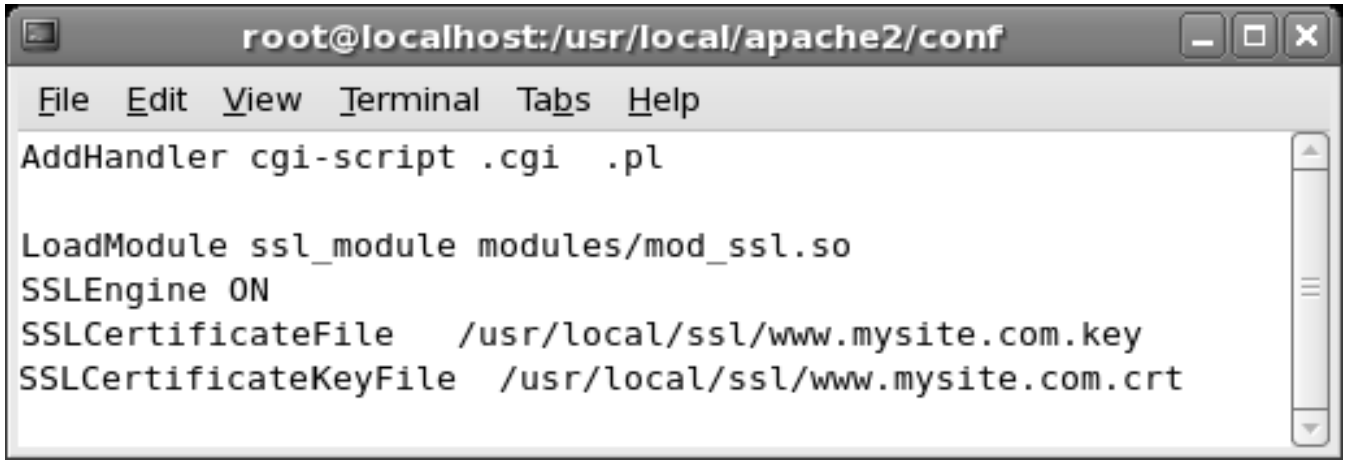


Figure 11.13: httpd.conf File

4. **Press 'Esc' and enter `:wq` to save and exit the `vi` editor.**

5. **To start the Apache server, enter the following command at the command prompt:**

```
/usr/local/apache2/bin/apachectl start
```

6. **Open the Mozilla Firefox Web browser and access the following page:**

```
https://localhost/
```

Figure 11.14 displays the output.

Figure 11.14: Localhost Page

Do It Yourself

1. Configure Apache for using SSLv3 protocol.

2. Configure Apache such that the client may present a certificate during the handshake sequence.

Session

12 Dynamic Pages

Concepts

Objectives

At the end of this session, the student will be able to:

- *Explain the working of dynamic pages.*
- *Explain the working of CGI.*
- *Describe the configuration process of Apache for supporting CGI.*
- *Explain Server Side Includes.*
- *Describe the configuration process of Apache for Server Side Includes.*
- *Describe the configuration process of Apache for Directory Indexing.*

12.1 Introduction

An interactive Web site accepts data from its users. This data is processed at the server end and dynamic information is generated and returned to the users. CGI is the protocol that enables server-side processing of this data.

In this session, you will learn about the concept of dynamic pages and CGI. You will also learn about the directives that configure Apache for supporting CGI. In addition, you will learn the concept of Server Side Includes (SSI) and the directives that configure Apache for working with SSI and directory indexing.

12.2 Introduction to Dynamic Pages and Dynamic Web Modules

A Web site is a collection of Web pages. When a client requests a page, Apache Web server serves the requested page from the document root folder. The contents of a static page remain unchanged. On the other hand, a dynamic page displays different information at different times. A dynamic page is generated in response to the user activity and requires a higher download time as compared to a static page. The content of a dynamic Web page changes with time, user interaction, context, or a combination of any of these parameters. For example, a search engine accepts keywords in an HTML format and generates a dynamic page depending on the keywords entered.

The modular design of Apache server imparts power and flexibility to Web administrators and developers. Different modules are available to implement different server features. Due to this modular design, you can add a server feature when required without the need to reinstall Apache server. A module can be loaded into Apache server either statically or dynamically. The static modules are loaded in the compiled httpd daemon.

Some of the examples of static Apache modules are `mod_dir.c`, `mod_cgi.c`, `mod_imap.c`, or `mod_actions.c`. A dynamic module is an Apache module that can be loaded when required by making modifications to the Apache configuration file, `httpd.conf`. Some of the examples of dynamic modules in Apache are `mod_perl`, `mod_php`, or `mod_dav`.

12.3 Common Gateway Interface

The CGI is a mechanism that enables a Web server to process dynamic page requests. It is a protocol that defines the communication standard between the Web server and an external program. In other words, it acts as an interface between the Web server and the programs that you write. The external program is a CGI program that is executed on the server to display dynamic information.

For example, you can write a CGI program to process the data submitted in an HTML form. You can also write CGI programs for databases, that can store, retrieve, and display information on a Web page.

A CGI program, also known as a CGI script, can be written in many languages, such as Perl, Python, PHP, or C. These programs are stored in a directory called `cgi-bin` present on the Web server.

Consider an example, where you request for a Web page by clicking on a hyperlink. In response to the request, the server sends back the requested page. However, the Web server typically passes the form information to a small application program that processes the data and sends back a confirmation message. The response sent back to the client can be an HTML document, GIF files, video clips or any other data that can be viewed by the client in the browser. CGI is a specification that enables information to be transmitted between a CGI program and the World Wide Web server. A CGI program is a script that accepts and returns data and conforms to the CGI specification.

CGI can be used to:

- Create interactive Web documents
- Insert data into databases
- Use a given database of information to generate pages
- Resolve form data
- Respond to a form entry with e-mail or another Web page
- Set and read cookies
- Calculate hit counts
- Monitor Web traffic

Concepts

12.3.1 Writing a CGI Program

A CGI program can be written to accept data from the user, process the data, and interact with a database. This program can be written using any languages, such as Perl, C, or PHP. The output of a CGI program is HTML content preceded by MIME type header, an HTTP header that informs the client about the type of content received in the CGI output.

Code Snippet 1 displays an example where a sample form accepts user data in the HTML file named `form1.html`.

Code Snippet 1:

```
<html>
  <head>
    <title>Sample Form</title>
  </head>
  <body><h1>Enter Your Details....</h1>
    <form action=”perl_script.cgi” method=post>
      Enter your name:
      <input type=”text” name=”Name” size=30 maxlength=50><p>
      Enter your email-id:
      <input type=”text” name =”Email” size=30 maxlength=50> <p>
      <input type =”submit” value = “Submit”>
      or
      <input type=”reset” value=”Reset”> <p>
    </form>
  </body>
</html>
```

Figure 12.1 displays the output of the HTML file.

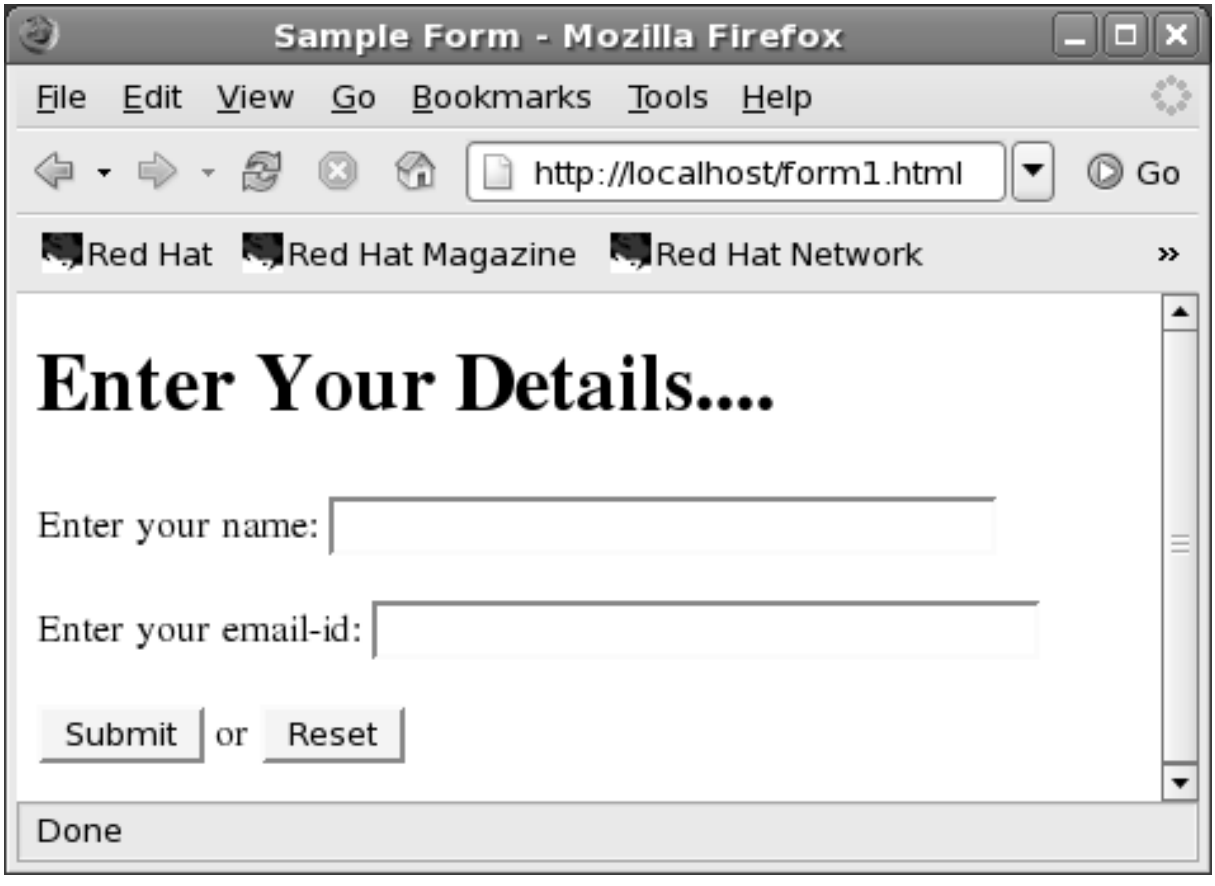


Figure 12.1: HTML Form

In figure 12.1, the HTML form accepts username and e-mail address. On clicking the submit button, a Perl script will be used to process and display the data entered by the user. The CGI scripts must be saved in the `/usr/local/apache2/cgi-bin` directory to be executed on the Web browser. In addition, the `httpd.conf` file must contain the following code to enable Apache server to execute CGI scripts from the `cgi-bin` directory.

```
ScriptAlias /cgi-bin/ /usr/local/apache2/cgi-bin/
```

To execute CGI scripts outside the `cgi-bin` directory, the `Options` directive can be used to enable execution of CGI scripts. Accordingly, the `httpd.conf` file must contain the code as shown in Code Snippet 2.

Concepts

Code Snippet 2:

```
<Directory /usr/local/apache2/htdocs/somedir>
Options +ExecCGI
</Directory>
```

where,

`somedir` - defines the directory name for executing CGI scripts

Code Snippet 3 displays a sample Perl script file named `perl_script.cgi` that displays the data entered by the user.

Code Snippet 3:

```
#!/usr/bin/perl
# perl script to process a user form
print "Content-type: text/html\n\n";

&Process;

print "<title>The Response</title><h1>Perl Output</h1><hr>";
print "The data entered in the form is as follows:<ul>";
foreach $key (keys %in)
{
     print "<li>$key: $in{$key}";
}
print "</ul>";

sub Process
{
    local (*in) = @_ if @_;
    local ($i, $key, $val);

    if ( $ENV{'REQUEST_METHOD'} eq "GET" ){
      $in = $ENV{'QUERY_STRING'};
   }
        elsif ($ENV{'REQUEST_METHOD'} eq "POST") {
               read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
        }
```

```
@in = split(/&/,$in);
foreach $i (0 .. $#in)
{
  $in[$i] =~ s/\+/ /g;
($key, $val) = split(/=/,$in[$i],2);
     $key =~ s/%(..)/pack("c",hex($1))/ge;

     $val =~ s/%(..)/pack("c",hex($1))/ge;
     $in{$key} .= "\0" if (defined($in{$key}));

     $in{$key} .= $val;

 }
      return length($in);
}
```

The data entered in the form is submitted to the server using URL-encoding. In URL-encoding, the entire form data is included in a single string in a `name = value` format separated by `&s`. Any space in a name or value is encoded with a `+` symbol. For example, in the form created earlier, if you enter the name as `SampleName` and the e-mail address as `sample@trial.com`, a part of the encoded URL string would contain the data as `Name=SampleName&Email=sample@trial.com`.

In the Perl script, the first statement specifies the location of the Perl interpreter that must execute the script. A `#` symbol is used as a comment symbol in Perl. The `Process` function reads the URL-encoded data and decodes it to extract the form data. The `split` function splits the received string based on `&` symbol to generate a list of `name=value` pairs. Next, all the `+` symbols existing in a name or value are replaced into spaces. The `name=value` pairs are further split on a = and stored in separate variables. Finally, the form data is sent back to the client browser to be displayed to the user.

Figure 12.2 displays the output of the Perl script.

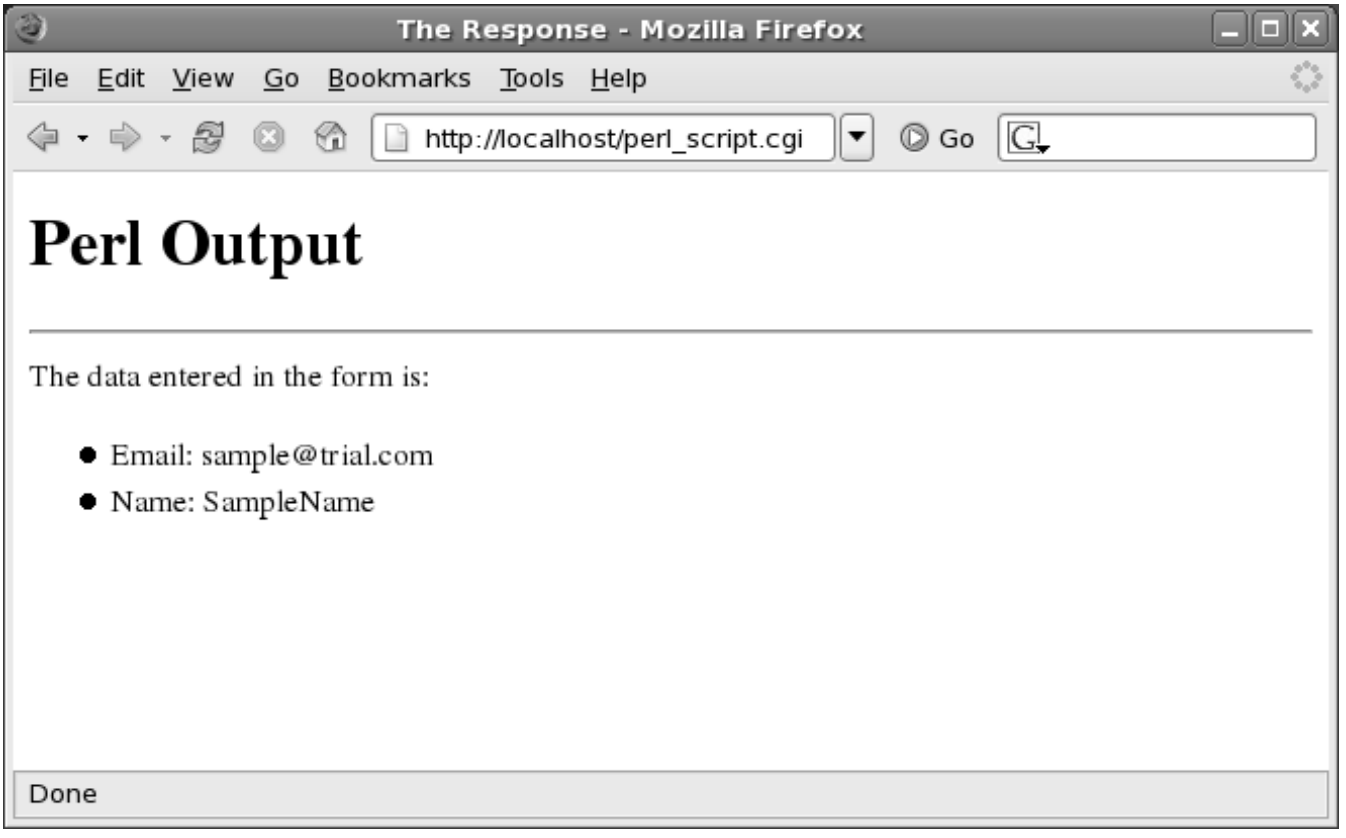


Figure 12.2: Output of the Perl Script

12.4 Configuring Apache for CGI

Apache server must be configured to support CGI. You must include the directives that enable CGI in Apache in the `httpd.conf` file. Once you configure Apache server for CGI support, it executes CGI scripts to generate dynamic information.

The directives that enable you to configure Apache server for CGI are as follows:

- **ScriptAlias directive**

  The `ScriptAlias` directive defines a directory for executing CGI programs. When the client requests Apache server for a resource from the directory specified in the `ScriptAlias` directive, Apache processes these resources as a CGI script. The `mod_alias` module implements this directive and you can use it for directories that are outside the `documentroot`.

The syntax of the `ScriptAlias` directive is as follows:

```
ScriptAlias URL-path file path | directory path
```

where,

`URL-path` - specifies a valid URL

`file path` - defines the location of the file

`directory path` - specifies the location of the directory

For example, to specify the directory that contains `cgi-scripts`, enter the code as shown in Code Snippet 4 in the `httpd.conf` file.

Code Snippet 4:

```
ScriptAlias /cgi-prog/ /usr/local/apache2/cgi-prog/
```

where,

`ScriptAlias` - specifies the directive

`/cgi-prog/` - defines the request to be processed by the server

`/usr/local/apache2/cgi-prog/` - defines the location of the requests

The `ScriptAlias` directive considers everything under the URL prefix as a CGI program. Therefore, in Code Snippet 3, the server executes the CGI program and returns the output to the client browser. For example, if the client requests from `http://www.myserver.com/cgi-prog/myfile.pl`, the server will execute the file `/usr/local/apache2/cgi-prog/myfile.pl` and return its output to the client.

- **Options directive**

The `Options` directive implemented by the core module of the Apache server enables you to specify the features applicable to a directory. It can be used with the `Directory` directive to configure Apache server for CGI execution. The syntax for using this command is as follows:

```
Options [+|-]option [[+|-]option] ...
```

where,

`Options` - specifies the directive

`option` - defines the directive to be processed

Table 12.1 lists the different options and their corresponding functions.

Options	Functions
`ExecCGI`	Enables the execution of CGI scripts
`FollowSymLinks`	Configures the Apache Web server to follow symbolic links. A symbolic link is a special file in Linux that contains the pathname to another file
`Includes`	Enables Server-side includes
`IncludesNOEXEC`	Enables Server-side includes but disables the #exec cmd and #exec cgi commands
`Indexes`	Returns a formatted listing of the directory when a directory does not contain a DirectoryIndex directive in that directory (The DirectoryIndex directive defines a list of files that must be searched when a client requests a directory)
`MultiViews`	Enables content-negotiated multiviews. Content negotiation enables Apache server to select the representation of a resource based on the media type, languages, character set, and encoding information provided by the browser
`SymLinksIfOwnerMatch`	Enables Apache server to follow symbolic links only for the target file where the user ID of the file is the same as the link
`None`	Disables all options
`All`	Enables all the options except for MultiViews

Table 12.1: Different options and their functions

For example, to enable CGI execution in a directory named `mydir`, enter the code, as shown in Code Snippet 5, in the `httpd.conf` file.

Code Snippet 5:

```
<Directory /usr/local/apache2/htdocs/mydir>
Options +ExecCGI
</Directory>
```

Before the execution of CGI files, the server must identify CGI files. You can use the `AddHandler` directive to execute specific files as CGI programs.

The `httpd.conf` file must contain the code as shown in Code Snippet 6 to execute Perl scripts as CGI programs:

Code Snippet 6:

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

- **SetHandler directive**

The `SetHandler` directive implemented by the core module defines a handler for a particular directory. In other words, it means that all files which are matching should be processed by a handler. You must specify it along with the `ExecCGI` inside a directory to enable CGI execution.

The syntax of the `SetHandler` directive is as follows:

```
SetHandler handler_name|None
```

where,

`handler_name` - specifies a handler type.

`None` - ignores the instructions specified earlier in the `SetHandler` directive.

Handlers are provided to handle different file types in different ways. For example, to handle CGI scripts, the handler name must be set to `cgi-script`. The `SetHandler` directive overrides default handlers therefore, it ignores the activities, such as, interpreting URLs ending in a slash(`/`) as directories.

- **AddHandler directive**

The `AddHandler` directive, implemented by the `mod_mime` module, enables the administrator to specify the filename extensions to the specified handler. You must use it along with `Options` directive inside a directory section. The syntax of the `AddHandler` directive is as follows:

```
AddHandler handler_name extension
```

where,

`handler_name` - specifies the handler type.

`extension` - includes a list of filename extensions to be treated by the specified handler It is case-insensitive. The leading dot is optional.

For example, to enable CGI execution in a directory called `cgidir`, enter the code, as shown in Code Snippet 7, in the `httpd.conf` file.

Code Snippet 7:

```
<Directory "/usr/local/apache2/cgidir">
  Options  +ExecCGI
  SetHandler cgi-script
  AddHandler cgi-script  .cgi  .pl  .py
</Directory>
```

The `AddHandler` directive enables CGI, Perl, and PHP scripts to be executed.

12.5 Server Side Includes

SSI are directives placed in HTML pages, which are executed on the server while the pages are being served. SSI is a server-side scripting language that adds small pieces of dynamic content, such as current date and time, document name, and content length to the existing HTML pages. SSI appends this content without processing the entire page through any dynamic technology, including CGI scripts. SSI is helpful only if a small section of the page needs to be modified, if a major portion of your page is being generated at the time it is served, SSI is not a feasible solution.

You can decide when to use SSI on the basis of two parameters—what percentage of the page is static, and what percentage needs to be computed again each time the page is served.

The SSI documents have an extension of `.shtml`. The `mod_include` module implements the SSI feature on Apache server.

SSI is used to:

- Provide interactivity to a Web site
- Help in maintaining the Web site
- Add a common piece of code, such as a page header, page footer, and navigation menu throughout a site. This information can be used for site navigation or copyright notices.
- Control directives in SSI that can include conditional navigation menus.

Features of SSI are as follows:

- Supports only one data type: text

- Has a simple design
- Is easy to learn
- Is used on Web servers, such as Apache, lighttpd, and IIS

Note: lighttpd is an open source, standards-compliant, secure, and flexible Web server especially designed for speed-critical environments.

12.5.1 Configuring Apache for SSI

To configure Apache to permit SSI, you must include the following directive either in the `httpd.conf` file, or in a `.htaccess` file.

```
Options +Includes
```

This directive enables Apache Web server to incorporate SSI provided by the `mod_include` directive. Majority of the configurations contain multiple `Options` directives that can override each other. Therefore, to make sure that your `Options` directive is evaluated at the end, you should apply the `Options` directive to the specific directory where you want SSI enabled.

You have to inform the server about the files that need to be parsed. You can do this in two different ways:

- Use directives to notify Apache server to parse any file with a particular file extension, such as `.shtml`. The directive is as follows:

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

The disadvantage of working with this approach is when you want to add a SSI directive to an existing page, you not only have to change the name of that page, but also change all the links to that page. To execute these directives, you have to give it a `.shtml` extension.

- Use the `XBitHack` directive

```
XBitHack on
```

This directive specifies Apache server to parse files for SSI directives if they have the execute permissions assigned. Therefore, to add SSI directives to an existing page, file must be assigned execute permissions, using `chmod` command.

```
chmod +x page.html
```

where,

`x` - specifies the execute permission

`page.html` - specifies the name of the page which is granted the execute permission

For document to be cached, SSI pages must contain the last modified date or content-length HTTP headers. The content-length request is specified in 8-bit bytes. However, Apache does not send these values in its default configuration, which can result in slower client performance.

You can solve this issue in two different ways:

- Using the `XBitHack` Full configuration
- Using the directives provided by the `mod _ expires` module to set the expiration time

The following directives can be used to configure Apache for SSI:

- **AddType Directive**

  The `AddType` directive links a filename extension to the given content type. The syntax of the `AddType` directive is as follows:

```
AddType   MIME_type   file_extension
```

  where,

  `MIME type` - specifies the MIME type to be used for the files having the specified extension

  `file_extension` - specifies the filename extension

  For example, to configure Apache for SSI using the `AddType` directive, enter the code, as shown in Code Snippet 8, in the `httpd.conf` file.

  Code Snippet 8:

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

Note: The `file_extension` argument is case-insensitive and can be specified with or without a leading dot.

- **AddOutputFilter Directive**

The `AddOutputFilter` directive links the specified filename extension to the specified filter. A filter processes the data transmitted between the server and client. When a client transfers data to a server, the input filter processes this data. On the other hand, when the server sends data back to the client, the output filter processes it.

The syntax of the `AddOutputFilter` directive is as follows:

```
AddOutputFilter filter file_extension
```

where,

`filter` - defines the name of the filter

`file_extension` - defines a filename extension

Note: If you specify two or more filters, define them in the order in which they should process the content and separate them using semicolons. The `filter` and `file_extension` arguments are case-insensitive, and `file_extension` can be specified with or without a leading dot.

For example, to configure Apache server for SSI by using the `AddOutputFilter` directive, enter the code, as shown in Code Snippet 9 in the `httpd.conf` file.

Code Snippet 9:

```
AddOutputFilter INCLUDES .shtml
```

This configuration will process all `.shtml` files for SSI.

- **AddInputFilter Directive**

The `AddInputFilter` directive directs the filename extension to the filters that will respond to the client requests and `POST` input when Apache processes them.

The syntax of the `AddInputFilter` directive is as follows:

```
AddInputFilter    filter   file_extension
```

where,

`filter` - defines the name of the filter

`file_extension` - defines a filename extension

Concepts

Note: If you specify two or more filters, define them in the order in which they should process the content and separate them using semicolons.

The `file_extension` argument is case-insensitive and can be specified with or without a leading dot. Filenames may have multiple extensions and the `file_extension` argument will be compared against each of them.

12.5.2 SSI Commands

SSI commands generate dynamic information in HTML pages. When the server retrieves an SSI document, it executes the SSI commands, and performs the specified action. You must include SSI commands in HTML comment tags. If SSI is not enabled on the server, the browser ignores the commands but the commands will be visible in HTML source file.

The syntax for an SSI command is as follows:

```
<!-- #command  attribute=value  attribute=value … -->
```

Some of the SSI commands are as follows:

- **echo Command**

  The `echo` command adds contents of an HTTP environment variable or the include variables, such as `DATE_GMT`, `LAST_MODIFIED`, and `DOCUMENT_NAME` in place of the SSI directive. For example, to display the document name, enter the code, as shown in Code Snippet 10 in an HTML file.

  Code Snippet 10:

```
The name of this file is <!--#echo var="DOCUMENT_NAME" -->
```

  To display the name of the referrer site, enter the code, as shown in Code Snippet 11 in an HTML file.

  Code Snippet 11:

```
You were referred to this site from <!--#echo var="HTTP_REFERER" -->
```

  Table 12.2 lists some of the options for the `echo` command.

Option	Description
`CONTENT_LENGTH`	Specifies the total number of data bytes sent by the `POST` method to the server
`DOCUMENT_ROOT`	Specifies the root directory from where the site pages are served
`DATE_LOCAL`	Specifies the current date and time
`LAST_MODIFIED`	Specifies the date and time when the document was last modified

Option	Description
HTTP_USER_AGENT	Specifies the browser name

Table 12.2: echo Command Options

- **Include File**

The `Include File` command is used to include the contents of a file that is placed in the same directory as the file containing the `include` command, or a subdirectory of it. It will not allow an absolute path. For example, to include a file containing the contact address, enter the code, as shown in Code Snippet 12 in an HTML file:

Code Snippet 12:

```
<!--#include file="contactadrr.txt" -->
```

- **Include Virtual**

The `Include Virtual` command includes a file whose path is relative to the document root. Preferably, use the `Include Virtual` command instead of the `Include File` command.

For example, to include a file, `test.txt`, enter the code, as shown in Code Snippet 13, in an HTML file.

Code Snippet 13:

```
<!--#include virtual="/usr/local/apache2/includes/test.txt" -->
```

- **printenv**

The `printenv` command lists all the existing variables and their corresponding values. It includes both environment and user-defined variables. This command does not have any attributes. You can use this command for debugging in Apache 1.2 and higher versions. The syntax to use this command is as follows:

```
printenv
```

- **set Command**

The `set command` defines the value of a variable and has two attributes, `var` and `value`.

Table 12.3 lists the attributes of the set command.

Attributes	Description
var	Defines the name of the variable
value	Specifies the value to be set for the variable

Table 12.3: Attributes of the set Command

For example, to define a variable, category, with the value set to help, enter the code, as shown in Code Snippet 14 in an HTML file.

Code Snippet 14:

```
<!--#set var="category" value="help" -->
```

12.6 Directory Indexing

Directory indexing displays the list of included directories instead of the actual Web pages.

You can display a directory index in two different ways:

- Generate a file containing the list of all the directories by using the DirectoryIndex directive, controlled by the mod _ dir module. Typically, this file is called index.html.
- Configure Apache Web server to generate a list of all the directories using the mod _ autoindex module, only if the Indexes option is set.

The following section explains the directory indexing directives.

- **DirectoryIndex Directive**

  The DirectoryIndex directive defines a list of files that must be searched when a client requests a directory.

  The syntax of the DirectoryIndex directive is as follows:

```
DirectoryIndex  filename1 filename2 ...
```

  For example, to define the index.html file, enter the code, as shown in Code Snippet 15, in the httpd.conf file.

Code Snippet 15:

```
DirectoryIndex index.html
```

Now, if you send the request, `http://yourserver/dirname/`, then the server will redirect you to `http://myserver/dirname/index.html`.

Note: If `index.html` or `index.txt` is not specified, then Apache can execute CGI scripts when a client requests a directory. The Perl script is located at `/cgi-bin/index.pl`.

You need to configure Apache server to execute CGI scripts. To enable this script to process a request only when `index.html` and `index.txt` does not exists, enter the code, as shown in Code Snippet 16 in the `httpd.conf` file.

Code Snippet 16:

```
DirectoryIndex index.html index.txt /cgi-bin/index.pl
```

Apache scans the list in the specified order and returns the first document found.

Note: A trailing slash / must be specified at the end of the directory name.

- **Automatic Index Generation**

Apache Web server can be configured to automatically generate indexes. The `Options` directive can be used to configure Apache to generate indexes.

To configure Apache Web server to generate indexes, enter the code, as shown in Code Snippet 17 in the `httpd.conf` file.

Code Snippet 17:

```
Options +Indexes
```

The `Options` directive sets the `Indexes` option. If the user requests a URL that maps to a directory, and the `index.html` file is not defined, then the server returns the formatted list of directories, only if the `Indexes` option is set.

Concepts

Summary

- A dynamic Web page interacts with users and displays dynamic content.
- CGI is a specification that transmits information between a CGI program and the Web. A CGI program is a script that accepts and returns data, and conforms to the CGI specification.
- A CGI program is a script written in scripting languages, such as Perl, PHP, or C to generate dynamic content.
- The directives that enable CGI support in Apache Web server are `ScriptAlias`, `Options`, `SetHandler`, and `AddHandler`.
- SSI is used as an alternative to CGI programming for implementing dynamic content into HTML pages.
- The SSI documents have a `.shtml` extension.
- The directives used to configure Apache for SSI include `AddType`, `AddOutputFilter`, `AddInputFilter`, and `Options`.
- Directory indexing displays the list of included directories instead of the actual Web pages and you can perform directory indexing with the help of the `DirectoryIndex` directive.

Check Your Progress

1. Which of the following module implements the `AddType` directive?

   a. `mod_include`

   b. `mod_mime`

   c. `mod_dir`

   d. `mod_alias`

2. Which of the following directive specifies a handler for a directory?

   a. `SetHandler`

   b. `AddHandler`

   c. `AddType`

   d. `Options`

3. Which of the following module is a dynamic Web module?

   a. `mod_cgi`

   b. `mod_dir`

   c. `mod_perl`

   d. `mod_rewrite`

4. The SSI documents have a _____ extension.

   a. `.htm`

   b. `.shtml`

   c. `.shtm`

   d. `.stm`

Check Your Progress

5. Which of the following file contains directives to enable CGI in Apache server?

   a. `.htaccess` files

   b. `httpd.conf` file

   c. `http.conf` file

   d. `ssl.conf` file

Onlinevarsity

your e-way to learning

Session

13 Dynamic Pages (Lab)

Lab Guide

Objectives

At the end of this session, the student will be able to:

- *Explain the configuration of Apache to permit CGI.*
- *Create a simple HTML form.*
- *Write a CGI program.*
- *Execute a CGI program.*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Configuring Apache for CGI

You must edit the `httpd.conf` file to configure Apache for CGI. The `ScriptAlias` and `AddHandler` directives must be included in the `httpd.conf` file to enable Apache to execute CGI scripts.

To enable Apache to Permit CGI, perform the following steps:

1. **Logon to Linux as `root` user in the GNOME environment.**
2. **Open the Linux Terminal.**
3. **To open the `httpd.conf` file, enter the following at the command prompt:**

```
vi /usr/local/apache2/conf/httpd.conf
```

Figure 13.1 displays the `httpd.conf` file.

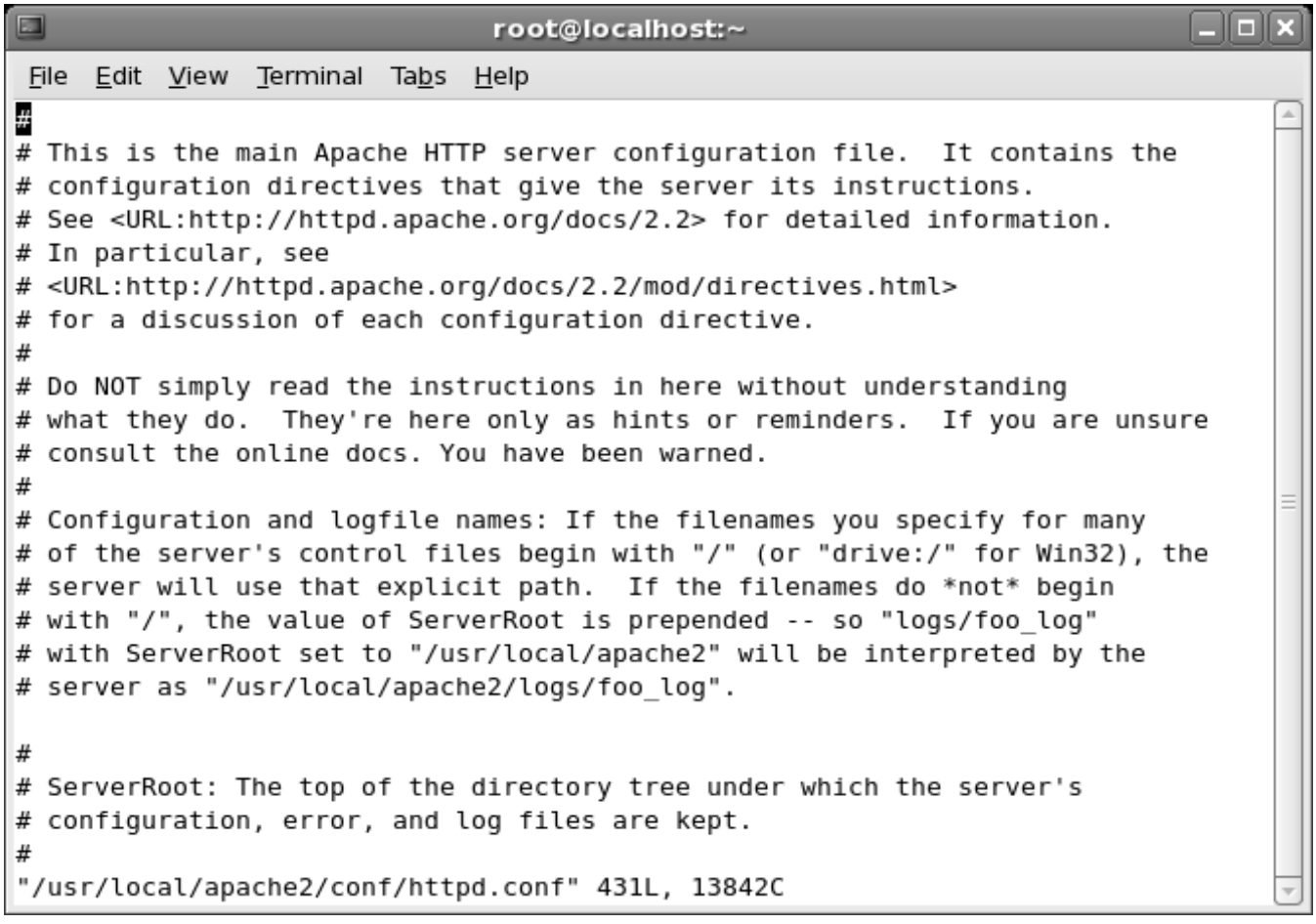


Figure 13.1: httpd.conf File

4. **Enter the following directive in the `<IfModule alias_module>` section of `httpd.conf` file:**

```
ScriptAlias /cgi-bin/ "/usr/local/apache2/cgi-bin"
```

Figure 13.2 displays the `ScriptAlias` directive in the `httpd.conf` file.

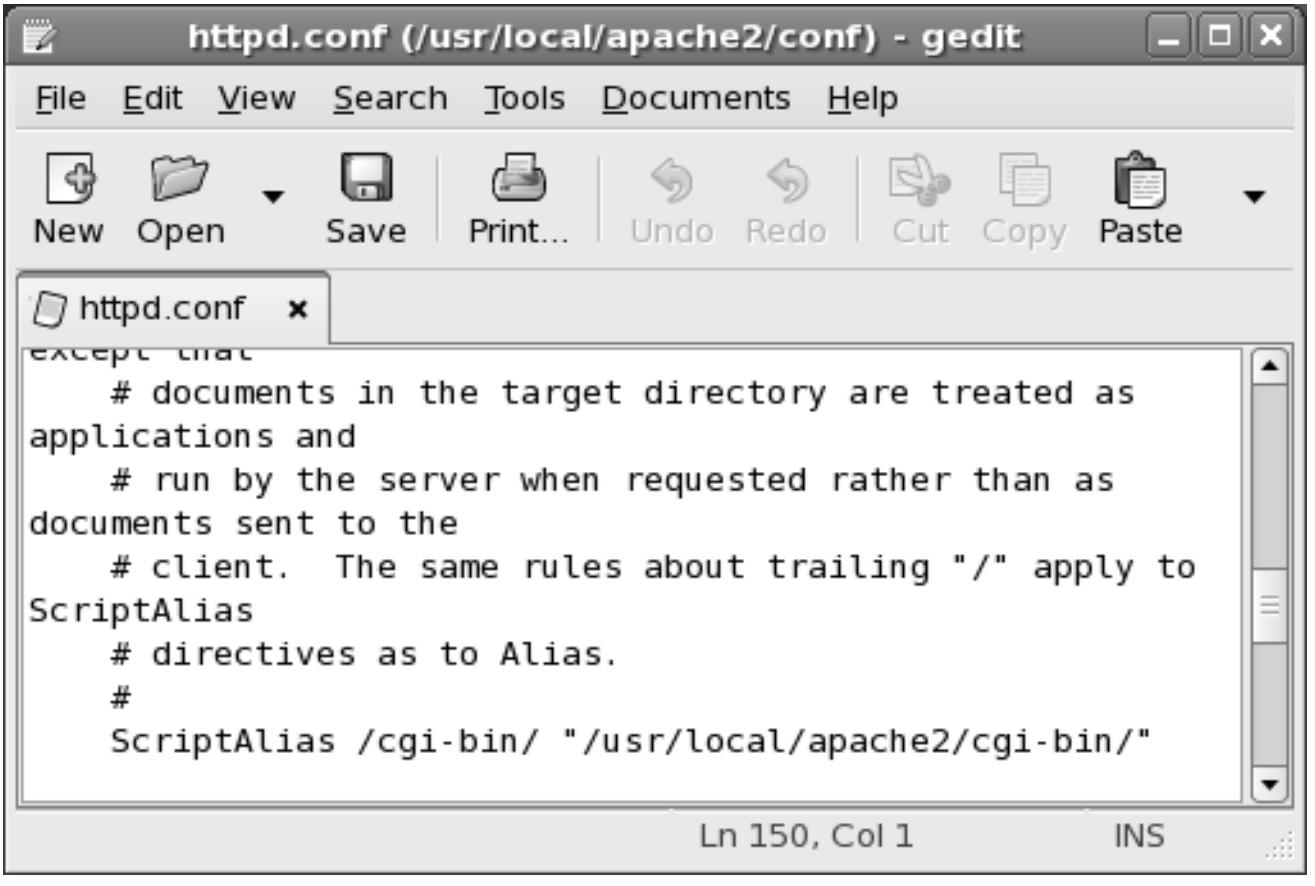


Figure 13.2: Configuring the ScriptAlias Directive

5. **Enter the following directive in the `<IfModule mime_module>` section of the `httpd.conf` file:**

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

Figure 13.3 displays the `IfModule` directive in the `httpd.conf` file.

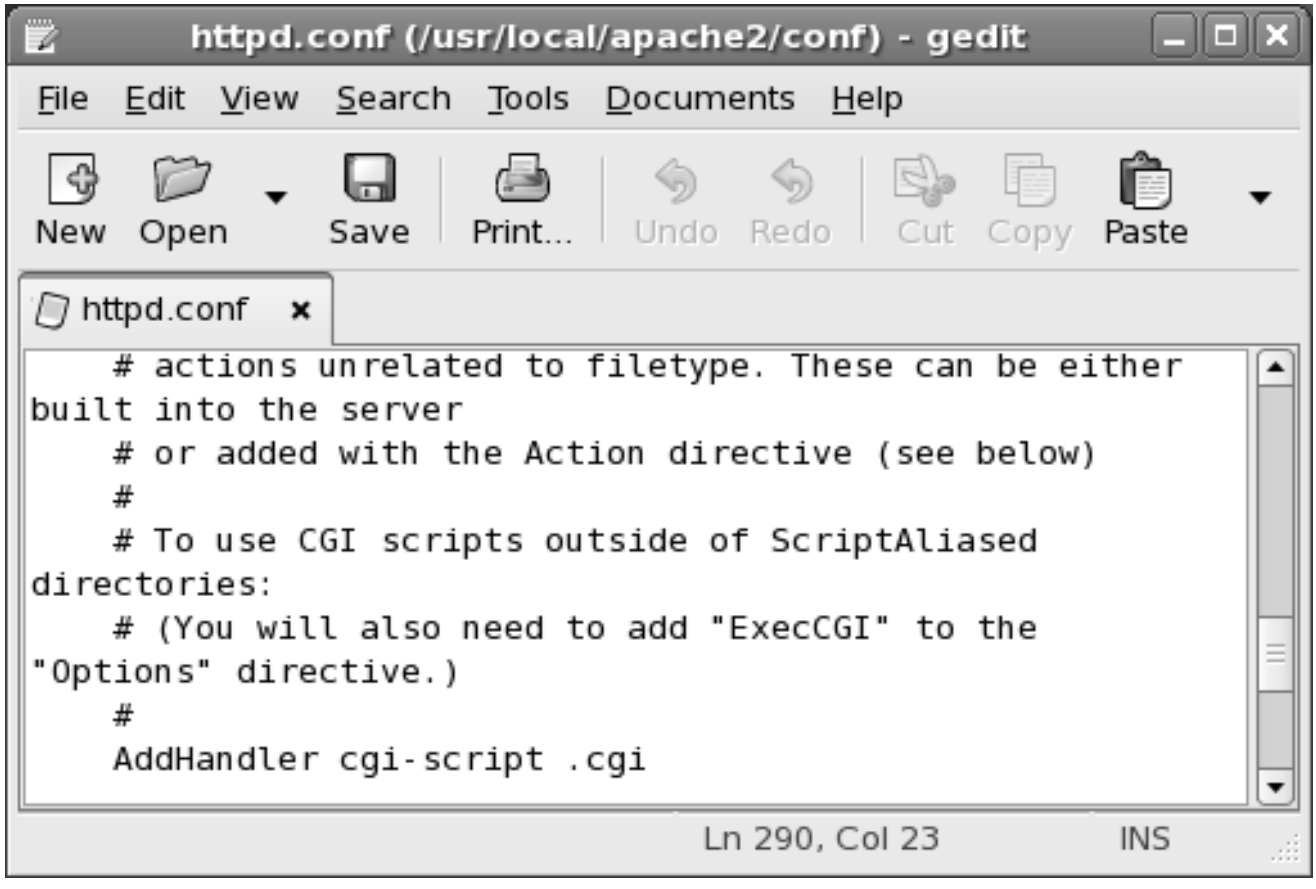


Figure 13.3: Configuring the IfModule Directive

6. **Press Esc and `:wq` to save and exit the `vi` editor.**

Creating an HTML Form

An HTML form is used to accept the data and submit it to the server. The form tag of HTML enables to design a user form. You will create a HTML form to accept the name, gender, and hobbies of the user. You will also create a CGI script that will retrieve the name, gender, and hobbies entered by the user and display it on the Web browser.

To create a HTML form for accepting data entered by the user, perform the following steps:

1. **To open a new file named `form.html` in the document `root`, enter the following at the command prompt:**

```
vi /usr/local/apache2/htdocs/form.html
```

2. **Enter the following HTML code in the file `form.html`:**

```
<html>
<head>
<title>Sample Form</title>
</head>
<body><h1>Enter Your Details....</h1>
<form action="/cgi-bin/perl_code.cgi" method=post>
Enter your name:
<input type="text" name="Name" size=30 maxlength=50><p>
Select your gender:
<input type="radio" name="Gender" value="Male" > Male
<input type="radio" name="Gender" value="Female" checked> Female
<p>
Select your hobbies:
<select name="Hobby">
<option selected> Computers
<option> Music
<option> Reading
<option> Sports
<option> Others
</select> <p>
<hr>
<input type="submit" value="Submit">
  or
<input type="reset" value="Reset"> <p>
</form>
</body>
</html>
```

3. **To save the file and exit the `vi` editor, press Esc and `:wq`.**

The `<form>` tag in HTML is used to design a form that consists of name, gender, and hobby fields. The `action` attribute of the form tag specifies the location of the cgi-script that accepts the form data as input. The method attribute denotes the method used for form submission to the server.

Writing a CGI Program

A CGI program is used to generate dynamic Web page. It accepts the data submitted in a user form and processes it. The CGI output is sent to the client browser that displays the html output.

A CGI program is also called as a cgi-script and can be written in any programming language such as C, Perl or PHP.

To create a CGI script for retrieving and displaying data entered by the user, perform the following steps:

1. **To open a new file named `perl_code.cgi` in the cgi-bin directory, enter the following command at the command prompt:**

```
vi /usr/local/apache2/cgi-bin/perl_code.cgi
```

2. **Type the following code in the file:**

```
#!/usr/bin/perl
# perl_form - a simple illustration of forms and Perl CGI

print "Content-type: text/html\n\n";

   &Process;

   print "<title>The Perl Output</title><h1>The Response</h1><hr>";
   print "The data entered in the form is:<ul>";

   local (*in) = @_ if @_;

   local ($i, $key, $val);
```

```
        if ( $ENV{'REQUEST_METHOD'} eq "GET" )
        {
          $in = $ENV{'QUERY_STRING'};
        }

        elsif ($ENV{'REQUEST_METHOD'} eq "POST")
        {
          read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
        }

         @in = split(/&/,$in);

        foreach $i (0 .. $#in)
        {
          # Convert all plus symbols to spaces
          $in[$i] =~ s/\+/ /g;

          # Split into key and value.
          ($key, $val) = split(/=/,$in[$i],2);

          # Convert %XX from hex numbers to alphanumeric
          $key =~ s/%(..)/pack("c",hex($1))/ge;
          $val =~ s/%(..)/pack("c",hex($1))/ge;

          # Associate key and value.
          $in{$key} .= "\0" if (defined($in{$key}));
          $in{$key} .= $val;
         }
           return length($in);
      }
```

3. **To save the code and exit the `vi` editor, press 'Esc' and `:wq`.**

4. **To assign execute permissions to the Perl script, enter the following command at the command prompt:**

```
chmod 755 /usr/local/apache2/cgi-bin/perl_code.cgi
```

The script displays the data submitted by the user. The first line in the script specifies the location of the Perl interpreter. It is mandatory to include this line in all Perl scripts. The symbol # is used to precede comments in a perl script. The `Process` function reads the URL-encoded data and decodes it to extract the form data. The split function splits the received string based on & symbol to generate a list of `name=value` pairs. Next, all the + symbols existing in a name or value are replaced into spaces. The `name=value` pairs are further split on a = and stored in separate variables. Finally, the form data is sent back to the client browser to be displayed to the user.

Executing the CGI Program

The user data entered in a form is submitted using the `Post` method to the server. The Perl script on the server accepts the data and generates the required result.

1. **To start Apache Web server, enter the following command at the command prompt:**

   ```
   /usr/local/apache2/bin/apachectl start
   ```

2. **Open the Mozilla Firefox Web browser.**

3. **Enter the following in the Address bar:**

   ```
   http://localhost/form.html
   ```

Figure 13.4 displays the output of the HTML file.

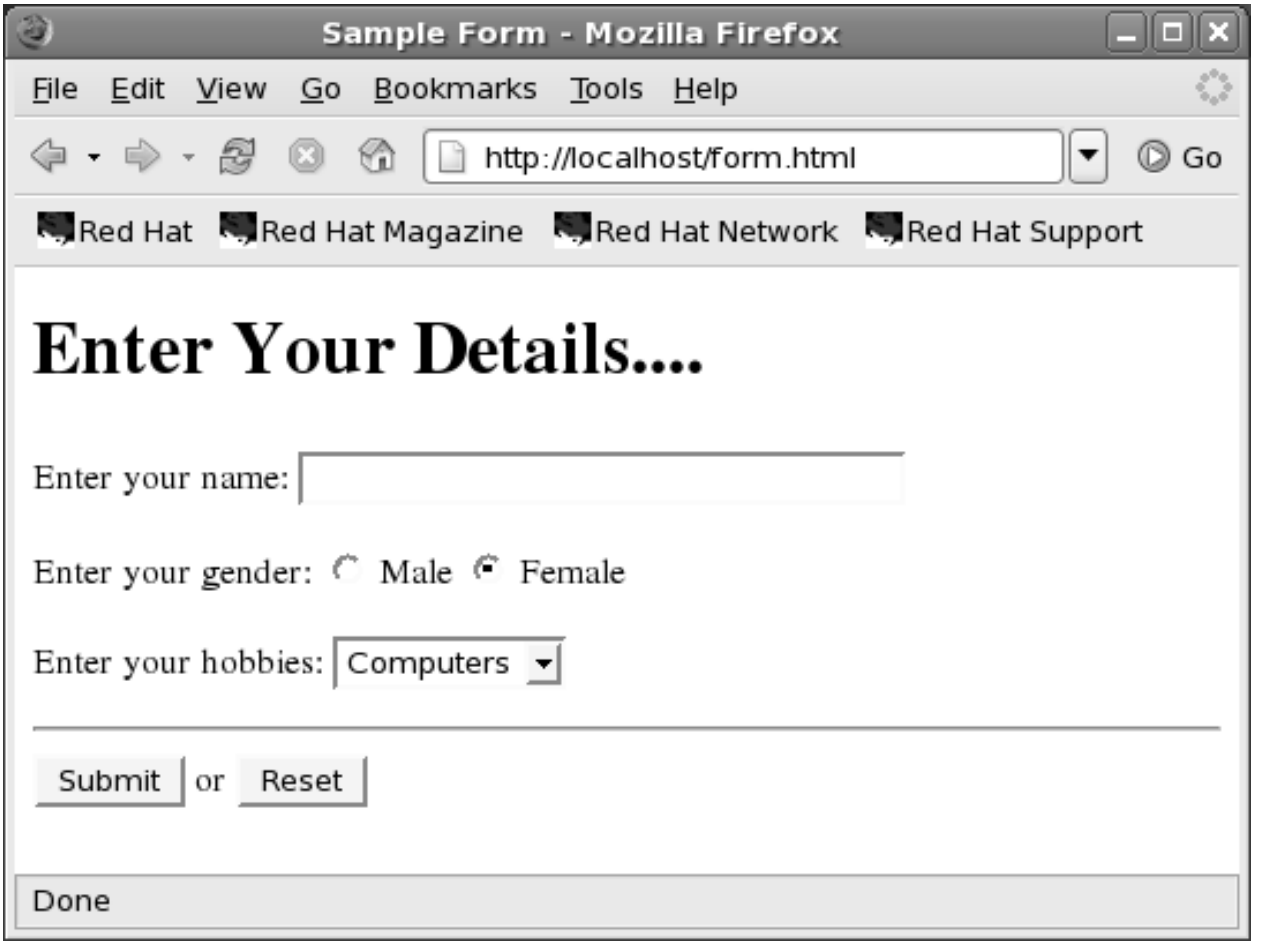


Figure 13.4: HTML Form

4. **Enter the following details in the Web page:**

```
Name:  Debbie Mark
Gender: Female
Hobby: Music
```

5. **Click Submit.**

Figure 13.5 displays the output of the Perl script.

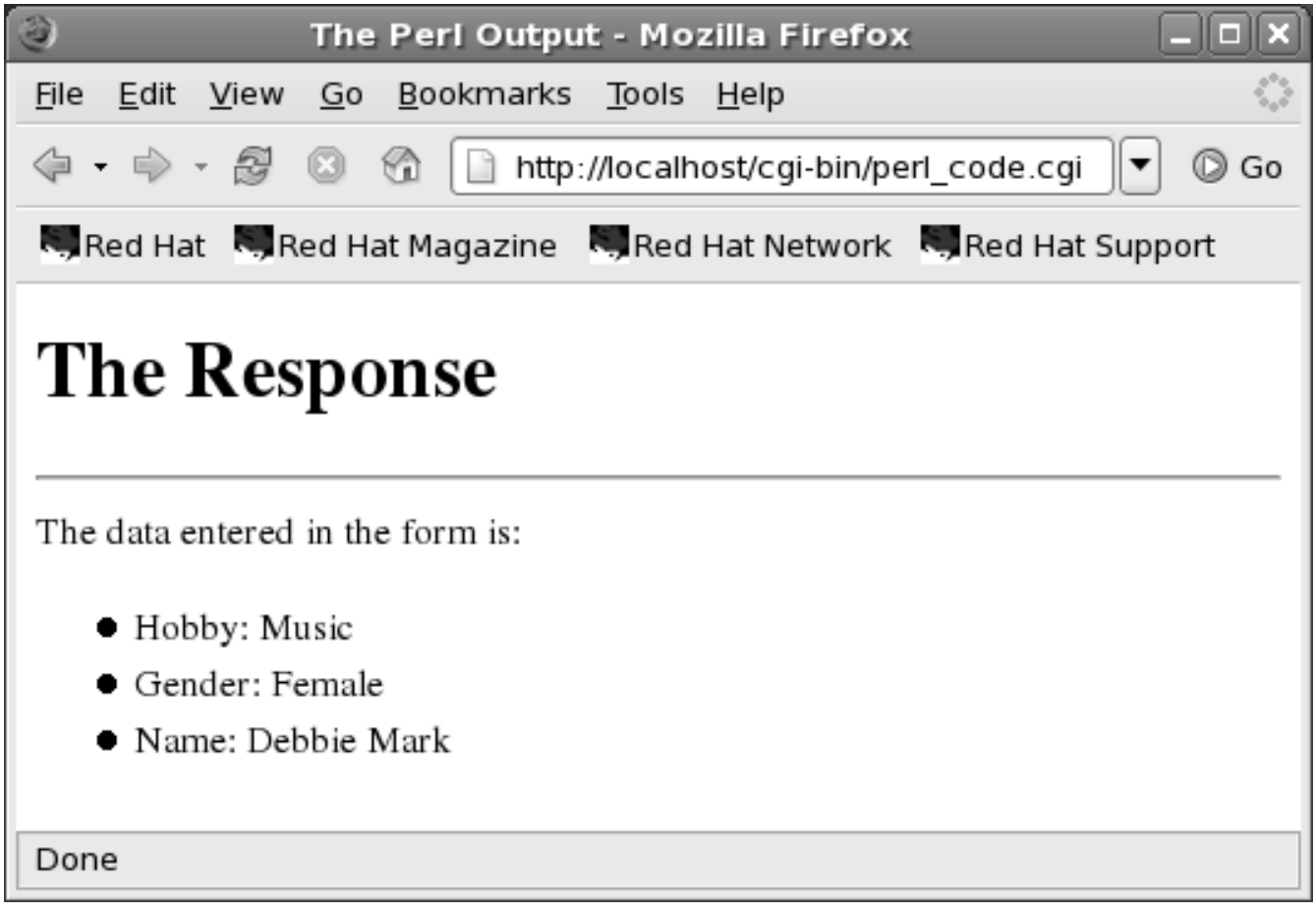


Figure 13.5: Output of the Perl Script

Do It Yourself

1. Add more input fields such as, date of birth and ethnicity to the above designed form and execute the script to view the output.

2. Configure Apache to execute cgi-scripts.

3. Configure Apache to execute CGI scripts from another directory instead of cgi-bin.

Onlinevarsity

your e-way to learning

BLOG

Balanced Learner-Oriented Guide

for enriched learning available

@

www.onlinevarsity.com

Session

14 Monitoring Apache Web Server

Concepts

Objectives

At the end of this session, the student will be able to:

- *Describe the configuration of Apache server to improve performance.*
- *Describe the configuration of caching on Apache server.*
- *Describe the configuration of Apache server to track user information.*

14.1 Introduction

The task of a Web administrator is to stabilize and retain the Web server's performance. The performance of Apache Web server will be affected if the directives are not configured optimally. Apache Web server provides certain directives that can be used for configuration of the server for optimum performance.

In this session, you will identify the configurations used to improve processes and threads in Apache Web server for optimal performance. You will also learn about proxy caching. In addition, you will learn about the directives that are used to track user information in Apache Web server.

14.2 Performance Tuning in Apache Web Server

Performance tuning means improving the overall functioning of the system. Apache Web server provides several directives that are directly related to its performance. Performance tuning of Apache Web server includes improving the working of the processes and threads. Memory and processors are the two critical elements that define the performance of a server.

14.2.1 Improving Processes in Apache Web Server

Apache Web server provides directives for process management. Apache directives are commands that defines the format in which the instructions in the configuration files are executed. You can use these instructions to configure Apache server. A process is a program that is currently running on the computer. The process management directives are dependent on Multi-Processing Modules (MPM). Apache contains a set of MPMs that bind to network ports on the machine, accept requests, and execute processes to handle the requests. There are some platform-specific MPMs, such as `beos`, `mpm_netware`, `mpmt_os2`, and `mpm_winnt`. The two most prominent MPMs in Apache server are `prefork` and `worker`.

The `prefork` MPM employs multiple child processes with a single thread. Each process handles one connection at a time. As compared to the `worker` MPM, `prefork` is faster but it utilizes more memory. The threadless design of `prefork` enables debugging on different platforms.

The `worker` MPM employs multiple child processes with multiple threads. Each thread handles one connection at a time. For high-traffic servers, using the `worker` MPM is preferred as it consumes less memory as compared to the `prefork` MPM.

Some of the directives that enable you to improve the processes in Apache Web server are as follows:

- **ServerLimit Directive** - The `ServerLimit` directive defines the maximum number of server processes that Apache Web server must create. It specifies the upper limit on the number of processes that can be configured. It is mandatory to restart Apache Web server for the changes in this directive to take effect. Table 14.1 lists the default value of the `ServerLimit` directive for each MPM.

MPM	Default Value
prefork	256
worker	16
perchild	8

Table 14.1: Default Value of MPMs in the ServerLimit Directive

The syntax for the `ServerLimit` directive is as follows:

```
ServerLimit number
```

For example, to set the number of processes that Apache must create to 8, enter the following code as shown in Code Snippet 1 in the `httpd.conf` file.

Code Snippet 1:

```
ServerLimit 8
```

You must exercise caution while using this directive. If the `ServerLimit` directive is set to a value higher than required, additional constraint is placed on the server as extra shared memory needs to be allocated.

Note: The maximum value that can be specified for the `ServerLimit:` directive is 20000. This value is compiled into the server at the time of configuration and installation.

- **StartServers Directive** - The `StartServers` directive defines the number of child processes that Apache Web server must create at startup. Depending on the server load, Apache server dynamically controls the number of processes. The configuration of this directive is optional.

Table 14.2 lists the default value of the `StartServers` directive for each MPM.

StartServers	Default Value
prefork	5
worker	3
perchild	2

Table 14.2: Default Values of the StartServers Directive

The syntax of the `StartServers` directive is as follows:

```
StartServers number
```

For example, to set the number of child processes that Apache Web server must create on startup to 3, enter the code as shown in Code Snippet 2 in the `httpd.conf` file.

Code Snippet 2:

```
StartServers 3
```

- **MinSpareServers Directive** - The `MinSpareServers` directive defines the minimum number of idle child server processes. A process is considered idle when it is not processing a request. The MPM `prefork` supports the `MinSpareServers` directive and has a default value of 5.

Note: If the number of Apache processes is less than the number of processes specified by the `MinSpareServers` directive, then the parent process creates new child processes with a maximum rate of 1 per second.

The syntax of `MinSpareServers` directive is as follows:

```
MinSpareServers number
```

For example, to set the minimum number of idle child server processes to 7, enter the code as shown in Code Snippet 3 in the `httpd.conf` file.

Code Snippet 3:

```
MinSpareServers 7
```

Note: The value for `MinSpareServers` directive must be set only on very busy sites and not to a very high value.

- **MaxSpareServers Directive** - The `MaxSpareServers` directive defines the maximum number of idle child server processes. A process is considered idle when it is not processing a request. If the numbers of idle processes exceed the limit specified in the `MaxSpareServer` directive, then the parent processes kill the excess child processes. MPM `prefork` supports this directive and has a default value of 10.

  The syntax of the `MaxSpareServers` directive is as follows:

```
MaxSpareServers number
```

  For example, to set the maximum number of idle child server processes to 15, enter the code as shown in Code Snippet 4 in the `httpd.conf` file.

  Code Snippet 4:

```
MaxSpareServers 15
```

  You should tune this parameter only on very busy sites. It is not advisable to set this parameter to a very large value. If the user or the administrator is trying to set the value lower than `MinSpareServers`, Apache server will automatically adjust it to `MinSpareServers + 1`.

- **MaxClients Directive** - The `MaxClients` directive defines the maximum number of requests or connections that will be served or processed simultaneously. The `prefork` and `worker` MPMs support this directive. The default value is 256. The value in the `ServerLimit` directive must be increased in order to modify the default value of the `MaxClients` directive. All connection requests exceeding the limit specified in the `MaxClients` directive will be processed chronologically. The requests will be processed up to the limit specified in the `ListenBacklog` directive.

  The syntax for the `MaxClients` directive is as follows:

```
MaxClients number
```

  For example, to set the maximum number of request to 40, enter the code as shown in Code Snippet 5 in the `httpd.conf` file.

  Code Snippet 5:

```
MaxClients 40
```

14.2.2 Improving Threads in Apache

A thread is a logical section of a program that executes independently of the other parts of the program.

Concepts

Multiple threads of a program executes concurrently. Apache Web server provides directives to monitor and optimize thread management.

Some of the directives that enable you to improve thread management in Apache Web server are as follows:

- **StartThreads Directive** - The `StartThreads` directive defines the number of threads to be created at startup. The perchild MPM supports the `StartThreads` directive and has a default value of 50.

  The syntax for the `StartThreads` directive is as follows:

  ```
  StartThreads number
  ```

  For example, to set the number of threads at startup to 8, enter the code as shown in Code Snippet 6 in the `httpd.conf` file.

  Code Snippet 6:

  ```
  StartThreads 8
  ```

Note: Apache dynamically modifies the value of the `StartThreads` directive depending upon the number of requests to be processed.

- **MinSpareThreads Directive** - The `MinSpareThreads` directive defines the minimum number of idle threads to process requests. Apache server creates new threads within a child process if there are no idle threads required to process a request. The `worker` and `mpm _ netware` MPMs supports the `MinSpareThreads` directive with default values of 75 and 10 respectively.

  The syntax for the `MinSpareThreads` directive is as follows:

  ```
  MinSpareThreads number
  ```

  For example, to define the minimum number of idle threads to handle the request to 10, enter the code as shown in Code Snippet 7 in the `httpd.conf` file.

  Code Snippet 7:

  ```
  MinSpareThreads 10
  ```

- **ThreadsPerChild Directive** - The `ThreadsPerChild` directive defines the number of threads that can be created by each child process. The threads are created by child processes during startup.

The `worker` MPM supports the `ThreadsPerChild` directive and has a default value of 25.

The syntax of the `ThreadsPerChild` directive is as follows:

```
ThreadsPerChild number
```

For example, to set the number of threads created by each child process to 70, enter the code as shown in Code Snippet 8 in the `httpd.conf` file.

Code Snippet 8:

```
ThreadsPerChild 70
```

- **ThreadLimit Directive** - The `ThreadLimit` directive defines the maximum number of threads per child process. The `worker` and perchild MPMs support this directive and has a default value of 64 for both the MPMs. If you modify the value of this directive at startup, Apache Web server ignores the modification, but you can alter the value of the `ThreadsPerChild` directive during a restart.

The syntax for the `ThreadLimit` directive is as follows:

```
ThreadLimit number
```

For example, to set the maximum number of threads per child process to 64, enter the code as shown in Code Snippet 9 in the `httpd.conf` file.

Code Snippet 9:

```
ThreadLimit 64
```

Note: Memory resources will be used if the value of the `Threadlimit` directive is set to a very high value.

14.2.3 Configuring Apache Web Server for Performance

Apache Web server's performance depends on the configuration of the directives. The directives in Apache Web server enables you to configure the server as per requirements. Some of these directives are time-consuming and impose a load on the server. You will learn about the directives, which, if possible, must be avoided to achieve a better performance.

The directives that affect Apache Web server's performance are as follows:

- **HostNameLookups Directive** - The `HostNameLookups` directive configures Apache Web server to log the hostname instead of the client's IP address. The core module supports this directive.

  The syntax for the `HostNameLookups` directive is as follows:

```
HostNameLookups On|Off|Double
```

  where,

  `On` - activates the `HostNameLookups` directive

  `Off` - disables the `HostNameLookups` directive

  `Double` - enables double-reverse DNS lookup

  Double-reverse DNS lookup works as follows: A forward lookup is performed on the result obtained after a reverse lookup. The original address must correspond to at least one of the IP addresses in the forward lookup.

  For example, to activate the `HostNameLookups` directive, enter the code as shown in Code Snippet 10 in the `httpd.conf` file.

  Code Snippet 10:

```
HostNameLookups On
```

  Enabling this directive affects Apache Web server's performance as it increases the time required to complete a request. You must use this directive only if required.

  You can use the `logresolve` utility, compiled to the `bin` directory, to resolve the IP address of the client in the log files. The `logresolve` utility resolves the IP addresses in Apache's access log files using its internal hash-table cache, thus minimizing the impact on your web server.

- **AllowOverride Directive** - The `AllowOverride` directive enables you to include directives allowed in `.htaccess` files. The core module supports the `AllowOverride` directive. If you enable the `AllowOverride` directive, Apache Web server searches for the `.htaccess` file in each directory starting at the root directory. This process is time-consuming as Apache performs this check for each request. When the server finds an `.htaccess` file, it must know which directives declared in that file can override the earlier configuration directives.

Note: The `AllowOverride` directive is valid only in the `<Directory>` sections declared without regular expressions.

The syntax for the `AllowOverride` directive is as follows:

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

When the `AllowOverride` directive is set to None, then Apache ignores the `.htaccess` files. However, when the `AllowOverride` directive is set to All, then any directive having the `.htaccess` context is permitted in the `.htaccess` files.

Table 14.3 lists the directive-type and the corresponding directives that they permit.

Directive-Type	Directives permitted
`AuthConfig`	Authorization directives
`FileInfo`	Document type directives
`Indexes`	Directory indexing directives
`Limit`	Host access control directives
`Options`	Directory specific directives

Table 14.3: Directive-type and the Directives permitted

- **Options Directive** - The `Options` directive, supported by the core module, controls the server features that are available in a specific directory.

Table 14.4 lists the different options and their corresponding functions:

Options	Functions
`ExecCGI`	Enables the execution of CGI scripts
`FollowSymLinks`	Configures the Apache Web server to follow symbolic links. A symbolic link is a special file in Linux that contains the pathname to another file.
`Includes`	Enables Server-side includes
`IncludesNOEXEC`	Enables Server-side includes but disables the #exec cmd and #exec cgi command
`Indexes`	Returns a formatted listing of the directory when a directory does not contain a DirectoryIndex directive in that directory. The DirectoryIndex directive defines a list of files that must be searched when a client requests a directory.

Options	Functions
MultiViews	Enables content-negotiated multiviews. Content negotiation enables Apache server to select the representation of a resource based on the media type, languages, character set, and encoding information provided by the browser.
SymLinksIfOwnerMatch	Enables Apache server to follow symbolic links only for the target file where the user ID of the file is the same as the link
None	Disables all additional features
All	Enables all the options except for MultiViews

Table 14.4: Different Options and their Functions

14.3 Caching

A proxy server is set up to cache the files received from the remote host. The `mod_proxy` module performs proxying and caching functions. Caching implies storing the commonly used data at a location from where it can be accessed quickly. Proxying allows indirect connection to a computer or a network service through a proxy server. In Apache 2.2, the caching operation is included in the `mod_cache` module. The `mod_cache` module is split into two sub modules, `mod_disk_cache` and `mod_mem_cache`.

The module `mod_disk_cache` handles the file-based cache. The `mod_mem_cache` module implements a memory based storage manager. The `mod_mem_cache` module can be configured to operate in two modes. They are caching open file descriptors and caching objects in heap storage. `mod_mem_cache` can be used to cache locally generated content or to cache backend server content for `mod_proxy`.

The following directives are supported by the `mod_disk_cache` module:

- **CacheRoot Directive** - The `CacheRoot` directive defines the location for storing the cached files. The value for `CacheRoot` directive must be specified if the `mod_disk_cache` module has been configured. Apache server will return a configuration file processing error if there is no value specified for the `CacheRoot` directive.

  The syntax for the `CacheRoot` directive is as follows:

  ```
  CacheRoot directory-path
  ```

  where,

  `directory-path` - specifies the location to store the cached files

For example, to define the location of the cached files to the `/usr/bin/log` directory, enter the code as shown in Code Snippet 11 in the `httpd.conf` file.

Code Snippet 11:

```
CacheRoot /usr/bin/log
```

- **CacheDirLevels Directive** - The `CacheDirLevels` directive defines the number of levels of subdirectories in the cache directory. The default value of the `CacheDirLevels` directive is set to 3, which implies that the cached data will be saved three levels below the `CacheRoot` directory.

The syntax for the `CacheDirLevels` directive is as follows:

```
CacheDirLevels levels
```

For example, to set the subdirectory level to 7, enter the code as shown in Code Snippet 12 in the `httpd.conf` file.

Code Snippet 12:

```
CacheDirLevels 7
```

- **CacheDirLength Directive** - The `CacheDirLength` directive defines the number of characters allowed in the subdirectory names present in the cache directory. The default value for this directive is 2.

The syntax for the `CacheDirLength` directive is as follows:

```
CacheDirLength length
```

For example, to set the number of characters in the subdirectory names to 3, enter the code as shown in Code Snippet 13 in the `httpd.conf` file.

Code Snippet 13:

```
CacheDirLength 3
```

Note: Ensure that the value of `CacheDirLevels*CacheDirLength` should not be greater than 20.

- **CacheMaxFileSize Directive** - The `CacheMaxFileSize` directive defines the maximum size for a file that can be stored in the cache. The file size must be specified in bytes.

The syntax of the `CacheMaxFileSize` directive is as follows:

```
CacheMaxFileSize bytes
```

For example, to set the maximum size of a file stored in the cache to 70000, enter the code as shown in Code Snippet 14 in the `httpd.conf` file.

Code Snippet 14:

```
CacheMaxFileSize 70000
```

Note: The default size of the `CacheMaxFileSize` directive is `1000000` bytes.

- **CacheMinFileSize Directive** - The `CacheMinFileSize` directive defines the minimum size for a file that can be stored in the cache. You must define the file size in bytes.

  The syntax of the `CacheMinFileSize` directive is as follows:

```
CacheMinFileSize bytes
```

  For example, to set the minimum size of a file stored in the cache to 50, enter the code as shown in Code Snippet 15 in the `httpd.conf` file.

  Code Snippet 15:

```
CacheMinFileSize 50
```

14.4 User Management and Tracking

Web developers track and maintain information about the visitors to Web sites. User information, such as the time spent on each page, links followed, or the frequency of visits is important for a commercial Web site.

The HTTP Referer header contains the URL of the page that contains the link to the current page. This provides information on how users navigate the Web site for URLs.

14.4.1 Tracking Cookies

A cookie is a text file that contains user-specific information. A Web server creates a cookie on the client machine and uses it to identify and track user information. Since HTTP is stateless, cookies store state information about the client-server transaction.

Cookies can be used to:

- Monitor Web site traffic on Apache Web server
- Analyze Web site traffic by identifying which features are used more frequently on the Web pages
- Strengthen server security by identifying the users and tracking their activity

If the information recorded in the access logs is not adequate to analyze the user behavior, you must use the `mod_usertrack` module of Apache.

The directives in the Apache Web server that enables you to track cookies are as follows:

- **CookieTracking Directive** - The `CookieTracking` directive enables you to track cookies.

  The syntax for the `CookieTracking` directive is as follows:

```
CookieTracking on|off
```

  For example, to enable `CookieTracking`, enter the code as shown in Code Snippet 16 in the `httpd.conf` file.

  Code Snippet 16:

```
CookieTracking on
```

  Apache server creates a user-tracking cookie for all the new requests received when the `CookieTracking` directive is set `on`. This directive can be configured to `on` or `off` for a required directory.

  To enable user to track cookies for all directories except for the `public_html` directory, enter the code as shown in Code Snippet 17 in the `httpd.conf` file.

  Code Snippet 17:

```
CookieTracking on
<Directory */public_html/>
CookieTracking off
</Directory>
```

- **CookieExpires Directive** - The `CookieExpires` directive specifies an expiration time for the cookie created by the `usertrack` module. The cookies will be valid only for the current browser session if the `CookieExpires` directive is not configured.

  The syntax for the `CookieExpires` directive is as follows:

```
CookieExpires expiry-period
```

where,

`expiry-period` - defines validity period for a cookie. The value specified can be number of seconds or a format that includes valid denominations, such as years, months, weeks, days, hours, and minutes.

Note: If the expiry time is in any format other than one number indicating the number of seconds, it must be enclosed within double quotes.

Some of examples of the `CookieExpires` directive are as follows:

- To define the cookie expiry period after two hours, enter the code as shown in Code Snippet 18 in the `httpd.conf` file.

Code Snippet 18:

```
CookieExpires 7200
```

Note: The cookie will not expire if the user revisits the site within two hours.

- To define cookie expiry duration for a longer period, enter the following code in the `httpd.conf` file.

```
CookieExpires "1 year 3 months 2 weeks 5 days"
```

- **CookieName Directive** - The `CookieName` directive enables you to define a name for the cookie that is used for tracking. The default cookie name is Apache.

The syntax for the `CookieName` directive is as follows:

```
CookieName token
```

where,

`token` - defines a valid name for the cookie. The name can include any characters such as A-Z, a-z, 0-9, "_", and "-".

For example, to specify the cookie name as Troubleshoot, enter the following code in the `httpd.conf` file.

```
CookieName Troubleshoot
```

- **CookieDomain Directive** - The `CookieDomain` directive defines the domain for which the

cookie is created or will be tracked. If this directive is not present, no domain is included in the cookie header field.

The syntax of the `CookieDomain` directive is as follows:

```
CookieDomain Domain
```

where,

`Domain` - specifies the domain name

Note: The domain name must begin with a dot and include a dot in the name.

For example, to, specify the domain to which the cookie applies, enter the code as shown in Code Snippet 19 in the `httpd.conf` file.

Code Snippet 19:

```
CookieDomain .apachesite.com
```

- **CookieStyle Directive** - The `CookieStyle` directive enables you to specify the format of the cookie header field.

The syntax for the `CookieStyle` directive is as follows:

```
CookieStyle Netscape|Cookie|Cookie2|RFC2109|RFC2965
```

where,

`Netscape`, `Cookie`, `Cookie2`, `RFC2109`, `RFC2965` are cookie standards

The three formats allowed are as follows:

- **Netscape** - original and default format
- **Cookie** or **RFC2109** - developed after the `Netscape` syntax
- **Cookie2** or **RFC2965** - current cookie syntax

14.4.2 Storing User Tracking Information in a Log File

You must use the `LogFormat` directive to create a cookie log. Apache contains internal notes that are used to pass information amongst modules. The transfer of information between a Web server and a client is implemented by using cookies.

The following examples illustrate the process of recording user tracking information in a log file:

- To add a user tracking cookie to the standard `Logformat` directive, enter the code as shown in Code Snippet 20 in the `httpd.conf` file.

Code Snippet 20:

```
LogFormat "%h %l %u %t \"%r\" %>s %b %{cookie}n"
```

Table 14.5 lists the options available for the `LogFormat` directive.

Option	Description
`%h`	Specifies the remote host
`%l`	Specifies the remote log name
`%u`	Specifies the remote username
`%t`	Specifies the time of receiving the request
`%r`	Specifies the first line of the request
`%>s`	Specifies the status returned to the client
`%b`	Specifies the size of the response. The size is specified in bytes

Table 14.5: Options for the LogFormat Directive

This is the default syntax for the `LogFormat` directive that specifies the format of the access log file.

An alternative form of the `LogFormat` directive allows you to link an explicit format string with a nickname. After this, you can use the nickname instead of the whole format string in the succeeding `LogFormat` or `CustomLog` directives.

- To create a separate cookie log file using the `CustomLog` directive, enter the code as shown in Code Snippet 21 in the `httpd.conf` file.

Code Snippet 21:

```
CustomLog mylogs/clickstream "%{cookie}n %r %t"
```

In the example, the cookie log contains the cookie, the request, and the date and time of the request.

Summary

- A process is a program that is currently running on the computer.
- The process management directives are MPM dependent, with more than one MPM implementing each directive.
- The directives that perform process management in Apache Web server are namely, `ServerLimit`, `StartServers`, `MinSpareServers`, and `MaxSpareServers`.
- The directives that perform thread management in Apache Web server are namely, `StartThreads`, `MinSpareThreads`, `MaxSpareThreads`, and `ThreadsPerChild`.
- The directives that directly affect Apache Web server's performance are namely, `HostNameLookups` and `AllowOverride`.
- A proxy server can be configured for caching documents received from the remote host.
- The `mod _ cache` module of Apache Web server is used to implement proxy caching.
- The directives that can be used to configure cache in the proxy server are `CacheRoot`, `CacheDirLevels`, and `CacheDirLength`.
- Cookies contain user-specific information in a text file.
- The `mod _ usertrack` module of Apache Web server is used to implement the cookie feature.
- The directives used to configure cookies are namely, `CookieTracking`, `CookieExpires`, `CookieName`, and `CookieDomain`.

Check Your Progress

1. Which of the following directive defines the minimum number of idle threads to process requests?

   a. `MinSpareThreads`

   b. `NumServers`

   c. `MinSpareServers`

   d. `MaxSpareServers`

2. Which of the following directive defines the maximum number of threads per child process?

   a. `ThreadLimit`

   b. `ThreadsPerChild`

   c. `MaxSpareThreads`

   d. `MinSpareThreads`

3. Which of the following module of Apache Web server enables the `CacheRoot` directive?

   a. `mod_status`

   b. `mod_disk_cache`

   c. `mod_log_config`

   d. `mod_access`

4. Which of the following directive monitors and reduces the cache size?

   a. `CacheSize`

   b. `CacheRoot`

   c. `CacheGcInterval`

   d. `CacheMaxFileSize`

Check Your Progress

5. Which feature of Apache Web server tracks user information?

   a. `mod_status`

   b. `log`

   c. `Cookie`

   d. `LogFormat`

Session

15 Monitoring Apache Web Server (Lab)

Lab Guide

Objectives

At the end of this session, the student will be able to:

- *Explain the configuration of Apache Web server to improve process execution.*
- *Explain the configuration of Apache Web server to improve thread execution.*
- *Explain the configuration of Apache Web server to track user activity.*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Improving Processes in Apache

Apache provides directives for process management. A process is an executable program. Directives in Apache server are dependent on the Multi-Processing Modules (MPMs). The prefork module enables you to implement process management directives in Apache Web server. In the prefork module, child processes are created by one control process. These child processes accept and process client requests. Apache creates a predefined number of processes to process requests.

1. **Logon to Linux as `root` in GNOME environment.**
2. **Open the Linux Terminal.**
3. **To browse to the conf directory, enter the following command at the command prompt:**

```
cd /usr/local/apache2/conf
```

4. **To open the `httpd.conf` file, enter the following command at the command prompt:**

```
vi httpd.conf
```

5. **To improve process execution, enter the following code in the `<IfModule>` section of the `httpd.conf` file:**

```
<IfModule prefork.c>
```

```
    StartServers          6
    MinSpareServers       6
    MaxSpareServers      12
    MaxClients          155
    </IfModule>
```

Figure 15.1 displays the `<IfModule>` in the `httpd.conf` file.]

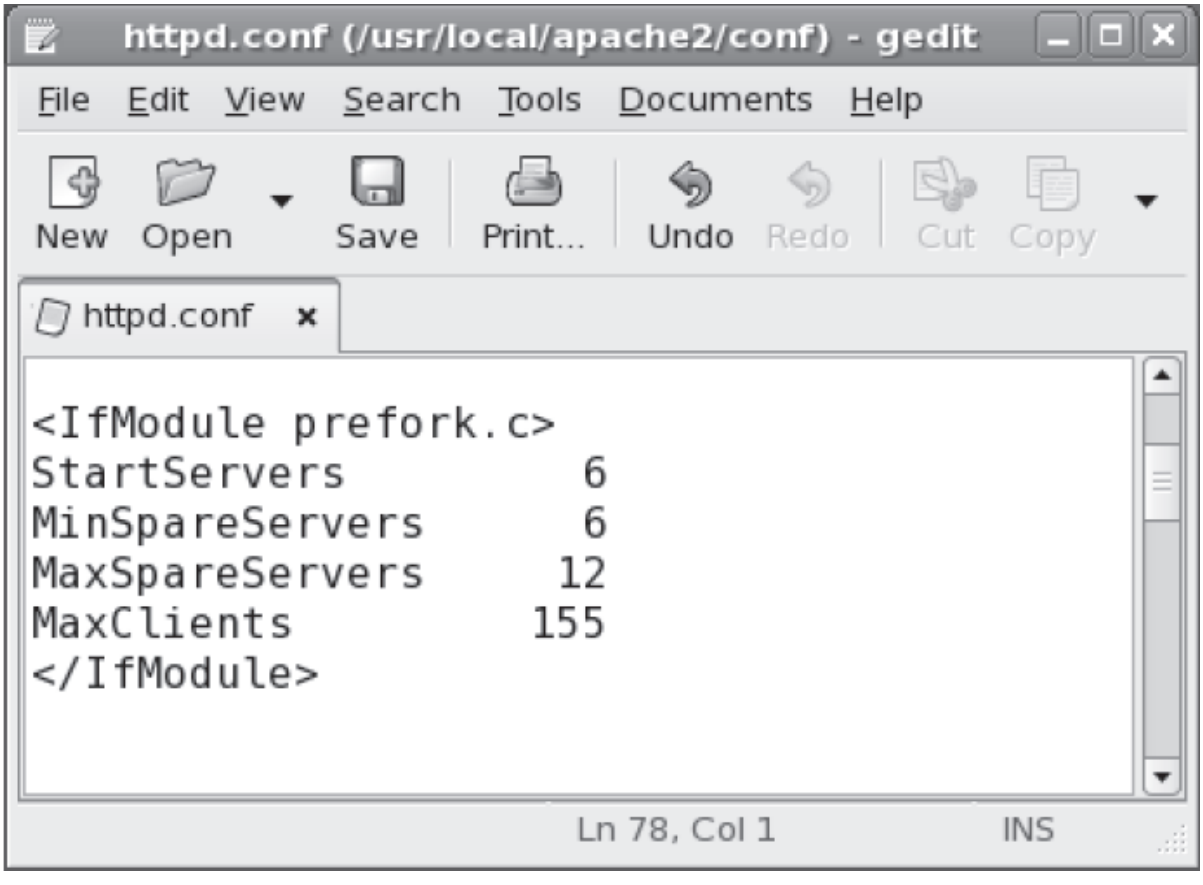


Figure 15.1: <IfModule> Directive

6. **Press the 'Esc' key and `:wq` to save changes and exit the `vi` editor.**

In the example, the `StartServers` directive defines the number of child processes to be created at startup. The minimum and maximum number of idle processes is defined in the `MinSpareServers` and `MaxSpareServers` directives. The maximum number of requests that Apache will process simultaneously is specified in the `MaxClients` directive.

Lab Guide

Improving Threads in Apache

Apache provides directives that enable thread management. The `prefork` MPM implements a non-threaded pre-forking Web server. The `worker` MPM implements a hybrid multi-process multi-threaded server. The configuration file can be modified to include the `prefork` and the `worker` directives that enable thread management.

To include worker thread management directives, perform the following steps:

1. **Open the `httpd.conf` file.**

2. **Enter the following code:**

```
<IfModule worker.c>
StartServers          4
MaxClients          152
MinSpareThreads      22
MaxSpareThreads      78
ThreadsPerChild      25
</IfModule>
```

Figure 15.2 displays the `<IfModule>` in the `httpd.conf` file.

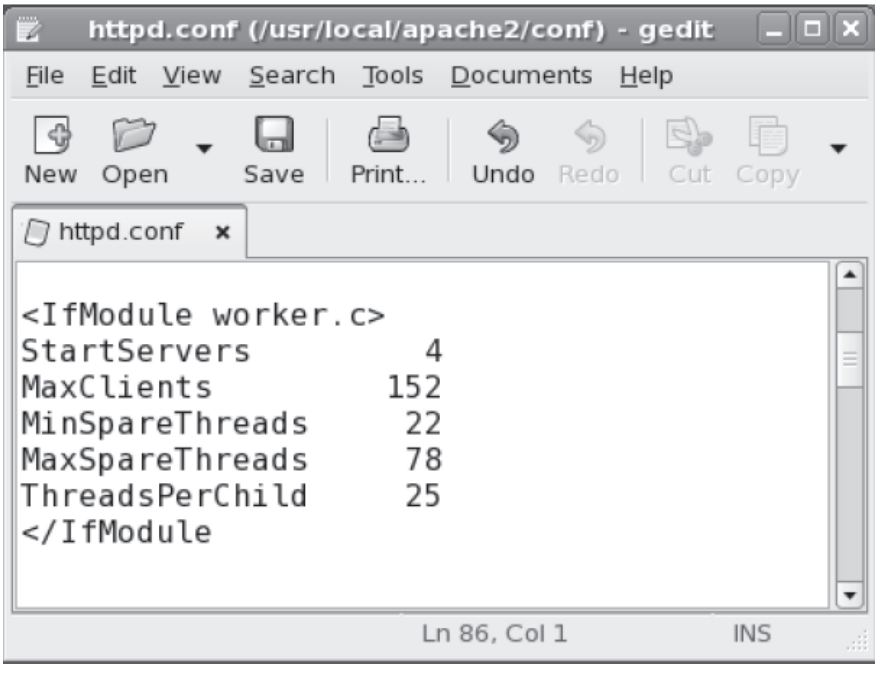


Figure 15.2: Including Worker Thread Management Directives

3. **Press the 'Esc' key and `:wq` to save changes and exit the `vi` editor.**

In the configuration, the `StartServers` directive defines the number of child processes to be created at startup. The maximum number of requests that Apache server will process simultaneously is specified in the `MaxClients` directive. The number of child processes that will be created by each child process is specified using the `ThreadsPerChild` directive. The maximum and minimum number of spare threads is defined using the `MinSpareThreads` and `MaxSpareThreads` directives.

Note: By default, the `httpd-mpm.conf` file contains information about the MPM modules, such as, prefork and worker. The `httpd-default`.conf file contains information about the `HostName` Lookups directive and `httpd-info`.conf file contains information about the `ExtendedStatus` directive. These files can be included in the `httpd.conf` file by uncommenting the directives in the `httpd.conf` file.

Editing Other Directives

Apache server provides certain directives that impose extra load on the server operation. These directives must be configured to impose minimum load on the server operation and avoid a performance loss.

To configure Apache for minimum load, perform the following steps:

1. **Open the `httpd.conf` file in the vi editor.**

2. **To disable `HostnameLookups` directive, enter the following code:**

```
HostnameLookups off
```

3. **To disable the `ExtendedStatus` directive, enter the following code:**

```
ExtendedStatus off
```

4. **Press the 'Esc' key and `:wq` to save changes and exit the `vi` editor.**

Tracking User Information

Cookies enable you to track user information. The `mod_usertrack` module of Apache enables the use of the cookie directives in Apache server.

Session 15

Monitoring Apache Web Server (Lab)

To configure Apache server for tracking user information for a domain, perform the following steps:

1. **To enable cookie tracking in Apache, enter the following directive in the `httpd.conf` file.**

   ```
   CookieTracking on
   ```

2. **To configure the cookie duration, enter the following directive in the `httpd.conf` file.**

   ```
   CookieExpires '4 months'
   ```

3. **To set the cookie name, enter the following directive in the `httpd.conf` file.**

   ```
   CookieName Customer
   ```

4. **To specify the `CookieDomain`, enter the following directive in the `httpd.conf` file.**

   ```
   CookieDomain .mysite.com
   ```

5. **Press the 'Esc' key and `:wq` to save changes and exit the `vi` editor.**

Do It Yourself

1. Configure Apache server to track cookies with the default cookie syntax.
2. Configure Apache server to store cookies for a duration of two days.
3. Configure Apache server for the prefork module to create a maximum 160 child processes at startup.
4. Configure Apache server to accept 120 requests on a persistent connection.
5. Set the maximum number of idle threads to 20.

Session

16 Virtual Web Hosting

Concepts

Objectives

At the end of this session, the student will be able to:

- *Define virtual web hosting.*
- *Identify the types of virtual hosts.*
- *Describe the process of configuring an IP-based virtual host.*
- *Describe the process of configuring a name-based virtual host.*
- *Describe the process of configuring Apache Web server for mixed virtual hosting.*

16.1 Introduction

Virtual Web hosting is a technique in which Web sites with different names run on the same server. It allows hosting of multiple domain names on a computer using a single IP address which provides efficient handling of resources. In other words, it allows a machine to share its resources. You can create virtual hosts based on their names and IP addresses.

In this session, you will learn about the types of virtual hosts supported by Apache Web server. You will learn the directives used to set up virtual hosts in Apache. In addition, you will learn to configure Apache server for each type of virtual host.

16.2 Define Virtual Web Hosting

Virtual Web hosting means running more than one Web site on a single machine. You can differentiate these Web sites by using hostname aliases. On the other hand, if a network is multi-homed, that is, it has its own public IP address range and an Autonomous System number then you can differentiate the Web sites by distinct IP addresses.

Virtual Web hosting provides clients with domain name registration, file storage, and directory services for the files from where the Web page is built. Virtual Web hosting runs multiple 'virtual' Web servers on a single physical host computer, thus allowing a single computer to host different independent Web sites.

A browser that supports HTTP/1.1 cannot distinguish between multiple virtual hosts on a single Apache server and multiple sites running on multiple servers.

Virtual Web hosting is one of the most popular and cost-effective hosting options available.

Virtual Web Hosting

It is also known as shared hosting. It is an optimal solution for small- to medium-sized (and even some larger) Web sites that has reasonable bandwidth requirements.

When you start Apache server, it binds to some ports and addresses on the local machine and waits for incoming requests. The virtual host feature of Apache server enables it to respond to different IP address, hostnames, and ports. A virtual host has the capacity to run different servers for different IP addresses, hostnames, and ports on the same server.

You can configure a virtual host by using the `httpd.conf` file. The two types of virtual hosts are as follows:

- **IP-based virtual hosts** - uses different IP addresses for each Web site. You must assign multiple IP addresses to the server machine.

- **Name-based virtual hosts** - uses names to determine a Web site. However, name-based virtual hosts share a single IP address.

A virtual host consolidates all sites on a single or a small group of machines. Using virtual host, a single server answers request for multiple IP addresses. However, the Web browser interprets each virtual host as a different Web site. This configuration enables a Web server to share resources, such as, memory and processor cycles.

The core module provides the directives for the virtual host. Some of the basic directives used to create a virtual host is as follows:

- **<VirtualHost> Directive** - The `<VirtualHost>` directive contain instructions that are applicable to specific IP addresses or hostnames. Apache uses this directive when it receives a request for a document on a particular virtual host set up on the server.

  The syntax for the `<VirtualHost>` directive is as follows:

```
<VirtualHost addr[:port] [addr[:port]] ...> ... </VirtualHost>
```

  where,

  `addr` - specifies any of the following:

  - IP address of the virtual host
  - Domain name for the IP address
  - The `*` character to include a range of IP addresses. (This character is applicable only to the `NameVirtualHost` directive.)

- The `_default_` string incorporates the IP addresses that are not clearly specified in any other virtual host. (This string is employed only with IP virtual hosting.)

`port` - specifies the port number on which Apache server accepts incoming requests

Consider the code as shown in Code Snippet 1, in the `httpd.conf` file.

Code Snippet 1:

```
<VirtualHost 192.154.127.21:80>
```

The code snippet binds the virtual host to the port 80.

- **NameVirtualHost Directive** - The `NameVirtualHost` directive defines the IP address for name-based virtual hosting. Apache Web server uses this IP address to receive requests for the name-based virtual hosts and name-based virtual hostnames resolve to this particular IP address. You can specify this IP address with the `NameVirtualHost` directive. There could be cases where a firewall or any other proxy server receives these requests and forwards them on a different IP address to the server. In such cases, you must specify the IP address of the proxy server that will respond to the requests. If you have several name-based hosts on multiple addresses, you must specify the `NameVirtualHost` directive for all of the addresses.

Note: A request made to a `NameVirtualHost` IP Address will be served neither by the main server nor by any default server. However, if a `NameVirtualHost` IP Address does not contain any virtual hosts for that address, then an exception is made.

The syntax for the `NameVirtualHost` directive is as follows:

```
NameVirtualHost addr[:port]
```

where,

`addr` - specifies the IP address

`port` - specifies the port on which Apache server accepts incoming requests

For example, to create a name-based virtual host that restricts to a specific port number, enter the code as shown in Code Snippet 2 in the `httpd.conf` file.

Code Snippet 2:

```
NameVirtualHost 196.214.370.45:80
```

- **ServerName Directive** - The `ServerName` directive defines the hostname and port used by the server for identification. You can use it while creating redirection URLs.

  The syntax for the `ServerName` directive is as follows:

```
ServerName fully-qualified-domain-name[:port]
```

Note: In the syntax, the port number is an optional specification. If no port number is provided, Apache assigns port 80 to the server name.

  For example, to define the server name, enter the following code in the `httpd.conf` file.

```
ServerName www.customer.com
```

  If the `ServerName` directive is not specified, then the server determines the hostname from the IP address by performing a reverse lookup. If you do not specify a port in the `ServerName` directive, then the server will use the port from the incoming request. You should specify an explicit hostname and port using the `ServerName` directive. In name-based virtual hosts, the `ServerName` directive in the `<VirtualHost>` section specifies which hostname should appear in the `Host:` header of the request to match the virtual host.

- **ServerPath Directive** - The `ServerPath` directive defines the URL pathname for the host to use with a name-based virtual host. You must use this directive only inside a `<VirtualHost>` section in the `httpd.conf` file.

  The syntax for the `ServerPath` directive is as follows:

```
ServerPath URL-path
```

  For example, to define the `ServerPath` directive, enter the code as shown in Code Snippet 3 in the `<VirtualHost>` section of the `httpd.conf` file.

  Code Snippet 3:

```
<VirtualHost  204.148.170.12>
...
ServerName www.myapacheserver.com
DocumentRoot /www/myapache/htdocs
ServerPath /myapache
...
</VirtualHost>
```

Concepts

In this example, the ServerPath directive ensures that a client request beginning with /myapache must be served by the virtual host.

- **ServerAlias Directive** - The ServerAlias directive defines alternate names for the hosts used in the name-based virtual host. You must use this directive along with a ServerName directive inside a <VirtualHost> section of the httpd.conf file.

The syntax for the ServerAlias directive is as follows:

```
ServerAlias hostname [hostname]...
```

For example, to define the alternate name for a server name, www.customer.com, enter the code as shown in Code Snippet 4 in the <VirtualHost> section of the httpd.conf file.

Code Snippet 4:

```
<VirtualHost 198.121.130.12:80>
  ServerName www.customer.com
  ServerAlias customer.com *. customer.com
  ...
</VirtualHost>
```

In the example, the <VirtualHost> directive contain instructions that apply to a specific hostname or IP address. The ServerName directive defines the hostname, and the ServerAlias directive specifies the alternate names for the hosts. This example creates two different aliases for the hostname: www.customer.com.

16.3 Creating an IP-based Virtual Host

In an IP-based virtual hosting, each Web site served by Apache server has a unique IP address assigned to it. You must assign multiple IP addresses to the server to support virtual hosting. You can achieve this by installing multiple network cards and assigning a unique IP address to each. Alternatively, you can assign multiple IP addresses to a single network interface by using virtual interfaces. This is known as IP aliasing.

16.3.1 Configuring Apache

You can configure Apache in two ways to support multiple hosts: You can run a separate httpd daemon for each hostname, or run a single daemon that supports all the virtual hosts.

You must use multiple daemons under the following conditions:

- Consider a scenario when the employees of company A want their company data to be sent to company B only using the Web and no other means. In such cases, two daemons can be used.

- Apache server can process requests specified by a 'wildcard' address or a specific addresses. If you are required to process the request for a specific address, you must configure Apache to listen to all specific addresses. In such a scenario, you will have to adjust the memory and file descriptor requirements for each IP address on the machine. Therefore, this is feasible only if you can afford to alter all of the requirements.

You can use a single daemon under the following conditions:

- If virtual hosts can share the `httpd` configuration

- If a machine has to process a large number of requests at a given time, separate daemons can result in performance loss as compared to a single daemon to process requests.

Web servers communicate through ports. Every port has a unique port number. The default port number for HTTP is 80. You can configure most Web servers to operate on almost any port number, provided no other program on the server is using that particular port number.

Consider a scenario where you want to host the Web site **www.abc.com** on the server. However, if you want to operate a second site, but do not have access to the domain name, and/or own no other IP addresses, you could instead use another port number. For example, `www.abc.com:81` for port 81, or `www.abc.com:8000` for port 8,000.

Concepts

Figure 16.1 displays an IP-based virtual host.

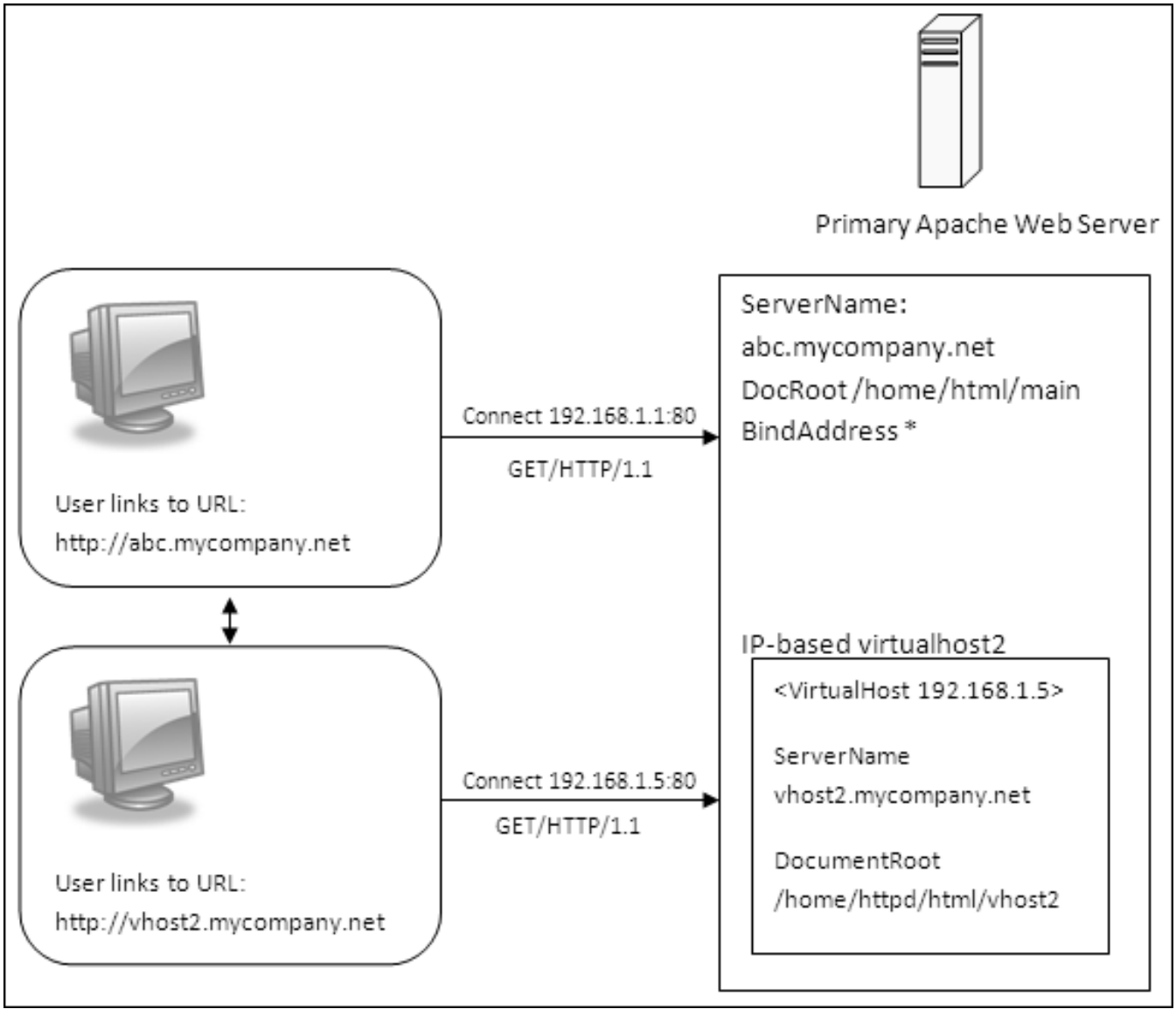


Figure 16.1: IP-based Virtual Host

In figure 16.1, there are two Web sites hosted on one Apache Web server. Both these Web sites have distinct IP addresses on the Web server. The first Web site, `http://abc.mycompany.net`, is assigned the IP address, `192.168.1.1`. On the other hand, the second Web site, `http://vhost2.mycompany.net`, is assigned the IP address, `192.168.1.5`. Any Web site hosted on this server will have a unique IP address.

To set up a virtual host on the virtual server, follow these two steps:

- **Register Domain Name** - Sets up a virtual host and registers the domain name for the virtual host.
- **Request the DNS Addition** - Registers the domain name and requests the entry of the domain to be added to the domain name records.

For example, to set up an IP-based virtual host, enter the code as shown in Code Snippet 5 in the `httpd.conf` file.

Code Snippet 5:

```
<VirtualHost 198.81.129.100:80>
  ServerAdmin webmaster@futurasoftltd.com
  DocumentRoot  /groups/futurasoft/www
  ServerName  www.futurasoftltd.com
  ErrorLog  /groups/futurasoft/logs/error_log
  TransferLog  /groups/futurasoft/logs/access_log
</VirtualHost>
```

In the example, the `<VirtualHost>` directive contains instruction that apply to the IP address `198.81.129.100`. The `ServerAdmin` directive defines the e-mail address that is included in every error message sent from the server to the client. The `DocumentRoot` directive specifies the directory from which the file server serves the requested documents and should not end with a slash. The `ServerName` directive defines the hostname that the server uses to identify itself. The `ErrorLog` directive specifies the name of the file to which the server adds the error it encounters. The `TransferLog` directive specifies the location of the log file. This virtual host listens to incoming requests on port 80. The directives that are not included in the virtual host section are inherited from the main server configuration.

You can also apply the same set of directives to multiple IP addresses and ports using the `<VirtualHost>` container.

For example, to set up a virtual host on different IP addresses such as `192.168.123.4` and `192.168.123.5`, enter the code as shown in Code Snippet 6 in the `httpd.conf` file.

Concepts

Code Snippet 6:

```
<VirtualHost 192.168.123.4:80  192.168.123.5:443>
  ...
  # virtual host directives
  ...
</VirtualHost>
```

To set up a virtual host for receiving the incoming requests on all the ports that Apache server listens to, enter the code as shown in Code Snippet 7 in the `httpd.conf` file.

Code Snippet 7:

```
<VirtualHost 192.168.123.8:*>
  ...
  # virtual host directives
  ...
</VirtualHost>
```

16.4 Creating a Name-based Virtual Host

Name-based virtual hosts use multiple hostnames for the same Web server IP address.

In a name-based virtual host, Apache runs multiple Web sites that share a single common IP address. It is a method of virtual hosting that requires a simple network configuration and no additional hardware or software. Apache server retrieves the required virtual hostname from the `Host:` header that the client sends as a part of an HTTP request. The server that receives this request knows only the IP address of the interface that receives it; it cannot find out which DNS name did the client use to determine that IP address.

To comply with HTTP/1.1, a second header must be present to identify the host that should process the request. This is usually the primary Apache server, but it may be any virtual host that has been defined in the Apache server configuration.

An HTTP/1.1 request would contain:

```
GET / HTTP/1.1 Host: abc.mycompany.net
```

The URL of the request is used to determine the hostname that the client browser includes in the `Host:` header.

The IP address must be associated with the network interface of the server.

Figure 16.2 displays a name-based virtual host.

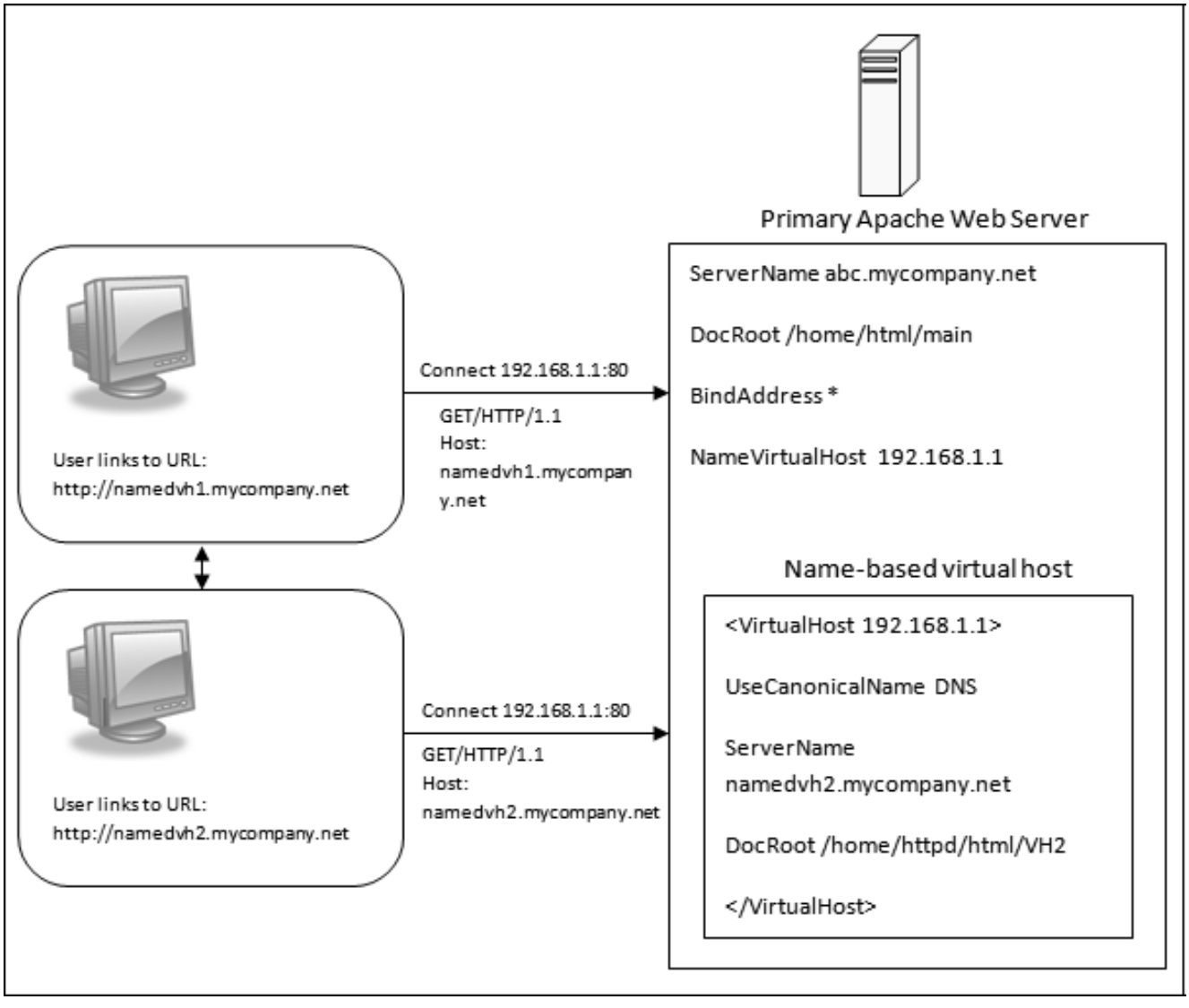


Figure 16.2: Name-based Virtual Host

In figure 16.2, there are two Web sites hosted on the same Web server. Both the Web sites, `http://namedvh1.mycompany.net` and `http://namedvh2.mycompany.net` are assigned the same IP address, `192.168.1.1`. Any Web site hosted on the server will share the same IP address and will be distinguished on the basis of its name.

To create a name-based virtual host, perform the following steps:

1. Select the IP address and the port on which the server must accept requests for the hosts. You can configure this by using the `NameVirtualHost` directive.

2. Create a <VirtualHost> block in the httpd.conf file for the different hosts that are to be served.

3. Designate the host to be served using the ServerName directive.

4. Specify the file system in which the contents on the <VirtualHost> must be stored.

For example, to create a name-based virtual host, enter the code as shown in Code Snippet 8 in the <VirtualHost> section of the httpd.conf file.

Code Snippet 8:

```
NameVirtualHost 210.145.160.42:80
<VirtualHost 210.145.160.42:80>
  ServerName www.mycompany.com
  DocumentRoot /groups/mycompany/www
  ServerAlias mycompany.com *.mycompany.com
  ServerPath  /mycompany
</VirtualHost>

<VirtualHost 210.145.160.42:443>
  ServerName www.abcompany.com
  DocumentRoot /groups/abcompany/www
  ServerPath /abcompany
</VirtualHost>
```

In the example, the NameVirtualHost directive designates an IP address for name-based virtual hosting. The <VirtualHost> directive specifies instructions for a specific hostname or IP address. The ServerName directive defines the hostname and the port that the server uses to identify it. The DocumentRoot directive specifies the directory for which the httpd file serves files. The ServerAlias directive defines the alternate names for the virtual host.

16.5 Combining IP-based and Name-based Virtual Hosting

IP-based virtual hosts uses the IP address of the connection to determine the correct virtual host to serve. Therefore, you need to have a separate IP address for each host. On the other hand, in name-based virtual hosting, the client reports the hostname as part of the HTTP headers, that allows different hosts to share the same IP address.

Name-based virtual hosting is simple, as you need to configure only your DNS server to map each hostname to the correct IP address and then configure the Apache HTTP server to recognize the different hostnames.

IP-based virtual hosting is used for the following reasons:

- In name-based virtual hosting, the client has to send the HTTP host header. Although most HTTP/1.0 Web browsers implement it as an extension, some earlier clients are incompatible with it.
- Name-based virtual hosting is incompatible with SSL secure servers.
- Networks implementing bandwidth management techniques are incompatible with name-based virtual hosts. For such networks, it is mandatory to operate hosts on distinct IP addresses, because the network cannot differentiate between these hosts.

Name-based and IP-based virtual hosts can be configured in the same server configuration if the name-based and IP-based virtual hosts share a different IP address.

To configure Apache server for name-based and IP-based virtual hosting, enter the code as shown in Code Snippet 9 in the `httpd.conf` file.

Code Snippet 9:

```
Listen 80

NameVirtualHost  204.148.170.10
<VirtualHost 204.148.170.10>
  DocumentRoot /www/mycompany1
  ServerName www.mycompany1.com
  ServerAdmin  webmaster@mycompany1.com
</VirtualHost>

<VirtualHost 204.148.170.10>
  DocumentRoot /www/mycompany2
  ServerName www.mycompany2.org
  ServerAdmin webmaster@mycompany2.org
</VirtualHost>

# Creating IP-based virtual hosts

<VirtualHost 204.148.170.15>
  DocumentRoot /www/mycompany5
  ServerName www.mycompany5.com
  ServerAdmin webmaster@mycompany5.com
  ErrorLog /www/logs/mycompany5_error_log
  TransferLog  /www/logs/mycomany5_access_log
</VirtualHost>
```

Concepts

```
<VirtualHost 204.148.170.16>
  DocumentRoot /www/mycompany6
  ServerName www.mycompany6.gov
  ServerAdmin  webmaster@mycompany6.gov
  ErrorLog /www/logs/mycompany6_error_log
  TransferLog  /www/logs/mycompany6_access_log
</VirtualHost>
```

Summary

- Virtual Web hosting means running more than one Web site on a single machine.
- There are two types of virtual hosts, IP-based virtual hosts and Name-based virtual hosts.
- In an IP-based virtual hosting, each Web site has its own IP address.
- In name-based virtual hosting, the server depends on the client to provide the hostname as a part of the HTTP header.
- Apache can be configured to support name-based and IP-based virtual hosting.
- The `VirtualHost` directive can be used to enclose the directives applicable for a specific virtual host.
- The `NameVirtualHost` directive is used to specify the IP address for name-based virtual hosting.
- The `ServerName` directive is used to specify the hostname and port used for accepting requests.
- The `ServerAlias` directive is used to specify alternate names for a name-based virtual host.

Check Your Progress

1. Which directive is used for setting the hostnames that address the same server?

   a. `ServerName`

   b. `ServerAlias`

   c. `ServerPath`

   d. `NameVirtualHost`

2. Which directive is used for setting the hostname and port?

   a. `NameVirtualHost`

   b. `ServerName`

   c. `ServerPath`

   d. `ServerAlias`

3. Which directive encloses the directives that configure a virtual host?

   a. `NameVirtualHost`

   b. `VirtualHost`

   c. `ServerName`

   d. `ServerPath`

Check Your Progress

4. Which module provides the virtual host directives?

   a. `mod_alias`

   b. `mod_vhost_alias`

   c. `core`

   d. `mod_headers`

5. In _____, the client sends a `Host: header` in each HTTP request to identify the virtual host it requires.

   a. Name-based virtual hosting

   b. IP-based virtual hosting

   c. Web hosting

   d. Dedicated hosting

Session

17 Virtual Web Hosting (Lab)

Lab Guide

Objectives

At the end of this session, the student will be able to:

- *Configure an IP-based virtual host in Apache Web Server.*
- *Configure a name-based virtual host in Apache Web Server.*

The steps given in this session are detailed, comprehensive, and carefully thought through in order to meet the learning objectives and understand the tool completely. Please follow the steps carefully.

Part I - For the first 1.5 hours:

Creating an IP-based Virtual Host

Virtual Web Hosting means running more than one Web site on a single machine. A virtual host configuration requires a registered domain name and DNS addition.

To setup IP-based virtual host, perform the following steps:

1. **Open the `httpd.conf` file in the gedit text editor.**

2. **To configure a virtual host, enter the following code in the `httpd.conf` file.**

```
Listen  80
<VirtualHost 192.120.110.41>
 ServerAdmin webmaster@myapacheserver.com
 DocumentRoot /usr/local/apache2/htdocs/myapacheserver
 ServerName www.myapacheserver.com
 ErrorLog mylogs/myapacheserver_error_log
</VirtualHost>

<VirtualHost  192.120.110.45>
 ServerAdmin   webmaster@apachetroubleshooter.org
 DocumentRoot   /usr/local/apache2/htdocs/apachetroubleshooter
 ServerName   www.apachetroubleshooter.org
</VirtualHost>
```

Figure 17.1 displays the `<VirtualHost>` directive in the `httpd.conf` file.

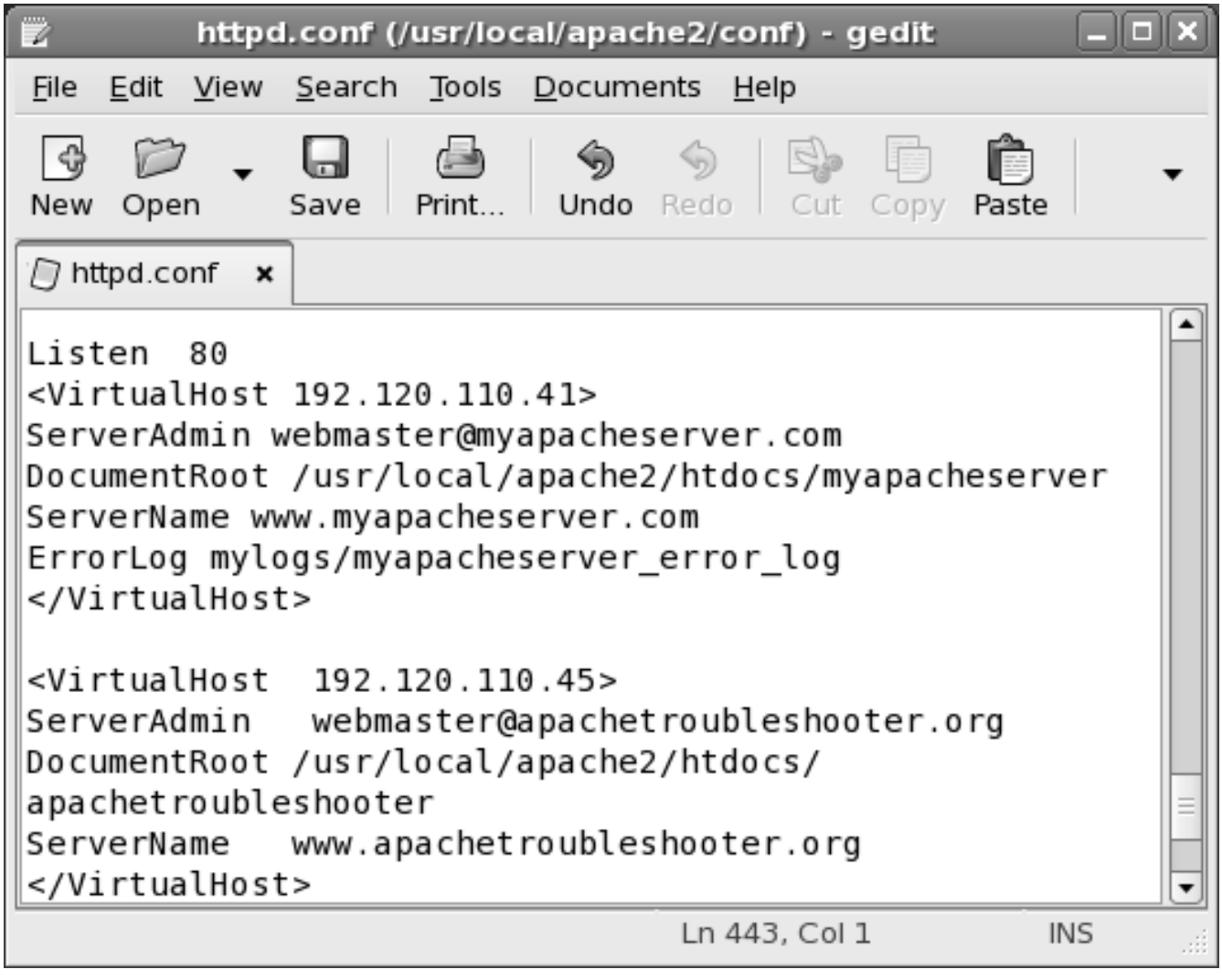


Figure 17.1: Configuring an IP-based Virtual Host

3. **Save the changes.**

4. **Restart Apache Web server.**

In the example, the `<VirtualHost 192.120.110.41>` statement specifies the IP address of the Web site on the server. The `VirtualHost` directive enables you to create different Web sites and assign multiple IP addresses to a single machine. You can configure an IP-based virtual host using the additional configuration file `httpd-vhosts.conf` located in the `extra` directory. This file can be included by uncommenting the following line present at the end of the `httpd.conf` file.

```
#Include conf/extra/httpd-vhosts.conf
```

Creating Name-Based Virtual Host

IP-based virtual hosts use the IP address of the server to process requests. Therefore, each host requires a separate IP address. However, a name-based virtual host requires you to configure the DNS server to map the hostname to the correct IP address. While creating a named-based virtual host you must define the name for the host, and select the IP address and the port on the server that will accept the request for the host. The port number, separated by a colon, follows the IP address in the `VirtualHost` directive of the `httpd.conf` file.

To create a name-based virtual host, perform the following steps:

1. **Open the `httpd.conf` file in the gedit text editor.**

2. **Enter the following code in the `httpd.conf` file:**

```
NameVirtualHost 198.160.210.10:80
<VirtualHost 198.160.210.10:80>
ServerName www.mydomain.com
 DocumentRoot /usr/local/apache2/htdocs/mydomain
 </VirtualHost>

 <VirtualHost 198.160.210.10:400>
     ServerName www.company.com
     DocumentRoot /usr/local/apache2/htdocs/company
 </VirtualHost>
```

Figure 17.2 displays the `<VirtualHost>` directive with name-based values in the `httpd.conf` file.

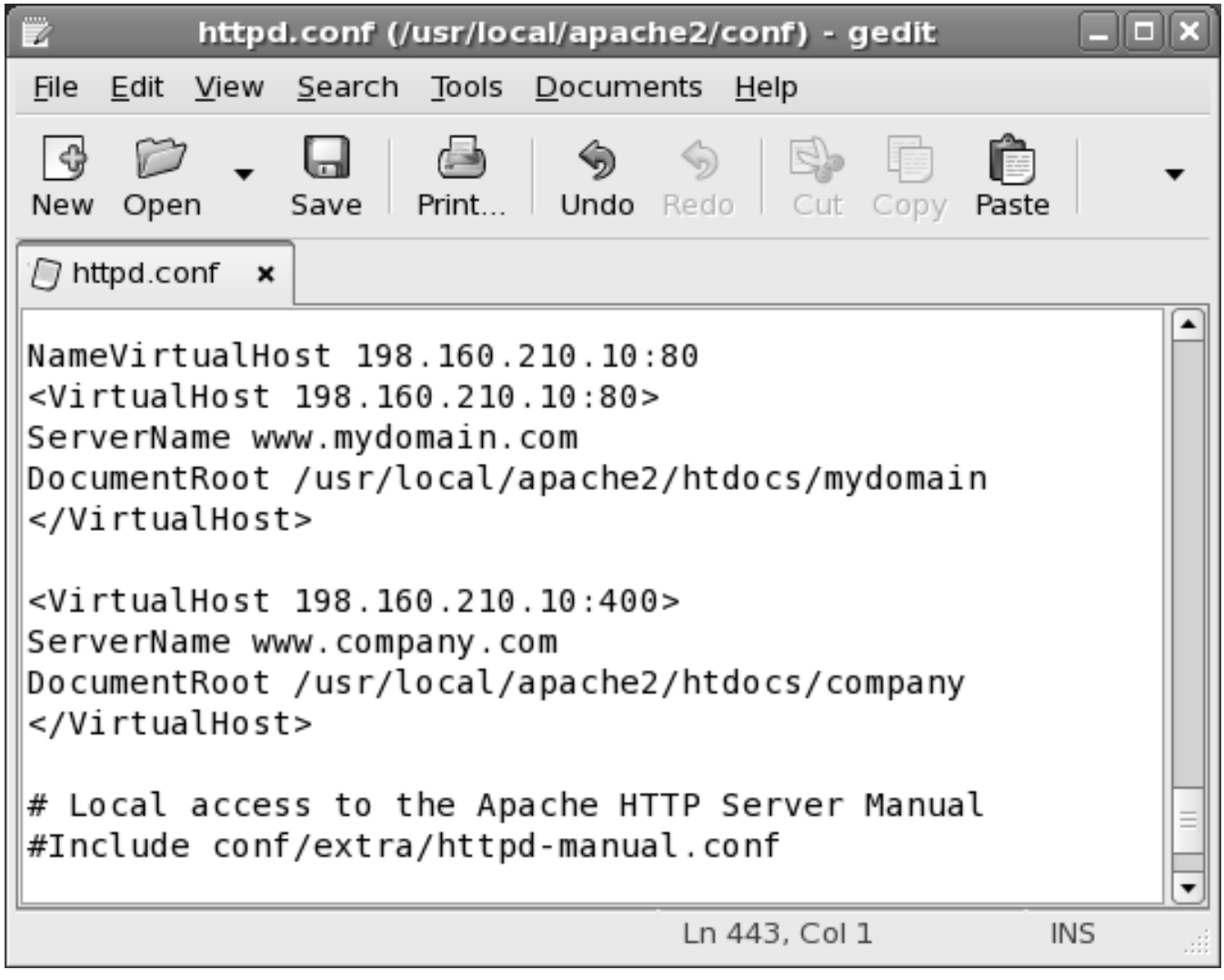


Figure 17.2: Configuring a Name-Based Virtual Host

3. **Save the changes.**

4. **Restart Apache Web server.**

The directories for the name-based virtual hosts must contain an `index.html` file to view the Web site in the Web browser.

To create an `index.html` for the **mydomain** Web site, perform the following steps:

1. **Open a new file in the gedit text editor.**
2. **Enter the following code:**

```
<html>
  <body>
    <h1>This is the mydomain home page!</h1>
  </body>
</html>
```

3. **Save the file as `index.html` in the `/usr/local/apache2/htdocs/mydomain` directory.**
4. **Open the Mozilla Firefox Web browser.**
5. **Enter `http://localhost/mydomain` to view the index page of the mydomain Web site.**

Figure 17.3 displays the homepage of the mydomain Web site.

Figure 17.3: Displaying the Home Page of a Name-Based Virtual Host

Do It Yourself

1. Setup a virtual host on the port 204.120.110.40 for the site www.jobs.com.

2. Create a name-based virtual host that listens on the port 443.