# Contents

Acknowledgments xi
Introduction xiii

## Part One Knowledge Matrix 1

### Chapter 1 Installation and Configuration 3

- Objectives 3
- Theory of Operations 3
- History of Apache 4
- Apache Today 5
- How Does Apache Work? 5
- How to Obtain Apache 10
- Overview of Content Negotiation 11
- Base Systems 18
- Preparing Linux 18
- Introduction to Packages 21
- Installation 29
- System Utilities 34
- The httpd Daemon 34
- Setting Up Apache 45

### Chapter 2 System Administration 55

- Objectives 55
Lab VI  Apache and Tomcat  223
Purpose  223
Theory  223
Lab Exercises  224
Questions  225
Answers  225
Advanced Questions  225

Lab VII  Configuration of a Proxy  227
Purpose  227
Theory  227
Installing mod_proxy  228
Configuring httpd  228
Configuring the Client  229
Lab Exercises  229
Installing mod_proxy  229
Configuring httpd  229
Configuring the Client  230
Questions  230
Answers  231
Advanced Questions  231

Lab VIII  URL Rewriting  233
Purpose  233
Theory  233
Lab Exercises  236
Questions  237
Answers  237
Advanced Questions  237

Lab IX  Create a Custom Log for Apache  239
Purpose  239
Theory  239
Lab Exercises  241
Questions  242
Answers  242
Advanced Questions  242
## Lab X

**Benchmark Your Server**

- Purpose: 243
- Theory: 243
- Lab Exercises: 245
- Questions: 247
- Answers: 247
- Advanced Questions: 247

### Part Three

**Practice Questions and Answers** 249

- Practice Questions: 251
- Answers: 273
- Glossary: 281
- Index: 290
With the growing body of certificate holders and with the development of Level II, the team effort required to implement the Sair Linux and GNU curricula has been amazing. As a result, our acknowledgment list continues to grow to reflect the greater involvement in the open source community:

Bill Patton, Steering Committee Chair, Compaq; Cheryl Foiles, Productivity Point International; Mark Muth, Prometric; Dan Kusnetzky, International Data Corporation; Kevin Whittier, MSC Software; Timothy Ney, Free Software Foundation; Eric S. Raymond, Consultant and Open Source Software Author; Richard Stallman, Founder of the GNU GPL and Free Software Movement; Tim Angle, University of Mississippi; Cary Sullivan, John Wiley and Sons Inc.; Jon “Maddog” Hall, Linux International; Bill Noyes, Magellan Group, Inc.; Evan Blomquist, Technical Trainer; Doug Dickerson, Compaq; Stephen Solomon, Course Technology; Bruce Perens, Hewlett-Packard; Stuart Trusty, Linux Labs; Bill Arvidson, Mission Critical Linux; Dan Greening, Motorola Computer Group; Bryan Ochs, New Horizons Computer Learning Centers; Mark Langston, The SysAdmin Company; Deb Murray, Uniforum; Kit Cosper, VA Linux Systems; Ken Kousky, Wave Technologies; Dr. James Stanger, ProSoft Training.com; Steven Wright, Caldera Systems; Paul Wildrick, CyberstateU.com; Tina Bush, Element K; Don Corbet, Team Linux Corporation; Harald Bertram, AG. Abt. Training; SuSE Linux Solutions; John Terpstra, Caldera; Jeremy Siadal, Intel Corporation; Kerry Hodgins, Corel Corporation; Karen Letain, Wave Canada; and Jim Lacey, Linuxcare.

Continuing the process of shaping mountains of trivia into some semblance of organization and developing that information so that it can be communicated quickly and effectively is the task of the Sair Development Team. These are the people who assisted with the production of the Apache material:
President and CEO: Dr. Tobin Maginnis
Managers: Ross Brunson, Trish W. Kemerly, Carlos Pruitt, and Paul Tate
Assistant manager: Albert Phillips
Contractors: Ancilla Allsman, Joseph Cheek, Mike Hartley, Tony Inson, Alex Larson, Rasmus Lerdorf, Shawn McKenzie, Martin Pool, Ed Riddle, Andrew Scott, Troy Vitullo, and Susan Wood
Technical contributors: Stephen Agar, David Austin, Tim Bauer, Jerald Jones, and Omar Wilson
Technical editor: Wesley Duffee-Braun
Editors: Annie Current, Summer Hill, Jamie Murphey, and Samantha Rayburn
Quality assurance: Jenna Johnson and Nikki Andersen
Graphics: Wesley Duffee-Braun and Leah Riley

The Sair Development Team would also like to thank the rest of the employees at Sair Linux and GNU for their support—albeit technical, promotional, or otherwise—and for contributing to the Linux community through their efforts:

Managers: Bob Buntyn, Les Driggers, Hollis Green, Leigh Jennings, Alex Lundy, and Lenny Sawyer
Assistant managers: Jimmy Palmer and Elizabeth Bonney
Technical contributors: Eric Blankenship, Beau Bourgeois, Jeff Britt, Brett Brown, Jayaprakash Chilumula, Joann Chong, John Furr, Stephen Goertzen, Scott Hicks, Jacob Jenkins, David Kearns, Rebecca Love, Chris Mavromihalis, Michael McGuire, Damir Mehmedic, Chance Mobley, Ken Montgomery, Rufus Peoples, Ben Pharr, Evan Rouse, Tyler A. Simon, Robert Thompson, Sunder Upadhyay, William Vaughan, Sudharshan Vazhkudai, James Walker, James Webster, and Dana Wilson
Test development: Tammy Betts, Michael McGuire, and Richard Swinney
Programmers: Michael Broadwater, Michael Calvi, Chris Mavromihalis, Rufus Peoples, William Vaughn, and Jeremy Webster
Web and database: Shafi Al-meher, Steve DeVries, Nileshwar Dosooye, Janakiram Govindaraju, Mark Ketcham, Arlene Pereira, Kubenthiran Ramanathan, Errol Sayre, and Julie Seay
Systems: Patrick Hood, Ross Reed, and William Taylor
Production: Brandi Bailey and Scott Rains
Sales and marketing: Wendy Chambers, Scott Thompson, Shellie Ross, Karan Mullen, Haley Teague, Beth Odum, Shelley Robbins, Elise Knapp, Ashly Ray, and Lori Redding

Again, thanks to John Wiley & Sons Computer Publishing, especially Cary Sullivan, for their foresight, effort, and belief in the future of free, open source software.
Welcome to the Level II study guide series for the Sair Linux and GNU Certified Engineer. This is the Apache study guide for Sair Linux and GNU Certification, Exam 3X0-202. The Apache study guide is one of the optional or elective subjects for the Sair Linux and GNU Certified Engineer (LCE) certificate.

Requirements for the Sair Linux and GNU Certified Engineer (LCE) include passing the Core Concepts and Practices exam, 3X0-201, plus three additional exams from other elective subjects. In addition to this study guide, other possible elective areas include Sendmail and mail system components, Samba and resource-sharing components, and PHP and scripting. See www.linuxcertification.com for a complete listing of available courses. Each course consists of a minimum of 32 classroom contact hours, making the LCE equivalent to an additional 128 hours of classroom contact beyond the LCA. We see achievement of the LCE to be evidence of the ability to perform as an advanced Linux systems specialist, as a system administrator supervisor, or in some other form of Linux system management.

Exams are offered at Prometric and VUE testing centers. Prometric offers the Sair exams at any one of its 3,500 testing centers in 141 countries. To take a test, simply call the Prometric registration line at 1-888-895-6717. Ask the customer service representatives about the 3X0 series of exams, and they will answer any test-taking questions related to the test, describe available local testing centers, and, if requested, schedule an exam. It is also possible to register for tests online at www.prometric.com. VUE also offers the Sair exams at any one of its 2,500 testing centers in 110 countries. To take a test, call the VUE registration line at 1-952-995-8800. Ask the customer service representatives about the 3X0 series of exams, and they will answer any test-taking questions related to the test, describe available local testing centers, and, if requested, schedule an exam. It is also possible to register for tests online at www.vue.com/sairlinux.
Preparing for test content is paramount in criteria-based certification tests, and Sair takes pride in the coordination of test topics from the Knowledge Matrix, objectives, competencies, study guides, and exams. These topics specify exact criteria that the candidate must meet. The tests directly measure these topics, and results are reported with a detailed summary for each of these six areas: theory of operation, base system, shells and commands, system utilities, applications, and troubleshooting.

Each exam consists of 50 questions that can be answered, reviewed, and changed for up to 60 minutes. Typical test takers use about 45 minutes to complete the exam. Successful completion of each test requires 74 percent correct, or 37 correct answers. Unlike tests that assign a rank to a test taker relative to all other test takers, Sair emphasizes mastery of material. Results of the test directly inform the candidate of the mastery level in each area, allowing each student to focus future studies on areas of relative weakness. Prospective employers can also use the detailed summary to evaluate job applicants’ or employees’ areas of strength. Sair is unique in this regard. Some other certification examinations supply the test taker with only a relative score of some type, without a raw score, percentage correct, or other measure for evaluating his or her performance. The effect of this practice is to leave the test taker who fails without guidance as to how to prepare for reexamination.

**Minimum Candidate Requirements**

The Sair Linux and GNU tests were not designed for the novice computer user. It is assumed that the candidate has approximately two years of computer experience and has experience in the configuration of one or more operating systems. For example, the candidate should be familiar with basic hardware concepts, such as CPU, cache, memory, interface adapters, hard disks, and networks. The candidate should also be familiar with basic operating system concepts, such as booting, file access, and device drivers. Finally, the candidate should know basic commands and the use of a Unix-type editor, such as Joe, Pico, Vi, or Emacs.

**Knowledge Matrix**

The test is based on the Apache Knowledge Matrix shown at www.linux-certification.com. Examination topics are listed here. Note that while the major topics listed in the Knowledge Matrix will not change, some subtopics may be added as needed to update the material. Please check the Sair Web site for any additions to this list.
Apache and Installation and Configuration

Theory of Operation

1.1.10 History of Apache
1.1.20 Apache Today
1.1.30 How Does Apache Work?
   A. Web server concepts
   B. Apache conventions
   C. Directives
   D. Handlers
   E. Logging
   F. Modules
   G. Lynx
1.1.40 How to Obtain Apache
1.1.50 Overview of Content Negotiation
   A. Type
   B. Encoding
   C. Language
   D. Multiviews
   E. Browsers and HTTP
   F. Style sheets
   G. Java
   H. JavaScript

Base Systems

1.2.10 Preparing Linux
   A. RAM
   B. Hardware
   C. Hard drive
   D. Kernel
   E. File handles and inodes
1.2.20 Introduction to Packages
   A. RPM security tasks
   B. Locate security updates
   C. Add modules via RPMs
   D. Start Apache
   E. Verify functionality locally
   F. Verify functionality over a network
   G. Source RPMs
1.2.30 Installation
   A. Compiling
   B. Dynamic shared objects
   C. Installing
   D. Testing before installing

**System Utilities**

1.4.10 The httpd Daemon
   A. Defaults
   B. Customization
   C. Modules and SSL
   D. Module performance and functionality
   E. Enabling modules
   F. Configuration files
   G. Configuring Apache logs
   H. Alias
   I. Modules
   J. Configuration methods
   K. Starting httpd at boot

1.4.20 Setting Up Apache
   A. Starting and stopping Apache
   B. Default index
   C. Basic configuration
   D. Port 80
   E. ServerType stand-alone
   F. StartServers 5
   G. MinSpareServers 5 and MaxSpareServers 10
   H. MaxClients 150
   I. MaxRequestPerChild 0
   J. Basic configuration
   K. Document directory configuration
   L. Options directive

**Apache and System Administration**

*Theory of Operation*

2.1.10 Being a Webmaster
   A. Tuning

2.1.20 Network Access Installation
   A. Allow and Deny
B. Max Clients
C. TCP/IP version
D. Hostname lookups
E. FollowSymLinks and SymLinksIfOwnerMatch
F. AllowOverride
G. Content negotiation
H. Process creation
I. Process death
J. KeepAlive

2.1.30 Introduction to Virtual Hosting
A. Directories
B. Network addresses

2.1.40 Introduction to Apache Modules
A. What is CGI?
B. When to use CGI
C. PHP – mod_php4
D. MySQL
E. mod_perl

2.1.50 Introduction to the Apache API
A. Modules
B. Introduction to phases
C. Phases in detail
D. The future: Apache 2.0

2.1.60 Introduction to Logging

**Base Systems**

2.2.10 Multiple Daemons

2.2.20 Configuration
A. Running multiple daemons
B. Multiple daemon verification

2.2.30 Number of httpd Processes

2.2.40 Alias

2.2.50 CGI Scripts
A. ScriptAlias directive
B. AddHandler directive

2.2.60 How to Configure CGI
A. Enable within certain directories
2.2.70 Apache Initialization

2.2.80 Log Files
   A. /var/apache/logs/access_log
   B. host
   C. ident
   D. authuser
   E. date
   F. request
   G. status
   H. bytes

2.2.90 Log File Formats
   A. Design your own log files format

**Shells and Commands**

2.3.10 Benchmarking
   A. ab

**System Utilities**

2.4.10 Creating a CGI script
   A. Content of CGI output
   B. Server environment
   C. Form handler
   D. Basic redirect

2.4.20 Performance Monitoring
   A. mod_status
   B. ExtendedStatus
   C. server-info

2.4.30 Some Good Log Analysis Tools
   A. Webalizer
   B. Analog
   C. Wusage
   D. Summary
   E. Logresolve

**Apache and Networking**

**Theory of Operation**

3.1.10 What Is TCP/IP and How Does Apache Use It?
   A. HTTP
   B. Headers
3.1.20 What Is HTTP?
   A. HTTP/1.1

3.1.30 Multiple Hosts

**Base Systems**

3.2.10 Virtual Hosting
   A. Configuring separate daemons

3.2.20 Directing the Request to a Virtual Host

3.2.30 Single Daemon/Virtual Hosting

3.2.40 IP-Based Virtual Hosting
   A. device
   B. aliasnumber
   C. address

3.2.50 Name-Based Virtual Hosting
   A. Virtual hosting with one IP address
   B. Virtual hosting using mixed methods

**Shells and Commands**

3.3.10 URL Rewriting mod_rewrite
   A. CondPattern TestString

**Apache and Security**

**Theory of Operation**

4.1.10 Security Concerns

4.1.20 Security Policies
   A. hosts.allow and hosts.deny
   B. motd and issue files
   C. U.S. encryption export laws

4.1.30 Authentication

4.1.40 Securing Apache
   A. Apache user

4.1.50 Vulnerabilities
   A. Trojan horses
   B. Viruses
   C. Worms
   D. Spoofing
   E. Buffer overruns
4.1.60 Security Issues with CGI
   A. System calls
   B. Buffer overruns

4.1.70 The Apache Proxy Server
   A. Advantages of the Apache proxy
   B. Obtaining the Apache proxy
   C. Obtaining documentation
   D. mod_proxy

4.1.80 Firewalls
   A. Types of firewalls
   B. Proxy server firewalls
   C. IP masquerading proxy servers
   D. Firewalls and network architecture
   E. Securing the firewall machine
   F. What to do if attacked

4.1.90 Password Protection

**Base Systems**

4.2.10 Apache, Users, and Groups

4.2.20 Permissions

4.2.30 Access Control
   A. AllowOverride
   B. Order, Allow, and Deny
   C. Testing
   D. Anonymous access

4.2.40 Setting Up the Apache Proxy
   A. Proxy-specific directives
   B. Server-side configuration

4.2.50 Security Fundamentals
   A. Permissions
   B. Scripting
   C. SuExec
   D. Matrix of ideal permissions

4.2.60 User Access Control
   A. Common access controls

4.2.70 Enabling Content from Home Directories
   A. UserDir html
4.2.80 Access Directives
   A. AuthType
   B. AuthName
   C. AuthName “Private Documents”

4.2.90 Defining within httpd.conf
   A. Defining within control files

**Shells and Commands**

4.3.10 Checksums
   A. PGP and checksums

4.3.20 Password Authentication
   A. User
   B. Group

**System Utilities**

4.4.10 Server-side Includes
   A. Basic commands
   B. Variables

4.4.20 XSSI
   A. Shell commands
   B. Including files
   C. Executing scripts
   D. Embedding XSSI's
   E. Conditional statements

4.4.30 ModSSL vs. Apache+SSL
   A. SSL—mod_ssl
   B. SSL—Secure Sockets Layer (SSL)
   C. Implementing SSL in Apache
   D. Legal issues

**Apache and Troubleshooting**

5.1.10 Online Troubleshooting Resources
5.1.20 Tracking Down an Apache Core Dump
5.1.30 Some Useful Sites
5.1.40 Configuration Issues
5.1.50 Logging Problems
Objectives

- Define the evolution of Apache.
- List the Apache levels of configuration.
- Define Apache configuration options and methods.

Theory of Operations

Almost two-thirds of the Web servers on the Internet use Apache (see Figure 1.1). Apache is common for the same reason that screwdrivers and crescent wrenches are common—doing a job right takes the proper tool. Apache is an excellent tool for World Wide Web hosting.

Although Apache may not perform well in some benchmark tests, it performs extremely well in the field. It is a fully capable, industrial-strength Web server that is able to compete with other servers that are much more costly to purchase or license. It is extremely stable and has been heavily tested over the years in many different environments and platforms. The
developers of Apache emphasize performance above all else. Webmasters will appreciate the richness of features that can be implemented on a site served by Apache.

**History of Apache**

Apache is based on NCSA’s `httpd`, a daemon that was first used in the early years of the Internet. Rob McCool was the creator of the NCSA Web server and continued his work on it until 1994. No single programmer took his place, but the server continued to grow in popularity. After his departure from the project, incompatibilities between versions began to develop. Web administrators had to make changes on their own if they wanted to fix bugs or add features. Eventually, a group of administrators began working together to regain control of the project. Their efforts created a single path through which patches and enhancements to `httpd` could be submitted. The name of the project is derived from its origins: a series of patches applied to the original `httpd` daemon. The project came to be known as “a patchy server,” and from there, Apache server.
Apache Today

Apache possesses a level of complexity that easily surpasses some operating systems. Although it is not necessary to learn every feature to gain a functional understanding of Apache, an introduction to the overall design of the environment and methods used is a good start.

The purpose of this section is to describe some of the configuration styles that Apache uses and to provide an overview of some commonly encountered server features. Later in the course, when topics are described in more detail, a context is developed on which to base further learning. Ultimately, the goal of this book is to teach the user to set up an Apache server.

The user can also go to the Apache Web site, www.apache.org/docs/, and study the full documentation in depth. The online user documentation for Apache is one of the most well-written and accessible sources of information for any project.

How Does Apache Work?

Apache is a monolithic Web server. It looks to centralized configuration files for all parameters that are not compiled. Apache works much like the Linux kernel. It forks off child processes, accepts loadable modules, and operates via user-specified configurations.

Web Server Concepts

On a system functioning as a Web server, the server process listens for incoming requests on a specified port of the network interface and responds to them appropriately. Originally, a Web site consisted of a directory tree of static documents, which would be served on request. Special functions could be handled by CGI scripts that are executed on request.

Over the years, people have developed variations of this basic function, requiring a rethinking and retooling of the original Web server definition to the point that today’s Web users and authors expect a certain amount of active participation by the server. Dynamic content is everywhere now; today’s server must not only provide a Web page, it must create it on user demand. Although the basic process is the same, expected enhancements place much more demand on a server than what was required when static content was the only possibility.
Apache Child Processes

Apache works by splitting itself and having the Apache copy handle the process. Child processes are the processes that actually dole out content. When Apache receives a request, it chooses one of its idle child processes to handle the request. When the child is done, it returns to an idle state, ready for more work.

Every child completes its list of requests before dying. Killing a child process after it completes its list of requests also kills any side effects of those processed requests, such as memory leaks. The user can determine the number of children Apache spawns at one time, as well as their life span. By default, each child handles an unlimited number of requests.

Apache Conventions

A good way to learn how Apache functions is through its configuration. The first level of configuration occurs when Apache is compiled. RPM- and Debian-based installations decide many things by default, but when compiling from source these options can be set:

- Whether Apache uses Dynamic Stored Objects or statically linked modules (modules will be described in further detail later in the chapter)
- If static modules are chosen, tell Apache which modules to compile into the server at this point
- Default location
- Layout of Apache’s files and directories

The next level of configuration occurs in httpd.conf after the installation. The httpd.conf is a global preferences file that controls all aspects of the server, including the following:

- Who can access the Web server
- What dynamic modules the server loads
- Where certain functions occur
- What kind of information Apache will log
- The location of the virtual Web sites Apache controls
- What content is allowed
The last level of configuration may occur in .htaccess files, depending on what is stated in httpd.conf. These are small, local files that contain specific orders that apply to the directories in which they reside.

The httpd.conf file contains more than three-fourths of the Apache configuration options. It allows administrators to configure Apache to an optimal level at performance. This is possible because, based on users’ needs, Apache offers many levels of control through the httpd.conf file. These levels are called scopes. A scope can apply to an entire system, a virtual host, an absolute directory path, or even a single file. Wild cards can be used with scopes to give greater freedom. Containers are used to define each scope.

The following are the four main containers:

```
<Directory /path>
# This container applies to the filesystem location of a directory.
# For example, /home/user/public_html would point to user's public_html directory.
Directive
</Directory>

<Location /~*>  
# Location applies to a url. For example ~/user would point to user's public_html directory.
Directives
</Location>

<VirtualHost host_name>
# VirtualHost applies to a virtual web site you are hosting, such as mysite.com
Directives
</VirtualHost>

<Files file_name>
# Applies to a file name, no matter where it is located. If file_name were index.html, then all instances of index.html (no matter what their location) would be affected by your directives.
Directives
</Files>
```

**Directives**

A directive is a command given to Apache that controls its behavior. Directives are a single line of text with the following format:

```
Directive [option [option]]
```
As shown, there is more than one option for your key word. Although there can be more than one directive in a given scope, each directive must occur on its own line. Directives that do not reside in any container apply to all scopes.

**Handlers**

When a user requests a file, Apache decides what to do with it. For example, if the user accesses a file called hiccup.cgi, Apache must know to execute that file as a CGI script and not try to display it as an HTML file. Handlers help you direct Apache to the correct helper application. For example, add a handler called cgi-script and apply it to all files with an extension of .cgi at the end.

**Logging**

Apache logs almost everything that happens to it, from errors to http transactions and even cookies. The user can set the level of logging and what is logged in the httpd.conf file.

**Modules**

Modules are chunks of software (usually written in Perl or C) that allow customization of Apache. For instance, for Apache to use Server Side Includes, the user must install that module.

Even though Apache was originally developed with this concept in mind, it was not available when the first versions of Apache were being developed. Still, because it was already being planned at that time, integration of dynamic module loading into the Apache server came relatively easily.

In early versions of Apache, modules were statically linked into the httpd application during compilation. The specific configuration of a server was defined during the configure process before compilation. Using this method, many helpful modules were developed over the years, making a tremendous amount of customization available for the Apache server. Since the release of Apache version 1.3, it is possible to compile the server and the modules to support dynamic loading. With this option compiled, the configuration of the server can be changed by simply restarting the server after editing the configuration file (rather than recompiling the server). Different instances of the server, with different features, can be started at separate times or simultaneously by using different configuration files running the same base installation.