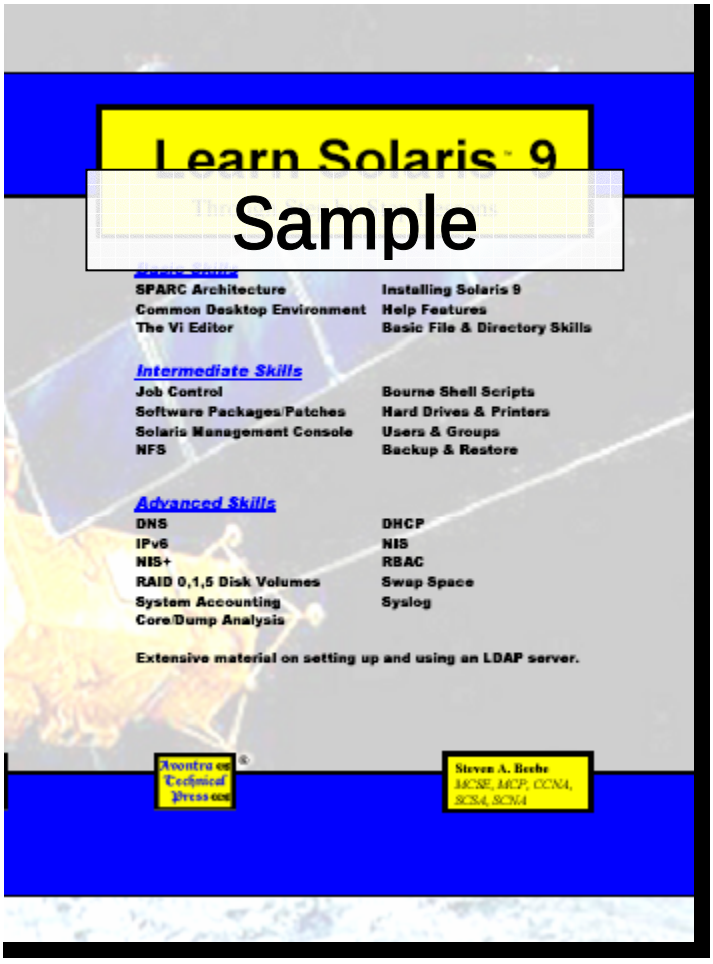
Please feel free to copy and redistribute this file.



This Adobe Acrobat file is an authorized limited sample of the book

Learn Solaris™ 9

Through Step by Step Lessons

Avontra Techncial Press® : Publisher
Steven Beebe : Author

This limited sample only features chapters 4, 12 and 31 of the book

Learn Solaris 9
Through Step by Step Lessons

If you are happy with our work, this book can be purchased from the book's official website:

http://www.teachmesun.com

This Adobe Acrobat file **(ATP-Book-Sol9-Sample.pdf)** may be freely copied and redistributed under the following copyright permissions:

1. This file can be freely copied and redistributed, but the file's name **ATP-Book-Sol9-Sample.pdf** and contents can not be modified in any manner by the distributor or recipient of the file.

2. This file's internal material (such as text, pictures, figures or diagrams, etc.) may not be incorporated into any other electronic or non-electronic media.

3. Anyone can copy, print, view, and redistribute this Adobe Acrobat file in it's unaltered condition, but that does not give anyone permission to copy the book *Learn Solaris 9 Through Step by Step Lessons* through any electronic or non-electronic means.

SPECIAL NOTE:

In this sample book, Chapter 4 reference **`user11`** this is a generic user created in Chapter 2. To follow the lessons in chapter 4 create **`user11`** with these steps:

1. Login as the root user (*The root user's login name is* **`root`**)
2. Type these commands
```
useradd -m -d /usr/user11 user11
cp /usr/user11/local.profile /usr/user11/.profile
passwd user11 ( set the password to user11 )
```
3. Log off the system

ORDER FORM – for the full edition of this book

Simply print the order form, fill out the order form and then send a check to:

Avontra Technical Press
12690 Falcon Drive
Colorado Springs, CO 80908

Your book should arrive one week after we receive your check (Alaska and Hawaii will experience shipping delays.) This order form is only good for the United States of America, Hawaii and Alaska. Prices indicted on this form are only good through 02/2003.

Name_____

Company_____

Street _____

City _____

State _____

Zip Code _____

Your personal information will not be sold or given away.

Payment: **$ 39.95 per copy**
Shipping: **$ 5.00 per copy**
Total: **$ 44.95 per copy**

Number of Copies [ ]

Total _____

Optional:

Where did you hear about this product?

What feedback would you like to give us?

Publisher

Avontra Technical Press
12690 Falcon Drive
Colorado Springs, CO 80908
sales@teachmesun.com

For information on book distributions outside the U.S.A. or to arrange bulk purchase discounts, sales contracts or other promotional/distribution ideas, please contact Avontra Technical Press via email sales@teachmesun.com or postal letter. Please visit this book's official website http://www.teachmesun.com for additional information.

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Avontra Technical Press can not guarantee that every trademark or service mark is perfect. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark. Solaris is a trademark of Sun Solaris. Windows is a registered trademark of Microsoft Corporation. MS-DOS is a registered trademark of Microsoft Corporation.

Warranty of Fitness and Disclaimer

Every effort has been taken to make this book complete and accurate, but no warranty of fitness is implied. The information is provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of any program or CD-ROM(s) shipped with this book or downloaded from the book's official webpage(s).

Intended Audience

This book is intended for junior to mid-level system administrators. Ideally, readers should have access to Solaris 9 and a SPARC-based system. Most lessons can also be performed with Solaris 7 and 8. This book can also be used with HCL compatible Intel-based systems.

Readers who have the Intel or SPARC version of Solaris 7 or Solaris 8 can still perform many of the lessons in this book. However, there are some lessons that specifically deal with new features of Solaris 9 that the reader can not perform with previous versions of Solaris. Please read the next section, "Recommended Solaris Versions and Hardware."

The reader needs to have root access on the test workstation. This workstation should not be used for mission-critical applications or for live production traffic. Try to use a SPARC 5 workstation or a SunBlade 100 workstation. Other higher performance SPARC systems can also be used for the lessons.

Recommended Solaris Versions and Hardware

To perform all the lessons in this book, the reader needs to have access to a SPARC-based system and Solaris 9. Sun Microsystems is planning on releasing an Intel version of Solaris 9 sometime in the year 2003. Please note that the lessons that deal directly with SPARC hardware cannot be performed on an Intel-based system. Most of the lessons in this book can be performed on the SPARC- and Intel-based versions of Solaris 7 and Solaris 8. Some lessons that deal specifically with Solaris 9 features obviously can not be performed with Solaris 7 or Solaris 8.

Solaris 9 can be purchased from http://sunstore.sun.com. New SPARC-based systems can be purchased directly from Sun Microsystems, or used SPARC-based systems can be purchased from online auction sites like http://www.ebay.com or http://auctions.shopping.yahoo.com.

Please Visit Our Website

http://www.teachmesun.com

This website has additional material used in some of the lessons. This website also has additional information about other books and training material in progress. It also has file uploads, promotions, and a calendar of trade shows and professional evens that we are involved with.

- Interested in a book proposal?
- Want to become a reseller of this book?
- Have comments and information about this book? Please report any errors that you may find.
- Are you an expert in Solaris or Cisco®? Would you like to freelance as a reviewer or technical writer?
- Do you have a proposal that might produce a good partnership on a project?

Table of Contents

Other Services We Offer

Solaris Consulting

At Avontra Technical Press, we offer a full range of Solaris consulting, including initial system setup, security monitoring and troubleshooting. We can provide on-site or remote secure connections to your servers. One of our most popular programs is the “second pair of eyes” program. In this program, one of our technicians can visit your site to review your current server configuration and security setup. Please contact us at consulting@teachmesun.com for further information.

Solaris Training

Currently, we offer Solaris Certification classes, as well as custom-tailored classes. The author of this book can come to your location . We plan to have scheduled Solaris classes in Denver Colorado; Colorado Springs, Colorado; and Salt Lake City, Utah. Other cities may be included in the future, depending on the demand. We can provide SPARC workstations or can use your facilities. Please contact us at training@teachmesun.com for further information.

Acknowledgments

Ed Winograd served as Technical Editor and Copy Editor for this book. He has twenty-two years of experience in computers and technology, including seven years writing and editing technical manuals, five years teaching Technical Writing at the University level, and many years of writing and delivering computer training. As a Technical Writer, Ed received the award for Best Technical Training Manual in the Rocky Mountain Regional competition of the Society for Technical Communication.

In the past few years, Ed has edited dozens of manuals and many online help files for clients in the networked storage and telecommunications industries. His most recent book edits include a best-selling book on computer game programming, as well as a book on selling and buying antiques and collectibles in flea markets, antique malls, and eBay. His literary edits include a book of multicultural short stories, Pinch a Lotta Enchiladas, for which he also wrote one of the stories.

As a Technical Editor, Ed has served on the Style Guide committee for the Storage Products Division of a major worldwide supplier of networked storage systems. He has also edited three newsletters in the computing and high tech fields and has written many articles on computing topics.

Ed received his B.A. in English from Colorado College and a graduate degree in English from the University of California, Berkeley. He is a Senior Member of the Society for Technical Communication, a Songwriter Member of ASCAP, and a Member of the Society of Children's Book Writers and Illustrators.

In his spare time, Ed enjoys storytelling and is webmaster and chief content writer for the website of the Rocky Mountain Storytellers' Conference. He lives in Colorado Springs with his wife Gaynelle and their children, Michael and Julie.

Contact Information:

Email: EdWinograd@ureach.com
Address: 220 Silver Spring Drive
Colorado Springs, CO 80919
Phone: 719-528-5448

Royce Williams participated as a contributing author on this book, he did an excellent job writing the LDAP chapter (Chapter 30) Royce Williams is (among other things) a directory administrator for a medium-sized ISP in Anchorage, Alaska. He's wrangled **`sendmail`** and LDAP and battled spammers and on behalf of a 50K user base for over three years on systems ranging from Solaris 2.5.1 to Solaris 9 and FreeBSD. He also has a B.A. in CS from the University of Alaska Anchorage on the ten-year plan. In his "spare" time, he collects Alaskan license plates and is an avid movie buff. He occasionally used to wonder why he wasn't an English major until he started reading the writing of Larry Wall.Most things Royce can be found at http://www.tycho.org.

Conventions Used in This Book

- **Commands:** In the text, the commands that you type are displayed in a monospace bold font. For example: "The **`format`** command is used to partition a disk." All file and directory names are also displayed in a monospace bold font. For example: "the **`/etc/passwd`** file is used to store user information."

- **Arguments and Options:** Some commands require arguments and options to pass information to the command itself. Command arguments and options are enclosed in angle brackets < >. When you see an argument or option inside the < > brackets, type the appropriate value for that argument or option, but not the the brackets themselves. For example: **`lpr –d <default printer>.`**

- **Comments:** Comments are shown in italics. For example: *Always back up this file before editing it.*

Foreword

The Basic Skills section is geared toward novice users. Before using Solaris 9, the reader is introduced to SPARC-based systems. The reader is then guided through a basic installation of Solaris 9, so that the reader's workstation is set up to follow the book's examples. The next couple of chapters cover the Common Desktop Environment, basic file and directory skills, the vi editor, and how to use the different help options available from Sun Microsystems.

The Intermediate Skills section covers typical day-to-day system administration functions. These chapters describe skills such as creating users, performing tape backups and working with software programs. All of these tasks can be performed using only the original Solaris 9 CDROMs and a workstation that is not attached to secondary devices, such as tape drives and modems.

The Advanced Skills section covers how to set up and use the following advanced servers:

DNS (Domain Name Service): Provides hostname to IP address resolution
NIS (Network Information System): Replicates systemwide configuration files
NIS+: An advanced version of NIS
DHCP (Domain Handling Control Protocol): Gives network settings to clients
LDAP (Lightweight Directory Access Protocol): Provides company-wide directory information

Other advanced topics include the Solaris Store Edge Volumes (RAID 0, RAID 1 and RAID 5 volumes) and troubleshooting topics such as analyzing core files and dump files .

One highlight of this book is the LDAP chapter. It features over fifty pages of material and over a dozen examples that illustrate how to set up and use the Sun One Directory Server. Topics in this chapter include how to set up the LDAP server, basic LDAP tree configuration, using LDIF files and Directory Management skills.

Steven Beebe

Chapter 4 Working with Files and Directories

Lessons in This Chapter

Introduction

It is important that a system administrator know how to use command line tools for system administration.. If a server loses its GUI (Graphical User Interface) or does not have a graphics card installed, it is important that a system administrator know how to work with directories and text files. Just ask the simple question, how does a system administrator repair a system that has a problem with the graphical interface like the CDE (Common Desktop Environment), if the GUI interface dies? What if a server never had a GUI interface?

Understand that one of the reasons companies like to use Sun Solaris and other versions of UNIX is because UNIX is designed to be very resistant to hardware failures that would easily crash Microsoft Windows. Part of this “crash resistance” is the fact that Sun Solaris does not need a functional video card or a monitor to work. Most companies do not even connect a monitor to a Sun server. Because of this, a UNIX administrator must be able to work with command line tools. It is extremely unlikely that a company would hire someone to work with Solaris that can not work in a text-only environment.

The first section in this chapter deals with making directories, subdirectories and blank files. In this section, readers will create temporary directories and subdirectories with the **mkdir** command. Next, readers will be shown how to use the **cd** command to move around the directory structure in Solaris. Next, directories will be moved and copied with the **mv** and **cp** commands. Finally, some blank text files will be created with the **touch** command. These files will be copied, moved and deleted within the directory structure.

The second part of this chapter deals with the **vi** text editor. The **vi** text editor is a command line text editor. There is no standard graphical interface to the **vi** text editor. It can be used when there is no GUI display available.

One of the reasons that **vi** is complex is because the standard a-z letters and keyboard symbols are used to give commands to the text editor and add text. The **vi** editor needs to know when a user is writing the characters as text and when characters are being used as a command. For example, the characters **:w** are used as a command to save a file on the hard drive. The only problem is that a user who is writing a program might want to type in an error message that says **Error in :w module**." The **vi** editor needs to know that the ":w" used here is not a command to save a file.

Because of this dual use of text in the **vi** editor, it has two modes of operation, command line mode and insert mode. This counter-intuitive use of both commands and text makes the **vi** editor very difficult for the novice user to master. This chapter is full of easy-to-follow step-by-step examples to explain the **vi** editor.

The last part of this chapter describes some advanced features of the **vi** editor. These include the ability to change words throughout an entire document and the ability to move around very large documents in a hurry.

Working with Directories

In theory, a hard drive could contain only files. Unfortunately, a hard drive with thousands of files and no directories would become a terrible mess. Sun Solaris 9 uses a rather standard System V Release 4 directory structure.

(Just as a quick side comment, this is pronounced "System Five Release Four" rather than "System Vee Release Four." If you do not want to get pegged as a "newbie" you should always say it this way.)

Directories can only hold files and other subdirectories. The **cd** command is used to move around directories. The **cd** command in Solaris is very similar to the **cd** command in MS-DOS or Microsoft Windows. There are some differences, such as that the **cd** command in Solaris uses a front slash (**/**) instead of a backslash (\) for the representation of a directory.

For example, in MS-DOS it's possible to have a directory named **c:\mydirectory**. In Solaris, the same directory would be referenced as **/mydirectory**. Also, understand that Solaris 9 does not use drive letters. The directory **c:\mydirectory** does not exist in Solaris 9. There is no such thing as "c:" to represent a hard drive. Hard drives are accessed with a "mount point" and not a drive letter. Accessing hard drives and the **mount** command will be covered later in the book. For now, just understand that there are no c: or d: or f: or any other drive letters in Solaris 9.

The **pwd** command is used to show what directory a user is in. This directory is known as the "present working directory" in UNIX. If a user makes a file, it will be created in the present working directory.

The command **ls** shows all the files in a directory. A directory can only contain files and subdirectories. A file can be a text file, a binary file or what is known as a "special file." Special files are primarily used by the Solaris operating system for internal use. Special files will be covered later in some of the more advanced topics chapters.

A subdirectory is a directory under a directory. For example, the **/dog/cat/mouse** directory would have **mouse** as a subdirectory of **cat** and **cat** as a subdirectory of **dog**. If a user type the command **cd /dog/cat** that user would be using an "absolute path name." An absolute path name gives the full directory structure, starting with a front slash (**/**). If a user was in the **dog** directory and typed the command **cd cat** that user would be using a "relative path name." A relative path uses only the subdirectory's name. To get to **/dog/cat** a user must first be in the **/dog** directory before typing the command **cd cat.** Otherwise**,** how would Solaris 9 know where the **cat** directory is located? Is the **cat** directory under **/etc/cat** or **/dog/cat** or **/usr/cat**? It is the responsibility of the user to specify what directory is being referenced. An operating system can not play a guessing game.

Lesson 4.1 Basic Directory Commands

In this lesson readers will practice moving around the Solaris 9 directory structure with various commands. The **cd** command is used to move around the directories. The **pwd** command is used to show what directory is the present working directory. The **ls** command is used to show the contents of a directory.

1. Log into the system as user11 (this user was created in Chapter 2).
2. Open a Terminal window
   To open a Terminal window, follow these steps:
   - When the CDE desktop starts, right mouse click anywhere in unoccupied desktop space.
   - Scroll down the Workspace menu until the cursor is over the Tools menu item.
   - Left click on the Tools menu item.
   - Scroll down the Workspace menu until the cursor is over the Terminal icon.
   - Left click on this icon. This should open up a Terminal window.
3. In the Terminal window, type the command **cd**
   *When the **cd** command is typed without any arguments (argument = words behind the command) it takes the user to his or her "home directory." A user's home directory can be thought of as that user's "piece of real-estate" on the server. This is where a user has full permission to create, modify and delete files and directories. The root user has (/) as the home directory. Most users have **/export/home/<username>** as the home directory. The location of the user's home directory depends on how the server was set up. The user11 account has the home directory under **/usr/user11** as setup in Chapter 2.*
4. Type the command **pwd**
   *The **pwd** command is used to show a user where he or she is in the directory structure.*
5. Type the command **cd /etc**
   *The command **cd /etc** changes the current working directory to the **/etc** directory. Any file created will now be created in the **/etc** directory.*
6. Type the command **ls**
   *This **ls** command shows the contents of the current working directory. In this case the current working directory is **/etc** directory.*
7. Type the command **cd /tmp**
   *The command **cd /tmp** changes the current working directory to the **/tmp** directory.*
8. Type the command **pwd**
   *The current working directory is **/tmp** as shown by the **pwd** command.*

9. Type the command **ls**
   *The **ls** command without any arguments shows what is in current working directory. The **/tmp** directory is the current working directory.*
10. Type the command **ls /tmp**
    *As can be seen in this example, the command **ls /tmp** shows what is in the **/tmp** directory. In this case the absolute path name **/tmp** is being used.*
11. Type the command **cd /usr**
    *The command **cd /usr** changes the current working directory to the **/usr** directory.*
12. Type the command **pwd**
    *The current working directory is **/usr** as shown by the **pwd** command.*
13. Type the command **cd bin**
    *The command **cd bin** command makes the current working directory **/usr/bin**. In this case the **cd** command is using a relative path name. The absolute path name is **/usr/bin**. If the working directory is **/usr** then the relative path name **bin** can be used to direct Solaris 9 to change to **/usr/bin**.*
14. Type the command **cd /etc**
    *The command **cd /etc** moved the current working directory to **/etc**.*
15. Type the command **pwd**
    *The command **pwd** shows the present working directory. The current working directory is **/etc**.*
16. Type the command **cd default**
    *The **/etc** directory has a subdirectory **/etc/default**. The command **cd default** used the relative path name **default** to change the current working directory to **/etc/default.***
17. Type the command **cd**
    The command cd by itself returns the user to the home directory.
18. Type the command **cd /**
    The user is sent to the root directory.
19. Type the command **ls -CF**
    *The command **ls -CF** shows a wide listing of all directories and files. The **C** option is used to make the display a wide display. The **F** option puts a **/** character at the end of directories.*

Using the touch Command

The **touch** command can be used to create an empty text file. Imagine opening Microsoft Word and immediately saving a file without any text inside it. An empty text file is the same concept. It is a text file that has no letters and numbers.

The Solaris 9 file system records the last time a file was created, modified or accessed. The **touch** command can also be used to update a file's last modified date.

Lesson 4.2 Using the touch Command

In this lesson the **touch** command will be used to create an empty text file.

1. Log into the system as user11 (*if not logged in as user11 now*).
   *Remember, **user11** was created in chapter 2. The password should be user11.*
2. Open a Terminal window
   To open a Terminal window, follow these steps:
   - When the CDE desktop starts, right mouse click anywhere in unoccupied desktop space.
   - Scroll down the Workspace menu until the cursor is over the Tools menu item.
   - Left click on the Tools menu item.
   - Scroll down the Workspace menu until the cursor is over the Terminal icon.

- Left click on this icon. This should open up a Terminal window.

3. In the Terminal window, type the command **cd**
   *When the **cd** command is typed without an argument, it returns the user to his or her home directory. In this case **user11** is returned to the **/usr/user11** directory.*
4. Type the command **touch myfile**
   *The **touch** command creates an empty text file named **myfile.***
5. Type the command **ls**
   *The command **ls** shows what is in the current directory. The **ls** command shows that the **touch** command created this file in user11's home directory.*

Using the vi text editor

Why should anyone learn the **vi** editor? It's very complex, and there are other "nicer" text editors on the market?

That is a question instructors hear all the time. On a strictly professional level, a UNIX system administrator who does not know how to use the **vi** editor would be looked down on by his or her peers. It is a rather odd "rite of passage" in the UNIX world that a system administrator must know how to use the **vi** text editor. There are other text editors available that are much easier to use but, as unfair as it seems, it is part of the UNIX culture that a system administrator must know how to use it. Also, most companies ask questions about the **vi** text editor during the interview process.

On a practical level, the **vi** text editor is a critical piece of software on a UNIX server. Imagine if a UNIX operating system lost its ability to display a GUI screen and there was not a command line text editor. How would the system administrator change a text file then? Also, terminal sessions and modem connections don't have a GUI interface available. When it is 3:00 A.M. and a server goes down, a system administrator must be able to connect to the server and change text files. Even if no GUI connection is available, he or she can use the **vi** text editor in this case.

To learn **vi**, the author of this book recommends the book *Learning the Vi Editor* by Linda Lamb, from O'Reilly Publishers. Its ISBN number is 0-937175-67-6, and it has a retail price of $24.95. This book is excellent! It has a lot of very good material on the **vi** editor. Readers should read Chapters 1 through 4 for basic **vi** questions. Chapters 5 and above deal with advanced topics that are not that critical for readers who are just learning the **vi** editor.

Some drawbacks to this book are that it does not have enough "hand holding" and easy-to-follow lessons in the very beginning. Also, the book covers a generic version of **vi** that does not seem to be 100% compatible with the **vi** editor in Sun Solaris. But once the basics of the **vi** editor are learned, the book can serve as an excellent training/reference material for readers to continue on by themselves.

Try to become familiar with the basics of **vi** before using some of the more advanced features. This editor is extremely counter-intuitive to new users. Also, when it comes to studying advanced features, understand there are some extraneous commands that are not really needed. For example, the command mode key combination "F *x*" is used to search for the character *x* backward in the current line. Under what circumstances would someone need to use that command? This chapter is not going to describe such oddball commands that a novice user does not need to use at this point in time.

The **vi** text editor is the default text editor installed in Solaris 9. It does not feature any kind of GUI or mouse controls. It can only change text in text files. Most of the file located in the **/etc** directory are text files, such as: **/etc/passwd**, **/etc/groups** and **/etc/shadow** (these three text files define a user). A seasoned Solaris system administrator can edit these files by hand to create a custom-made user, most likely by using the **vi** editor.

The **vi** editor operates in two different modes:

1. Command Mode – **vi** is given commands such as open file, save file, quit. Anytime the **ESC** key is pressed, **vi** goes into command mode.

2. Insert/Append Mode – This is where the user adds text: “My dog likes to chew bones." When **vi** is in command mode, the user can type any of the letters “**i I a A o O**" to go to insert/append mode.

Lesson 4.3 Using the vi Editor

In this lesson readers learn how to use the **vi** editor. The readers create a simple text file with three lines. The text file is then saved and viewed with the **cat** command.

If for some reason a mistake is made, just type the command :q to quit the **vi** editor and then start the lesson again.

1. Log into the system as user11.
2. Open a Terminal window.
3. Type the command **cd**
4. Type the command **vi textfile**
   *This command starts the **vi** editor on a file named **textfile**. By default **vi** starts in command mode. It is expecting the user to perform operations like saving a file, reading a file, moving to a new line, etc. Nothing the user types will be entered as text in the document.*
5. Now type the letter **i**
   The vi editor is now in “ insert mode.“ Now, anything typed will be added to the text.
6. Type in the following lines of text. Press the Return key after each line.

   This is the first line
   This is the second line
   This is the third line

   *If a mistake is made, press the ESC key several times and then type the command **:q** in the **vi** editor. Start the lesson over again from step 4.*
7. If all three line are correct, press the ESC key.
   *The **vi** editor is now back in command mode. Remember, when in command mode, **vi** is performing operations (read a file, write a file, quit, etc). Nothing typed now will be saved as text.*
8. Now type **:wq!**
   *The command **:wq!** tells **vi** to write the file to the disk (**w**) and then quit (**q**). A message should appear on the screen that looks like this:*

   “textfile” 3 lines, 70 characters

   *This is only information about the file created. It gives the name of the file (**textfile**), the number of lines in the file (**3**) and the total number of characters (**70**).*
9. Type the command **cat textfile**
   *The **cat** command can be used to view the contents of a text file. In this case the **cat** command is being used to view the text in the file just created. The screen text should look something like:*

   This is the first line
   This is the second line
   This is the third line

10. Type the command **cp textfile textfile.backkup**
    *The command **cp textfile textfile.backup** makes a copy of **textfile** with the name **textfile.backup**. The file **testfile.backup** will be used in future lessons.*

Lesson 4.4 Vi : Using the h j k l Keys

It is possible to move the cursor with the arrow keys or the **h k l** keys. The reason that the **h j k l** keys are used is that some keyboards and terminals might not properly understand and transmit codes to move the cursor when the arrow keys are pressed. This is very useful if a system administrator comes across a bad combination of keyboard and terminal connection.

1. Type the command **vi textfile**
   *Remember that **vi** starts in command mode, so you will not be entering text.*
2. Try any combination of the different keys **h j k l**
   h – moves the cursor left
   j – moves the cursor down
   k – moves the cursor up
   l – moves the cursor right
3. Press the ESC key several times
   *Anytime the ESC key is pressed, it puts **vi** into command mode.*
4. Type the command **:q!**
   *The command **:q!** tells the **vi** editor to quit **q** immediately **!** and not save any work.*

Lesson 4.5 Vi : Using the .exrc File

The **vi** editor can have options set that control the its behavior. When it starts, it looks for a file named **.exrc** and reads variables from it that affect the way **vi** opens and operates.

1. Type the command **cd**
   *When the **cd** command is typed without any arguments (argument = words behind the command) it takes the user to his or her "home directory." The location of the user's home directory depends on how the user was set up. The **user11** account has the home directory under **/usr/user11** as setup in Chapter 2.*
2. Type the command **vi .exrc**
3. Press the "i" key.
   *This puts **vi** in insert mode.*
4. Type the words **set showmode**
   *This enters the text "set showmode" into the **.exrc** file*
5. Press the ESC key.
   *Anytime the ESC key is pressed, it puts **vi** into command mode.*
6. Type the command **:set all**
   *The command **:set all** shows all the possible variables that can be set with the **vi** text editor.*
7. Type the command **:wq!**
   *The command **:wq!** tells vi to write (**w**) and then quit (**q**).*
8. Type the command **vi textfile**
9. Press the letter **i**
   *Notice the text "***Insert Mode***" in the lower right corner. The **.exrc** file with the line **set showmode** is responsible for the insert messages now seen in the **vi** editor.*
10. Press the ESC key several times
    *Anytime the ESC key is pressed, it puts **vi** into command mode.*

11. Type the command **`:q!`**
    *The command **`:q!`** tells the **`vi`** editor to quit **`q`** immediately and not save any work.*

Now when the **`vi`** text editor is used, it will show on the bottom line when it is in insert mode, append mode or command mode.

Lesson 4.6 Vi : Insert and Append Modes

There are six different insert and append commands within the **`vi`** editor. The insert mode of **`vi`** starts when the user presses the "**`i`**" key, as shown in the first lesson. Table 4.1 shows some other insert/append commands.

Insertion Letter	Type of insertion
I	Insert text after cursor
I	Insert text before cursor
A	Insert text at the start of the line
A	Insert text at the end of the line
O	Open a new line above the current line
O	Open a new line below the current line

Table 4.1 Vi Insert and Append Modes

1. Type the command **`vi textfile`**
   *This command opens the text file named **textfile**.*
2. Move the cursor between the **s** and the **e** in the word "second."
3. Type the letter **`o`** (small letter **o**, as in Oscar)
   This opens a new line below the second line.
4. Now type the line **This is another line in the text file**
   The text file should now look like

   This is the first line
   This is the second line
   This is another line in the text file
   This is the third line

5. Press the **`ESC`** key.
   *Any time the **`ESC`** key is pressed, the **`vi`** editor goes into command mode.*
6. Now, using the arrow keys, put the cursor just in front of the **s** character of the word "second" in the file.
7. Now type the letter **`A`**
   *This puts **`vi`** into Append Mode and moves the cursor to the end of the line.*
8. Now type " **of my text file**"
   The text should now read:

   This is the first line
   This is the second line of my text file
   This is another line in the text file of my text file
   This is the third line

9. Now type the command **`:wq!`**
   *This writes the file and then quits the **`vi`** editor.*
10. Type the command **`cat textfile`**

*The **cat** command displays the contents of the file "**textfile**" on the screen.*

11. Type the command **cp textfile.backup textfile**
    *The **cp** command in this case is being used to copy the text file **textfile.back** over the **textfile** file. This command restores the original file contents to what they were before the lesson.*
12. Type the command **cat textfile**
    *The original contents of the file **textfile** are displayed.*

Lesson 4.7 Vi : The x key

The **x** key is used to delete a single character. It removes the character just before the cursor. If an error is made, the **u** key can be used to "**undo**" it. The **u** command only works immediately after a mistake.

1. Type the command **vi textfile**
   *This command opens the text file named **textfile**.*
2. Move the cursor between the **s** and the **e** in the word "second."
3. Press the **x** key.
   *When the **x** key is pressed, the **e** character disappears. The **x** key is used to delete the next character in front of the cursor.*
4. Press the **u** key
   *When the **u** key is pressed, the **e** character reappears. The **u** key is used to undo whatever step was just performed.*
5. Practice using the **x** and **u** keys.
6. Type the command **:q!**
   *The command **:q!** tells the **vi** editor to quit immediately and not save any work .*
7. Type the command **cp textfile.backup textfile**
   *The **cp** command in this case is being used to copy the text file **textfile.back** over the **textfile** file. This restores the original file contents to what they were before the lesson.*

Lesson 4.8 Vi : Opening Files

In this lesson readers will learn how to open a file in a directory other than their current home directory. Readers will open a file **/etc/release** with the **vi** editor. The **/etc/release** file is a rather harmless file that gives the name and date of the operating system. Do not modify this file.

1. Type the command **vi /etc/release**
   *It is possible to open a file in any directory if the full path name is given for that file. In this example the **/etc/release** file was opened. As mentioned above, do not to modify this file.*
2. Now type the command **:q!**
   *The command **:q!** tells **vi** to not save any changes and quit. Most of the **vi** commands that deal with opening, reading and writing a file require a **(:)**. Some **vi** commands require a **(:)** at the start, and others do not. This is something that has to be learned by experience. At this pointe the rationale behind the colon (:) will not be described.*

Lesson 4.9 Vi : Using the $ ^ and 0 characters

Most modern keyboards have the keys **HOME** and **END** to move quickly to the start or finish of a line of text. Instead of using these special keys, the **vi** editor uses the (**$**) symbol to move to the end of a line and the **(^)** symbol or the zero (**0**) key to move to the beginning of a line. These keys work when **vi** is in command mode.

1. Type the command **vi textfile**
2. Try moving the cursor with the **h j k l** keys, and with the **$** and ^ and **0** keys (**0** = number zero).

   h – moves the cursor left
   j – moves the cursor down
   k – moves the cursor up
   l – moves the cursor right
   $ - moves the cursor to the end of the line
   ^ - moves the cursor to the start of the line
   0 – moves the cursor to the start of the line

3. Type the command **:q!**
   *The command **:q!** tells the vi editor to quit immediately and not save any work .*

Lesson 4.10 Vi : Using the cw Command

The **cw** command changes the current word to a different word. The word selected is the word directly under the cursor. The **cw** command finishes the change when the **Return** key is pressed.

1. Type the command **vi textfile**
2. Move the cursor over the **s** in the word "second."
3. Type the command **cw**
4. Type the word **mouse**
5. Press the **ESC** key.
   The text file should look like this

   This is the first line
   This is the mouse line
   This is the third line

6. Type the command **:q!**
   *This command **:q!** quits **vi** without saving the text.*
7. Type the command **cp textfile.backup textfile**
   *The **cp** command in this case is being used to copy the text file **textfile.back** over the **textfile** file. This command restores the original file contents to what they were before the lesson.*

Lesson 4.11 Vi : Using the dw command

The command **dw** deletes the word directly under the cursor. This is much quicker than pressing the **x** key several times to delete every character of the word letter by letter.

1. Type the command **vi textfile**
2. Move the cursor over the s in the word "second."
3. Type the command **dw**
   The text should now look like this

   This is the first line
   This is the line
   This is the third line

4. Type the command **:wq!**

This command writes the changes then quits ***vi***.

5. Type the command **`cat textfile`**
   Notice the deleted word?
6. Type the command **`cp textfile.backup textfile`**
   The ***`cp`*** *command in this case is being used to copy the text file* ***`textfile.back`*** *over the* ***`textfile`*** *file. This command restores the original file contents to what they were before the lesson.*

Lesson 4. 12 Vi : Using the dd command

To delete an entire line of text from a file, use the command **`dd`**. The **`u`** command can undo this is you make an error.

1. Type the command **`vi textfile`**
2. Move the cursor to the word "second" using the arrow keys or the **`h j k l`** keys until the cursor is in-between the **s** and the **e.**
3. Type the command **`dd`**
   The text should now look like this

   This is the first line
   This is the third line

4. Type the command **`u`** **(to undo the change)**
   The text should now look like this

   This is the first line
   This is the second line
   This is the third line

5. Type the command **`:q!`**
   This command ***`:q!`*** *quits* ***`vi`*** *without saving the work.*
6. Type the command **`cp textfile.backup textfile`**
   The ***`cp`*** *command in this case is being used to copy the text file* ***`textfile.back`*** *over the* ***`textfile`*** *file. This command restores the original file contents to what they were before the lesson.*

Lesson 4.13 Vi : Using the J Command

The **`J`** key is used to join two lines together. This is most often used when the Return key is accidentally pressed.

1. Type the command **`vi jfile`**
2. Press the **i** key
3. Type the following all on one line

   This is the first line. This is the second line.

4. Press the ESC key
5. Place the cursor on the **T** in the second word "**`This`**" as indicated bellow

   This is the first line This is the second line

6. Press the **i** key
7. Press the `Return` key

The text should look like

```
This is the first line
This is the second line
```

8. Press the ESC key
9. Move the cursor back to the first character of the first line.
10. Press the **J** key

The text should now look like

This is the first line This is the second line

11. Type the command **:wq!**

Lesson 4.14 Vi : Using the :r Command

It is possible in **vi** to read the contents of one file into another file. In this example, the contents of the **/etc/release** file (a text file) will be read into the **textfile** file.

1. Type the command **vi textfile**
2. Move to the cursor to the last line and type the letter **A.**
3. Press the Return key twice.
4. Press the ESC key.
   The cursor should now be at the start of a blank line, at the end of the text document.
5. Type the command **:r /etc/release**
   *This command copies the contents of the **/etc/release text** file into your current file.*
6. Press the ESC key.
7. Type the command **:wq**
8. Type the command **cat textfile**
   *The screen should show the original **textfile**, with some extra information about Sun Solaris at the end. The content of the file **/etc/release** was added to the **textfile** file.*
9. Type the command **cp textfile.backup textfile**
   *The **cp** command in this case is being used to copy the text file **textfile.back** over the **textfile** file. This command restores the original file contents to what they were before the lesson.*

Lesson 4.15 Vi : Searching for text

The **vi** editor has a built-in word search capability. In this lesson the pattern search command (**/**) is used to find all occurrences of a word. The **n** key can be used to find the next occurrence of a word. The capital **N** key is used to repeat the search in the reverse order.

1. Type the command **man man > bigtextfile**
   *This command creates a rather large text file named **bigtextfile**, if it doesn't exist already.*
2. Type the command **vi bigtextfile**
3. Type the command **/displays**
   *This tells **vi** to find the word "displays" in the file and place the cursor on the first letter of the word.*
4. Type the letter **n**
   This finds the next occurrence of "displays" in the file.
5. Type the letter **n** several times
   This finds more occurrences of the word “displays” in the file.
6. Type the letter **N** several time

*Using capital **N** searches backward in the file.*

7. Type the command **`?example`**
   This searches backward and lets you specify the search pattern ("example") in the same command.
8. Type the command **`:wq!`**
   *The command **`:wq!`** is used to write and then quit the **`vi`** editor.*

Lesson 4.16 Vi : Using the G Command

With large files, it becomes rather tedious to use the up and down arrow keys to move to a given line. The command **`<line#>G`** moves the cursor to a line very quickly.

1. Use this following command if the file **`bigtextfile`** does not exist:
   `man man > bigtextfile`
   *This command creates a rather large text file named **`bigtextfile.`***
2. Type the command **`vi bigtextfile`**
3. Type the command **`3G`**
4. *This moves the cursor to line 3 of the file. The commands shown below move it to lines 10 and 73.*
5. Type the command **`10G`**
6. Type the command **`73G`**
7. Type the command **`:wq`**
   *This command writes the file and quits the **`vi`** editor.*

Lesson 4.17 Vi : Scrolling Commands

This lesson covers the following commands for scrolling through a file:

Command	Cursor Movement
H	To top of screen
L	To bottom of screen
M	To middle of screen
CTRL + F	One screen forward
CTRL + B	One screen backward
CTRL + D	Half a screen forward
CTRL + U	Half a screen backward

Table 4.2 Vi Scrolling Commands

1. Use this following command if the file **`bigtextfile`** does not exist:
   `man man > bigtextfile`
   *This command creates a rather large text file named **`bigtextfile.`***
2. Type the command **`vi bigtextfile`**
3. Type the command **`L`**
   This command moves the cursor to the bottom of the screen .
4. Type the command **`H`**
   This command moves the cursor to the top of the screen.
5. Type the command **`M`**
   This command moves the cursor to the middle of the screen.
6. Now experiment with the commands **`CTL+F CTL+D CTL+B`** and **`CTL+U`**.
   These commands move the cursor around the screen and scroll the screen up and down.
7. Type the command **`:q!`**
   This command quits the vi editor.

Lesson 4.18 Vi : Using the ! Command

It is possible to run a command within the **vi** text editor. This is more of a time saving feature than anything else. Instead of quitting the vi editor, running a command and then starting **vi** again, the exclamation point (**!**) command breaks out of **vi** and runs a command.

1. Type the command **vi textfile**
2. Now type the command **:!ls /etc**
   *This command displays the contents of the **/etc** directory.*
3. Type the command **:q**
   *This command quits the **vi** editor without saving the work.*

Lesson 4.19 Vi : Global Replacement

The **vi** editor has a command option that will search through a line or an entire text file for a particular word or words. After they are found, they can be replaced. For example, it is possible to tell the **vi** editor to replace the word “dog” with the word “cat” on the current line, or all throughout the entire document.

1. Type the command **man man > bigtextfile**
   *This command creates the large text file **bigtextfile**, if it doesn't exist already.*
2. Type the command **vi bigtextfile**
3. Now type the command **/markup**
   This command finds the next occurrence of "markup"
4. Now type the command **:s/markup/alter**
   *The command **:s/markup/alter** changes the **first** occurrence of "markup" to "alter" in the current line. If "markup" occurred **more than once**, only the **first** occurrence would be changed.*
5. Now type the command **20G**
   *The command **20G** takes the cursor to line 20. If for some reason your bigtextfile does not have text on line 20 (different terminal settings,etc) move to a line with plenty of text.*
6. Now type the command **:s/e/z/g**
   *The command **:s/e/z/g** changes **all** occurrences of the letter **e** to **z** on the current line (because of the **/g**). This does not change all occurrences of the letter **e** in the document, just on the current line.*
7. Now type the command **1G**
   *The command **1G** takes the cursor to line 1.*
8. Now type the command **:%s/e/z/g**
   *This command changes the letter **e** to **z** throughout **the entire document**. The **/g** makes it change all occurrences on a line, and the **%** makes it search the entire document, not just the current line.*
9. Now type the command **:q!**
   This command quits the vi editor and does not write the changes to the file.

Lesson 4.20 Vi : Advanced File Opening Techniques

There are several convenient ways to open the **vi** text editor other than the default edit mode. Rather than opening the file to the first line, it is possible to type the following commands:

vi +<line-number> <filename> starts on a specific line number.
vi /<search_pattern> <filename> opens the file and finds the search pattern.

1. Type the command **man man > bigtextfile**
   *This command creates the large text file **bigtextfile**, if it doesn't exist already.*
2. Type the command **vi +10 bigtextfile**
   *This command opens **bigtextfile** with the cursor on the tenth line.*
3. Type the command **:q!**
4. Type the command **vi +/displays bigtextfile**
   *The command **vi +/displays bigtextfile** opens **bigtextfile** and moves to the first occurrence of the word "displays."*
5. Type the command **:q!**

Advanced Vi Editing

This part of the chapter deals with some advanced features of the **vi** editor. These features can be thought of as "speed techniques." These commands let you delete several characters or lines of text in just a few keystrokes. Understand that the previous material is more than sufficient for day to day UNIX administration. These techniques are only convenient commands for advanced **vi** users.

Lesson 4.21 Vi : Using + and - Commands

In this lesson the readers learn that the + and - keys are very convenient for quickly moving to the first character in the next or previous lines.

1. Type the command **man man > bigtextfile**
2. Type the command **vi bigtextfile**
3. Practice using the **+** and **-** keys
   *Remember to hold down the Shift key with the [+ =] key to send the + character to the terminal. Always use the keys at the top of the keyboard for this. The **+** and **-** keys on the number pad do not work in this example.*
4. Type the command **:q!**

Lesson 4.22 Vi : Marking Text

In this example the user will experiment with marking text and returning to the marked text. The advantage to marking text is that the cursor can be quickly returned to the previous spot. This is a nice feature when it comes to reading through a document. The disadvantage of marking text is that it is so easy to forget where all the marks are.

1. Type the command **man man > bigtextfile**
2. Type the command **vi bigtextfile**
3. Type the command **13G**
   *The command **13G** moves the cursor to the **13**th line.*
4. Type the key combination **ma**
   This creates an invisible mark called "a" at the start of the 13th line.
5. Type the key combination **20G**
   *The command **20G** moves the cursor to the **20**th line.*
6. Type the key combination **mb**
   This creates an invisible mark called "b" at the start of the 20th line.
7. Type the key combination **'a** (an apostrophe or "single quote" followed by an **a**).
   *The command **'a** moves the cursor back to the "a" mark that was created with the **ma** command.*
8. Move to any line in the document, then press the **'b** key combination.
   *The command **'b** moves the cursor back to the "b" mark that was created with the **mb** command.*

9. Type the command **:q!**

Lesson 4.23 Vi : Using the C Command

The **C** key can be used to change the rest of a line.

1. Type the command **man man > bigtextfile**
2. Type the command **vi bigtextfile**
3. Type the command **30G**
   *The command **30G** moves the cursor to the **30th** line.*
4. Move four characters to the right
   *To move four characters to the left, press the **L** key several times, or press the right arrow key several times.*
5. Press the **C** key
   *There should be a dollar sign (**$**) sign at the end of the line. This means that everything between the cursor and the dollar sign (**$**) will be changed to any new text that is typed in.*
6. Type **with some more text**
7. Press the ESC key.
   The remainder of the line is replaced with the text that you typed.
8. Type the command **:q**

Lesson 4. 24 Vi : Using the Tilde (~) Command

The tilde key (**~**) can be used to change the upper/lower case of a letter.

1. Type the command **man man > bigtextfile**
2. Type the command **vi bigtextfile**
3. Move to any letter in the **bigtextfile** file.
4. Press the tilde key (**~**) several times.
   The tilde key toggles a letter between the upper case and the lower case. Hold down the Shift key when pressing the [~ `] key to send the ~ character.
5. Type the command **:q**

Lesson 4.25 Vi : Advanced Delete Techniques

In this lesson the user will try using some advanced delete commands within the **vi** editor. The delete commands can be used to delete several characters, words, or the end of a sentence, with just a few keystrokes.

1. Type the command **man man > bigtextfile**
2. Type the command **vi bigtextfile**
3. Type the command **30G**
   *The command **30G** moves the cursor to the **30th** line.*
4. Type the command **4x**
   *The command **4x** is used to delete the next 4 letters. Whenever a number is in front of a command, the command is repeated that number of times. In this case, **4x** tells the **vi** editor to use the **x** (delete character) command four times.*
5. Press the **u** key
   *The u command tells the **vi** editor to undo the last command. In this case, the **4x** is undone.*
6. Press the **D** key.

7. *Notice that this is capital* D. *The* ***D*** *command deletes the entire line that the cursor is in.*
8. Press the **u** key
   This undoes the last command (undeletes the line).
9. Move the cursor to different lines and practice using the capital **D** and **u** commands.
10. Type the command **d$**
    The command ***d$*** *instructs the delete function to delete to the end of the line symbol ($).*
11. Press the **X** key.
    Notice that this is capital ***X****. The* ***X*** *key is different than the lower case* ***x*** *key in that it deletes the character before the cursor.*
12. Type the command **:q!**

Lesson 4.26 Vi : Using Copy and Paste

In this lesson readers will learn how to copy and paste text in the **vi** editor using the **Y** and **P** keys.

1. Type the command **vi textfile**
2. Create the text lines (if they don't exist)

   This is the first line
   This is the second line
   This is the third line

3. Move the cursor to the first character of the second line.
4. Press the capital **Y** key.
   This copies the entire line.
5. Move the cursor to the first character in the third line.
6. Press the capital **P** key.
   This pastes the copied text into the file. At this point, the file ***textfile*** *should look like:*

   This is the first line
   This is the second line
   This is the second line
   This is the third line

   The second line should now be pasted under the third line.
7. Press the ESC key several times
8. Type the command **:q!**

Examining Text Files

There are several tools that are used to view text files. The **head** command displays the first few lines of a text file. If no options are specified the **head** command shows the top ten lines of a file. A user can specify the number of lines with the options **head -# <filename>**. To view the top ten lines of all files in a directory, type the command **head ***.

The **tail** command displays the last lines of a text file. If no options are specified the **tail** command shows the bottom ten lines of a file. A user can specify the number of lines with the options **head -# <filename>**. To view the bottom ten lines of all files in a directory type the command **tail ***.

The **more** command is used to display text one screen full at a time. The **cat** command is used to display small text files.

The **sed** command lets you look at a range of lines in a file.

Lesson 4.27 Using Text Displaying Commands

This lesson shows reader how to use the **head**, **tail**, **sed, more** and **cat** commands to look at text files on the system.

1. To view the top 10 lines of a text file called **/etc/passwd**, type the command **head /etc/passwd**
2. To view the top 5 lines of **/etc/passwd**, type the command **head -5 /etc/passwd**
3. To view the bottom 5 lines of **/etc/passwd**, type the command **tail -5 /etc/passwd**
4. To view lines 10-15 of **/etc/passwd**, type the command **sed -n '10,15p' /etc/passwd**
   Make sure to use single quotes (') around the 10,15p text. This command is very sensitive to typing errors, be careful when using the command.
5. Type the command **man man > bigtextfile**
6. Type the command **more bigtextfile**
   *The **more** command lets you view a large file without using an editing program.*
7. Press the **Return** key several times. This moves you down one screen at a time.
8. Press the **b** key several times to scroll back up.
9. Press the **space** bar several times to scroll down one line at a time.
10. Press the **q** key to quit the **more** command.
11. To view a small file, (such as **/etc/passwd**) type the command **cat /etc/passwd**
12. To open a file and move directly to the first occurrence of a search pattern, start **vi** with an argument such as **vi +/Invocation bigtext**
    *This opens the **bigtext** file and moves to the first occurrence of "Invocation" in the file.*

Key Points to Remember

This chapter demonstrated how to work with files and directories. It is important that the user spend a great deal of time working with the **cd**, **pwd**, **ls** and **vi** commands. The UNIX file system is very different from the Microsoft Windows file system, so if you are used to Windows, you should make enough practice time to learn the Solaris 9 commands.

Chapter 12 Working with Shell Scripts

Lessons in This Chapter

Introduction

Shell scripts can be thought of as miniature programs that are written with a text editor and kept in text files. Shell scripts work just like batch programs in MS-DOS. There are three primary types of shells that are used with Solaris: the Bourne Shell, The C Shell and the Korn Shell. A shell is a command

interpreter that is a direct interface between the user and the Kernel. With several shells, the user has the ability to choose the type of command interpreter that is used.

The shell's main responsibility is to parse the command line and handle redirection, pipes, job control and wild cards. In layman's terms, the shell watches what the user types and then translates those commands into the binary language that the Kernel can understand. If a series of commands is typed again and again, it is a good idea to put those commands in a shell script. When the script is run, the commands in it are executed. The shell script "types" the commands, instead of the user.

This chapter is going to focus on the Bourne shell script. The Bourne shell is the default shell used by the root user. Almost all of the system shells are written in Bourne. Software developers also like to use Bourne shells with their programs so that the end user can modify the script (unlike compiled software code).

Lesson 12.1 Understanding Shells

In this lesson readers will open and close various shells. The **ps** command will be used to display the currently running shells.

1. Log in as user11 or log in to a non-root account.
   The user account user11 was created in Chapter 2. If this account does not exist on this system, log in as the root user, open a Terminal window and create this user with the following commands:
   ```
   useradd –m –d /usr/user11 user11
   mv /usr/user11/local.profile /usr/user11/.profile
   passwd user11
   ```
2. Open a Terminal window.
   To open a terminal window, right click anywhere in unoccupied desktop space. The Workspace menu will appear. Left click on the Tools menu item, then left click on the Terminal icon.
3. Type the command **ps**
   *The **ps** command shows the active processes. One of the displayed processes should be **sh**. This is the Bourne shell interpreter.*
4. Type the command **sh**
   *The command **sh** starts a new instance of the Bourne shell.*
5. Type the command **ps**
   There should now be two Bourne shells running. The last shell listed is the active shell. If the user quits the active shell, the previous shell will become the active shell. If the user quits all the shells, the terminal window will close.
6. Type the command **exit**
   *The **exit** command takes the user out of the latest opened Bourne shell.*
7. Type the command **ps**
   As can be seen, there is only one Bourne shell running now.
8. Type the command **ksh**
   *The **ksh** command starts the Korn Shell.*
9. Type the command **ps**
   *One of the processes that is running should be **ksh**, the Korn shell.*
10. Type the command **exit**
    *The **exit** command leaves the Korn Shell*
11. Type the command **csh**
    *The **csh** is the C shell interpreter. The command **csh** starts the C shell. The C shell has the prompt* **<Hostname%>**.
12. Type the command **ps**
    *One of the processes that is running should be **csh**, the C-shell interpreter.*

13. Type the command **`exit`**
    *The **`exit`** command leaves the C shell.*
14. Type the command **`exit`**
    *The last **`exit`** should close the last Bourne shell. When that happens, the terminal window dies because the last shell interpreter has been closed.*

Bourne Shell

The Bourne Shell is the shell that is primarily used for administration on the system. Solaris 9 uses Bourne shell scripts in what are known as Run Control Scripts. These Run Control Scripts are located in the **`/etc/rc0.d`**, **`/etc/rc1.d`** , **`/etc/rc2.d`**, **`/etc/rc3.d`** and **`rcS.d`** directories.

When the system starts it executes the Bourne shell scripts located in these directories. These shell scripts start with a capital **S** or a capital **K**. Any shell script that does not start with a capital **S** or a capital **K** will be ignored. The scripts that start with a capital **K** run first. These scripts are used to kill a process or end a program. After all the K scripts (common shorthand lingo for kill scripts or scripts that start with a **K**) have been called, the S scripts are called. The S scripts are used to start a process or a program. Chapter 6 describes the boot process in detail and how the boot process uses Run Control Scripts.

If something goes wrong at startup and a script or a command called by a script hangs, the system can not start properly. The system administrator must be able to modify these shell scripts to fix the system.

As noted above, scripts are text files. A text file can have three types of permission: read, write and execute. These permissions can be seen when the command **`ls -l <filename>`** (that is **`ls -`** little **L** <filename>) is typed. A shell script must be an executable file. If a file does not have the execute permission set, the file will not be run as a script.

The command **`chmod u+x <filename>`** is used to set the execution bit on a file. After the **`chmod`** command is used on a file, the command **`ls -l <filename>`** should show an **`x`** in the file's permissions. The **`x`** indicates that the file is executable.

If a script needs to be run in the Bourne shell, the line **#!/bin/sh** must be the first line of the script.

Quick Tip

If an error message appears saying:

myscript: not found ***myscript = (name of a script)***

this indicates that Solaris does not know the location of the script in the directory structure. If this happens for the script listed above, experiment with the following command structures:

./myscript
or
/directory1/directory2/myscript.
or even
myscript

Find the command structure that make Solaris execute the script. After that, use the command structure that works with your system. Understand that the **PATH** variable can be different for different users. This affects how scripts can be called to execute.

The user's **PATH** variable can have the entry **`:.`** (a colon followed by a period). This lets the user type a command in the current working directory without the need for the **`./`** or **`/directory/directory/script`** command structure. For example, if a user had the following PATH variable:

PATH=/usr/bin:/mydata/directory32:.

the user could change to the script's directory and type **myscript** to make the script execute.

To temporarily change the **PATH** variable, type the commands

```
PATH=$PATH:.
export PATH
```

Have an experienced Solaris system administrator modify your **`.profile`** file in your home directory to make these changes permanent. The **`.profile`** file is a very critical file for a user, and a mistake in this file could cause your system account to become unstable.

Lesson 12.2 Create a Simple Bourne Shell Script

In this lesson readers create a simple shell script that prints the message:

this is my script

on the screen.

In this lesson, a text editor is used to create a shell script. Then, the command **`chmod u+x`** is used to make it executable. Finally, the command **`ls -l`** is used several times to check the file's permissions.

1. Log in as **user11** or as a normal user (if not already logged in).
2. Open a Terminal window.
3. Open the Text Editor.
   To open the Text Editor, right click anywhere in unoccupied desktop space. The Workspace menu will appear at the click point. Left click on the Applications menu choice, then left click on the Text Editor icon.
4. The reader can either use the Text Editor or the **`vi`** editor to create the following script.:
   #!/bin/sh
   echo "this is my script"
5. Save the file with name **`myscript2`**
   *If the reader is unfamiliar with the **`vi`** editor, Chapter 4 describes how to use this program. To save a text file with the text editor, left click on File, then left click on Save As… . Make sure that you save the file to the same directory that the Terminal is in.*
6. Type the command **`ls -l`** *(little **L**)*
   The output of the command should be:
   -rw-r--r-- 1 user11 staff 35 Sep 22 20:54 myscript2
   *The command **`ls -l`** shows the permissions on a file. In this case the first character(**`-`**) indicates the file is a regular file. The next three characters (**`-rw`**) show that the user has read/write permission. The next three characters (**`r--`**) show that others in the same group have read permission, and the last 3 characters (**`r--`**) show that others have read only permission.*
7. Type the command shown below
   `myscript2`
   *This command tells Solaris to execute the script **`myscript2`**. However, at the moment, this command will not work! It is designed to show some typical error messages that Solaris 9 shows with scripts.*

   One error message that you might see is the following:
   ./myscript2: not found
   This indicates that Solaris does not know the location of the script in the directory structure.
   If this happens, e*xperiment with the following command structures:*

   ./myscript2
   or
   /usr/user11/myscript2
   or even
   myscript2

   Find the command structure that make Solaris recognize the script's location.
   Another error message that should appear is something like:
   myscript cannot execute
   The script should not be able to run, because it has not yet been made executable. As shown in step 9, the **`chmod`** *command must be run to make the script executable.*
8. Type the command **`chmod u+x myscript`**
   *The command **`chmod u+x`** sets the execution bit **`x`** on the file **`myscript`** and makes the file executable. The **`u+x`** indicates that the execution bit is being set for the user (**`u`**).*
9. Type the command **`./myscript2`**
   *Now that it has been made into an executable file, the script named **`myscript2`** should work.*
10. Type the command **`ls –l`**
    The output of the command should be:

-rwxr--r-- 1 user11 staff 35 Sep 22 20:54 myscript2

11. *Notice that the **myscript2** file now has an **x** as the fourth character. As this shows, because the file is now executable, the user has execute permission.*
12. Do not close the Terminal window or the Text Editor.
    The Terminal window and Text Editor will be used in some other examples.

The **chmod** command can be used with either numbers (**1-7**) or letters (**r w x**) to set permissions. As shown above, for a text file to be used as a script, its status must be set to executable. You can use any of the methods shown below.

The **chmod** command can use the following numbers to set permissions:

7 - Read/write/execute
6 - Read/write
5 - Read/execute
4 - Read
3 - Write/execute
2 - Write
1 - Execute

By specifying three numbers with the **chmod** command, the permissions for a file can be set for the user, for other people in the user's group and for all other users (in that order). Looking at the list of numbers above, we can see the results of the following commands:

Command	Permissions Set By the Command	Display of Permissions
chmod 744	user rwx group r others r	rwx r-- r--
chmod 317	user -wx group --x others rwx	-wx --x rwx
chmod 215	user -w- group --x others r-x	-w- --x r-x
chmod 777	user rwx group rwx others rwx	rwxrwxrwx

Instead of using numbers, the letters **r w x** and **u g o** can be used to set permissions. With this method, the letters **r**=read, **w**=write, **x**=execute, **u**=user, **g**=group, and **o**=others are used to set permissions.

Here are some examples:

chmod u=x Sets the permissions u = --x g= not affected , o = not affected .The group and others permissions are not changed by this command.

chmod u+x Adds the permission u=x to whatever the permissions were before.

chmod u-x Removes the permission u=x from whatever the permission were before.

chmod u=rw,g=rwx Sets the permissions u= rw- g= rwx o= not affected.

chmod o-rx Removes the read and execute permissions from others, u = not affected, g = not affected.

chmod u=rwx,g=rwx,o=rwx
Sets the permissions u=rwx g=rwx o=rwx on a file

Using Variables

Variables are used to save a value for future use. The easiest variable to use is one that saves characters. With this type of variable, a single character or text message can be saved. A variable is saved with the script line **`<variable>=<value>.`** For example: **`var1=cat`** or **`var2=mice`**.

The Solaris 9 operating system creates some variables automatically. These variables can be seen by typing the command **`env`**. The variables are known as Keyword Shell Variables. They can be changed by a user if needed.

The **`echo`** command can be used to display the contents a variable. The **`echo`** command ordinarily only displays whatever follows it on the screen. To display the contents of a variable, type the name of the variable with a dollar sign in front of it. For example, for a variable called **`myvar`** the command **`echo $myvar`** would display the contents of that variable.

Lesson 12.3 Using Variables in a Script

In this lesson users create some variables and then display those variables on the screen with the **`echo $VARIABLE`** command.

1. Log in as user11 or an ordinary user.
2. Open the Text Editor.
   To open the Text Editor, right click anywhere in unoccupied desktop space. The Workspace menu will appear at the click point. Left click on the Applications menu choice, then left click on the Text Editor icon.
3. Open a Console window.
   In previous lessons the reader was instructed to open a Terminal window. In this lesson we will open a Console window. Understand that both a Console window and a Terminal window can be used to enter text on the screen. To open a Console window, right click anywhere in unoccupied desktop space. The Workspace menu will appear. Left click on the Hosts menu item, then left click on the Terminal Console icon.
4. Type the following three lines into the Text Editor:
   #!/bin/sh
   mydog=spot
   echo "my dog is named $mydog"
5. Save the script with the name **`myscript3`**
   To save a text file with the Text Editor, left click on File, then left click on Save As…
6. In the Console window, type the command **`chmod 777 myscript3`**
   *The command **`chmod 777`** sets rwxrwxrwx permissions on the file.*
7. Type the command **`myscript3`**
   The output of the script should be:
   `my dog is named spot`

   If an error message appears saying:

   myscript3: not found

   this *indicates that Solaris does not know the location of the script in the directory structure. Experiment with the following command structures:*

   ./myscript3
   or
   /usr/user11/myscript3

or even
myscript3

Find the command structure that make Solaris recognize the script's location.

8. Type the command **`env`**
   *The **`env`** command shows the Keyword Shell Variables.*
9. Modify **`myscript3`** so it looks like this:
   #!/bin/sh
   mydog=spot
   echo "my dog is named $mydog"
   echo "the HOME system variable is $HOME"
   echo "the TERM system variable is $TERM"
   echo "the MANPATH system variable is $MANPATH"
   echo "the LANG system variable is $LANG"
   echo "the DISPLAY system variable is $DISPLAY"
   echo "the LOGNAME system variable is $LOGNAME"
   To save a text file with the Text Editor, left click on File, then left click on Save As…
10. Save the file as **`myscript3.2`**.
11. Type the command ./**`myscript3.2`**
    The output of the command should be something like:
    my dog is named spot
    the HOME system variable is /usr/user11
    the TERM system variable is dtterm
    the MANPATH system variable is /usr/bin::/usr/dt/bin:/usr/openwin/bin:bin:usr/ucb
    the LANG system variable is en_US.ISO8859-15
    the DISPLAY system variable is :0.0
    the LOGNAME system variable is user11
    Different systems are set up differently, so your display may be different.

To clear the value of a variable add a line such as "**`myvariable`**=" to the script. This redefines the variable as nothing. The command **`unset`** can also be used to clear a variable. For example, the line **`unset myvariable`** would also clear out the value of the variable.

To save more than one word in a variable, put quotation marks around all the words, as in the second example below.

`myvariable = how are you`	- This command will not work
`myvariable="how are you"`	- This sets the variable to "how are you"

A variable can be made read-only with the command **`readonly <variable name>`**. Read-only variables can not be changed later in the script. In the Bourne shell, a variable is ordinarily local only to the process that created the variable. Another process or program can not use the variable from your script unless you type the command **`export myvariable`**.

Lesson 12.4 Advanced Variables in the Bourne Shell Script

In this lesson readers will work with some of the advanced features of variables in the Bourne shell. A variable will be created with a string of text. The variable will be cleared with the **`unset`** command. Later, the variable will be protected with the **`readonly`** command so that it can not be changed later in the script.

1. Log in as **user11** or an ordinary user.

2. Open the Text Editor.
3. Open a Console window.
4. Create or copy the following lines into a script:
   #!/bin/sh
   myvar1=”how are you”
   echo “The value of myvar1 is - $myvar1 - before the unset command.”
   unset myvar1
   echo “The value of myvar1 is - $myvar1 - after the unset command.”
   myvar1=”new myvar1 value”
   echo “the value of myvar1 is - $myvar1 – as defined again”
   readonly myvar1
   unset myvar1
   echo “the value of myvar1 is now - $myvar1”
5. Save the text file with the name **`myscript4`**
6. Type the command **`chmod 777 myscript4`**
   *The command **`chmod 777`** gives read/write/execute permission to **`myscript4`***
7. Type the command **`./myscript4`**
   *The script will stop with an error, because it attempts to use the **`unset`** command with the readonly variable **myvar1**. You will see the error message:*
   ./myscript4 myvar1: is read only
8. Modify the ninth line of the script to read as follows:
   # unset myvar1
   Notice that the # character has been added at the start of the line. This converts the line to a non-executable line, or comment. When the script is run, this line will not be executed. This means that the Bourne Shell will not try to change the value of myvar1.
9. Execute the script with the command **`./myscript4`**
   *The script should work this time, now that it does not try to change the value of a **readonly** variable.*

Lesson 12.5 Understanding the `export` Command

In this lesson readers will work with two shell scripts. In the first script, the variables **myvar1** and **myvar2** are defined. In **`script1`** the variable **myvar1** is not exported, while the variable **myvar2** is exported. In the second script, only the **`myvar1`** variable is exported. Also in this lesson, readers will learn how one script can call another script

1. Log in as **user11** or an ordinary user.
2. Open the Text Editor.
3. Open a Console window.
4. Create the following script:
   #!/bin/sh
   myvar1="myvar 1 this is not exported"
   myvar2="myvar 2 this is exported"
   export myvar2
   echo "script 1:the value of myvar1 is : $myvar1"
   echo "script 1:the value of myvar2 is : $myvar2"
   ./script2
   *The first line **`#!/bin/sh`** calls the Bourne shell. The next two lines define **`myvar1`** and **`myvar2`** with a text string. The fourth line (**`export myvar2`**) exports the variable **`myvar2`**. The next two lines display the values of **`myvar1`** and **`myvar2`**. The last line (**`./script2`**) executes the second script, which will be created below.*
5. Save this script as **`script1`**
6. Type the command:**`chmod 777 script1`**

The command ***chmod 777*** *gives read/write/execute permission to* ***script1****. (Optional: use the command* ***ls -l*** *to view the permissions).*

7. Create the following script:

   #!/bin/sh
   echo "script 2: the value of myvar1 is $myvar1"
   echo "script 2: the value of myvar2 is $myvar2"

   *The first line of the script (****#!/bin/sh****) calls the Bourne shell. The next two lines display the value of* ***myvar1*** *and* ***myvar2****.*
8. Save this script as **script2**
9. Type the command **chmod u+x script2**
   The command ***chmod u+x*** *makes* ***script2*** *executable. You could also use the command* ***chmod 777 script2***
10. Type the command **./script1**
    Look at the results carefully. Notice that within ***script1****, both* ***myvar1*** *and* ***myvar2*** *have values, but that in* ***script2****, only the exported variable* ***myvar2*** *has a value.*

Reading Values into a Script

Up to this point, all variables have been defined within the script. Now it is time to let the user type in a response to a question and have that response saved in the script. The values are recorded by the **read** command. The **read** command takes text from the keyboard and saves it in a variable. When the user presses the Return key, the value is read into the **read** command.

Lesson 12.6 Reading Values into a Script

This lesson introduces the readers to the read command. Here, the read command is used to record the user's first name in a variable. The variable is then printed with the echo command.

1. Log in as **user11** or as an ordinary user.
2. Open the Text Editor.
3. Open a Console window.
4. Type the following script:
   #!/bin/sh
   echo "what is your name?"
   read myname
   echo "your name is $myname"
5. Save the script as **myscript6**
6. Type a **chmod** command that will make the script executable. Look at previous lessons if necessary.
7. Type the command **./myscript6**
   When ***myscript6*** *runs, it simply asks the user for his or her name and saves this value in the variable named* ***myname****. The last line of the script uses the* ***myname*** *variable to print out the user's name.*
8. To make the script more visually appealing, replace the line
   echo "what is your name?"
   with the line
   echo "what is your name? \c"
   The ***\c*** *option makes the cursor stay on the same line as the prompt, rather than moving to the start of a new line. The user answers the question on the same line where it appears.*

Lesson 12.7 Using the `read` Command to Pause a Script

The **`read`** command can also be used to pause a script until the user presses the Return key. In this example, the **`read`** command is used to read a value into a variable called **ignore**, but the value of that variable is never used. This script also shows the reader how to open a console and how to send a message to the console window.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script in the Text Editor:
   echo "line executed before the read command"
   echo "press the Return key to continue:"
   read ignore
   echo "line executed after the read command"
5. Save the script as **`myscript7`**
6. Type the command **`chmod 777 myscript7`** in the Terminal window.
7. Type the command **`./myscript7`**
   This script should create the following output:

   line executed before the read command
   press the Return key to continue
   after the Return key is pressed
   line executed after the read command

Lesson 12.8 Reading Multiple Values into a Script

The **`read`** command can be used to read in more than one variable at a time. In this lesson the **`read`** command will be used to capture a person's full name (first, middle, last).

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

   #!/bin/sh
   echo "what are your first, middle, and last names? \c"
   read first middle last
   echo "Your first name is $first"
   echo "Your middle name is $middle"
   echo "Your last name is $last"

   Most of the script is easy to understand. The **`\c`** *option makes the cursor stay on the same line as the prompt, rather than moving to the start of a new line. This produces the display:*
   What are your first, middle and last names? *<prompt is still on the same line, which looks better >*
5. Save the script as **`myscript8`**
6. Type the command **`chmod 777 myscript8`**
7. Type the command **`./myscript8`**
   Assume that the user typed "Karen Smith Anderson" at the prompt and then pressed the Return key. The prompt and its answer will appear as:
   What are your first, middle, and last names? Karen Smith Anderson
   The output is :

Your first name is Karen
Your middle name is Smith
Your last name is Anderson

Command Line Arguments

Instead of getting information by prompting a user to respond to a series of questions, some shell scripts use *arguments* that appear after the shell script's name to get information. For example, a script could be created with the name **`mydiskinfo`**. It could be designed to let the user type the command **`mydiskinfo floppy`** to tell the script that he or she wanted information on the floppy drive, or **`mydiskinfo harddrive`** to obtain information on the hard drive.

This script uses **`floppy`** and **`harddrive`** as its arguments. To record the first word typed after the script name, a script uses the reserved variable **`$1`** for its first argument. To record the second word typed after the script name, a script uses the reserved variable **`$2`** for the second argument. The reserved variable **`$0`** is the name of the script itself.

Lesson 12.9 Using Command Line Arguments

This lesson uses a simple script that takes the first argument (a person's name) and sets the **`myname`** variable to the first argument. The echo command is used to display the text

your name is < *your real name* >

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Create and save the following script, named **`myscript9`**

   #!/bin/sh
   myname=$1
   echo "your name is $myname"
5. Type the command **`chmod 777 myscript9`**
6. Run the script, using your name as the first argument. If your name is Karen, type

   `./myscript9 Karen` < *type in your name instead of Karen* >

   The output of the script should be: ***`Your name is Karen`*** < *or your name* >
7. Modify the script so it looks as follows. Then save the script as **`myname2.`**

   #!/bin/sh
   nameofscript=$0
   myname=$1
   mylastname=$2
   echo " Your full name is $myname $mylastname "
   echo " The name of the script is $nameofscript"

   The line ***`mylastname=$2`*** *takes the second argument or* ***`$2`****. The line* ***`nameofscript=$0`*** *takes the "#0" argument, the name of the script.*
8. Type the command **`./myname2 Karen Smith`** < *type in your first and last name instead of Karen Smith* >

   The output of the script should be:

   Your full name is Karen Smith
   The name of the script is myname2
   Saving Output To a Text File

Scenario: A system administrator creates a wonderful script that reports on all the files that have been modified in the last three days. The script is run, and over 500 file names fly past on the screen. Even if it were possible to grab a pencil and paper and write down all the file names by hand, consider the consequences. Say that the system administrator gives such a handwritten list to the team lead. In that case, it might be time for the system administrator to start updating his or her resume!

To solve this problem, the Bourne shell has a function called *redirection*. Instead of having the output go to the screen, the output can be sent into a text file. To use redirection, at the end of the command line, add the (**>**) symbol (which says to redirect the output), and a filename to put the output into. For example, the command **`ls -l > mylisting`** puts the output of the **`ls -l`** command into a file called **`mylisting`**. When output is redirected into a file, it does not appear on the screen.

Technically, redirection works by taking the STDOUT (Standard OUT) that would go to the monitor and redirecting it to a text file. The greater than symbol (**>**) is used to create a text file and to add the STDOUT to the text.

You can also use the symbol (**>>**) to redirect output. In this case, if the file already exists, the output will be *appended* to the end of it. If the file does not exist, it will created, just as if the (**>**) symbol had been used. For example, the command **`ls -l >> mylisting`** appends the output of the **`ls -l`** command to the current contents of an existing file called **`mylisting`**.

You can also use the symbol (**>>**) to redirect output. In this case, if the file already exists, the output will be *appended* to the end of it. If the file does not exist, it will created, just as if the (**>**) symbol had been used. For example, the command **`ls -l >> mylisting`** appends the output of the **`ls -l`** command to the current contents of an existing file called **`mylisting`**.

WARNING

- When text is redirected with the **>** operator into an *existing* file, the file's contents will be permanently *overwritten* by the new text. Do not use the **>** operator on an existing file that you want to keep!
- You can safely use the **>>** operator to append text to the end of an existing file.

There is one other redirection operator. If (**<**) is used with a command, information is taken from a text file, rather than from the keyboard, and is then sent to a command.

Lesson 12.10 Using Redirection

In this lesson readers will use the (>), (<) and (>>) redirection symbols. Remember, the (>) symbol is used to create and add text to a file. The (>>) symbol adds extra text to an existing file without damaging the original text contents. The (<) symbol extracts file from a text file.

1. Log in as **user11** or as an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the command **`date`** *(in the terminal window).*
   The system date should now appear on the screen.
5. Type the command **`date > datefile`**

The system date is now written to a text file named **datefile**

6. Type the command **cat datefile**
   The **cat** *command shows the contents of the* **datefile** *text file.*
7. Create the following script:
   #!/bin/sh
   ls /etc > etcfile
   The command uses the **ls** *command to list all the files in the* **/etc** *directory. The output is redirected to the text file* **etcfile**.
8. Save the script as **myscript10**
9. Type the command **chmod 777 myscript10**
10. Type the command **./myscript10**
    The **myscript10** *file creates a new text file with the name* **etcfile**.
11. Type the command **cat etcfile**
    The **etcfile** *file contains the system date.*
12. Type the command **wc -l < etcfile**
    The **wc** *(word count) command counts the words in its input. Normally, the input comes from the keyboard. Here, the* **<** *operator tells* **wc** *to read the input from* **etcfile** *instead. The screen now shows how many words are in* **etcfile.**
13. Type the command **echo "add line 1" > myfile1**
    The file `myfile1` **is created. It has one line: "add line 1"**
14. Type the command **echo "add line 2" >> myfile1**
    The line "add line 2" is appended to the end of the file **myfile1**.
15. Type the command **cat myfile1**
    As shown onscreen, the **echo** *commands created* `myfile1` *and added a second line to it, using* **>** *and >> for redirection.*

Lesson 12.11 Using Pipes

The standard output from one command can be *piped* directly into the standard input of another command. For example, take the command

ls /etc | grep "^a"

This command take the output of the command **ls /etc** and instead of putting this output on the screen (the screen is STDOUT), it pipes the information directly into the **grep** command. The **grep** command here takes the standard input (STDIN) and searches for all words that start with the letter **a**. The pipe symbol (**|**) is created by pressing the key that is located above the Return key on a PC keyboard. The pipe symbol is above the Back Space key on a Sun keyboard.

1. Login as **user11** or as an ordinary user.
2. Open a Terminal window.
3. Type the command **ls /etc > etcfile**
   The command uses the **ls** *command to list all the files in the* **/etc** *directory. The output is redirected to the text file* **etcfile**.
4. Type the command **cat etcfile | grep "^a"**
   The command **cat etcfile |** *pipes the output of the* **cat etcfile** *command into the* **grep** *command. The* **grep** *command is a pattern matching command. The command* **grep "^a"** *finds all the files and directories in the* **/etc** *directory that start with the letter a.*

Mathematical Expressions in the Bourne Shell Script

It is possible to do some basic mathematics in the Bourne shell script. This includes addition, subtraction, multiplication and division. Unfortunately the Bourne shell can only work with integers, so if the actual answer is something like 2.35 or 4.362, it will be reported (incorrectly) as a whole number such as 2 or 4. If you need to work with math that produces decimal numbers, use a more advanced shell, like the C shell.

Lesson 12.12 Using Math Expressions in the Bourne Shell Script

This lesson will demonstrate how to use simple math operations, such as addition, subtraction, multiplication and division in a shell script.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Create the following script

```
#!/bin/sh
a=10
b=4
addval=`expr $a + $b`
subval=`expr $a - $b`
mulval=`expr $a \* $b`
divval=`expr $a / $b`
echo "the value of $a + $b = $addval"
echo "the value of $a - $b = $subval"
echo "the value of $a * $b = $mulval"
echo "the value of $a / $b = $divval"
```

   *IMPORTANT: Note that the lines that perform math (the ones with the variables **addval**, **subval**, **mulval** and **divval**) use the **grave accent character (`)** around the mathematical expressions. **This is not the single quote character (')**.*

   On a Sun keyboard, the grave accent character is the key just above the Back Space Key. On a PC keyboard, this key is usually above the Tab key on the left side of the keyboard, on the same key as the tilde (~) key.
5. Save the script with the name **`myscript12`**
6. Type the command **`chmod 777 myscript12`**
7. Type the command **`./myscript12`**

   The Bourne shell script numbers should create the following output:

```
the value of 10 + 4 = 14
the value of 10 - 4 = 6
the value of 10 * 4 = 40
echo "the value of 10 / 4 = 2
```

   *Notice that on the last line, **10 / 4 = 2** is not correct. Obviously, it should be 10 / 4=2.5. Unfortunately, the Bourne shell only works with integers. If you want to work with floating point numbers and other detailed mathematics, try using another shell.*

Using Functions in a Bourne Script

The Bourne shell script supports functions. Functions let you create mini-programs within a script. Once a function is established, a script can call that function, rather than having to include all the

operations that the function carries out. This is good because it helps to clean up a shell script. This makes the script easier to read and understand.

Lesson 12.13 Using Functions in a Bourn Shell Script

The reader will create a simple function (adding two numbers) inside a Bourne shell script and will then have the script call that function.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Create the following script:

```
#!/bin/sh
# Start of the add_numbers( ) function
  add_numbers ( )
  {
  echo "value of \$1 is $1"
  echo "value of \$2 is $2"
  c=`expr $1 + $2`
  }
# End of the add_numbers ( ) function
add_numbers 5 1
echo "the value of $1 + $2 = $c "
```

The script works as follows:

a. In the first line, the expression **`#!/bin/sh`** *invokes the Bourne shell script.*

b. The other lines that start with the # character are comments.

c. The lines between the comments are the function **`add_numbers ()`**.

- *The starting curly bracket* **`{`** *marks the start of the function.*
- *The line* **`echo "value of \$1 is $1"`** *displays the value of* **`$1`**
- *The line* **`echo "value of \$2 is $2"`** *displays the value of* **`$2`**.
- *The variables* **`$1`** *and* **`$2`** *in this case take the first and second arguments that are passed to the function when it is run. As shown in the line after the second comment, in this case* **`$1`** = **`5`** *and* **`$2`** = **`1`**
- *The line* **`c=`expr $1 + $2``** *evaluates $1 + $2 and puts the result in the variable* **`c`**
- *The ending curly bracket* **`}`** *marks the end of the function*

d. The line **`add_numbers 5 1`** *calls the function* **`add_numbers`** *with the first argument* **`5`** *and the second argument* **`1`**

e. The final line shows what was added, and the final result.

5. Save the script with the name **`myscript13`**
6. Type the command **`chmod 777 myscript13`**
7. Type the command **`./myscript13`**

   The output should be

   the value of 5 + 1 = 6

Lesson 12.14 Using Variables with Functions

In this lesson, the user creates a slightly more complex script that uses the read command to read in numbers from the command line. These numbers are then added in the add_numbers () subroutine.

1. Login as user11 or an ordinary user

2. Open the Text Editor.
3. Open a Terminal window.
4. Call up **myscript13** and find the line that says **# End of the add_numbers ( ) function**. Delete all the lines after that line, then add the following lines at the end:

```
echo "type in the first number : \c"
read a
echo "type in the second number: \c"
read b
add_numbers $a $b
echo "the value of $a+ $b = $c "
```

The finished script should look like this:

```
#!/bin/sh
# Start of the add_numbers( ) function
add_numbers ()
{
echo "value of \$1 is $1"
echo "value of \$2 is $2"
c=`expr $1 + $2`
}
# End of the add_numbers ( ) function
echo "type in the first number : \c"
read a
echo "type in the second number: \c"
read b
add_numbers $a $b
echo "the value of $a+ $b = $c "
```

5. Save the script with the name **myscript14**
6. Type the command **chmod 777 myscript14**
7. Type the command **./myscript14**

The output of the command should be something like:

```
Type in the first number : 23
Type in the second number: 85
the value of 23 + 85 = 108
```

Lesson 12.15 A Very Complex Set of Scripts

This lesson is used as a review of the previous material. Two scripts are created that, combined with each other, add two numbers together.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Create the following script:

```
#!/bin/sh
echo "type in the first number : \c"
read num1
echo "type in the second number : \c"
read num2
./complex2 $num1 $num2
```

5. Save the script with the name **complex1**
6. Create the following script

```
#!/bin/sh
```

```
a=$1
b=$2
add_numbers ( )
{
num3=`expr $1 + $2`
echo "value of $a + $b = $num3"
}
add_numbers $a $b
```

7. Save the script with the name **complex2**
8. Type the command **chmod 777 complex1**
9. Type the command **chmod 777 complex2**
10. Type the command **./complex1 3 7**

These scripts do the following things:

1. *The **complex1** script reads the two variables **num1** and **num2** when the user types them.*
2. *In the last line of **complex1** these variables are used as the arguments for the **complex2** script: **complex2 num1 num2***
3. *The **complex2** script takes the two arguments and assigns them to the variables **$a** and **$b***
4. *The function **add_numbers** is defined.*
5. *When it is called in the last line of the script, **add_numbers** adds the numbers and displays the output in the statement*
   echo "value of $a and $b = $num3"

Control Statements

Most UNIX administrators are familiar with control statements. The Bourne shell supports the following control statements:

The if control statement

The **if** control statement is used to test a conditional statement. Figure 12.1 and Figure 12.2 show the construction of the **if** statement.

```
if (test expression)
        then
                commands
fi
```

Figure 12.1 The if Control Statement

If the test expression is **true** all statements between the **then** and **fi** are executed. If the test expression is **false**, nothing is done. The statements between the **then** and **fi** are ignored, and program execution continues with the next line.

The **else** statement can be used to specify what happens if the control statement is **false**:

```
if (test expression)
        then
                commands
        else
                commands
        fi
```

Figure 12.2 The `if...then...else` Control Statement

If the test expression is **`true`** all statements between the **`then`** and **`else`** are executed. If the test expression is **`false`** all statements between the **`else`** and **`fi`** are executed. The command **`elseif`** can be used instead of **`else`** to test a new set of expressions.

The test expression is any statement that is either true or false. For example, if a variable **$dog** had the value **Niki** and the **`if`** statement had the test expression (**Nikki = $dog**) the expression would be considered **true** and the commands following **`then`** would be executed. But if the variable **$dog** were changed to **zeppi**, the test expression (**Nikki = $dog**) would be **false** and the commands following the **`else`** would be run, instead.

Valid test expressions are:

Integer Tests

integer1 -eq integer2	integer1 is equal to integer2
integer1 -ne integer2	integer1 is not equal to integer2
integer1 -gt integer2	integer1 is greater than integer2
integer1 -ge integer2	integer1 is greater or equal to integer2
integer1 -lt integer2	integer1 is less than integer2
integer1 -le integer2	integer1 is less than or equal to integer2

Text Tests

text1 = text2	Value of text1 equals text2 (same characters)
text1 != text2	Value of text 1 does not equal text 2 (different characters)
text	The text is not null (a value has been assigned to this variable)
-n text	The length of the string is not zero (variable contains some text)
-z text	The length of the string is zero (variable does not contain any text)

File Tests

-b file	block special file
-c file	character special file
-d file	directory
-f file	regular file
-g file	setGID present
-k file	sticky bit present
-p file	file is a named pipe
-r file	file is readable
-s file	file has some content, is not a blank file
-w file	file is writeable
-x file	file is executable

Lesson 12.16 Using the `if` Command

In this lesson readers will use the **`if`** statement on a test expression. In this case, the test expression is used to determine if a file exists. At first the file does not exist, so the test expression is **`false`** and the statement **`echo "myfile23 exists"`** is not run. Later, the **`touch`** command is used to create a file with the name **`myfile23`**. After the file is created, the **`if`** statement evaluates the test condition as **`true`** and the command **`echo "myfile23 exists"`** is run.

1. Log in as **user11** or as an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type in the following script.
   *Please note that the **`if`** command is very picky about things. Type the script exactly as it appears. This includes the square brackets in the second line, the blank space after the left square bracket, and the blank space before the right square bracket.*

   ```
   #!/bin/sh
   if [ -f myfile23 ]
     then
       echo "myfile23 exists"
   fi
   ```
5. Save the script as **`myscript16`**
6. Type the command **`chmod 777 myscript16`**
7. Type the command **`./myscript16`**
   *When the script runs, no output is produced. This is because the test condition **`[ -f myfile23 ]`** tests to see if a file named **`myfile23`** exists. The file **`myfile23`** does not exist, so the command **`echo "myfile23 exists"`** is never run.*
8. Type the command **`touch myfile23`**
   *The command **`touch <filename>`** creates a blank file. In this case **`touch myfile23`** creates a blank file with the name **`myfile23`**.*
9. Type the command **`./myscript16`**
   *This time, the script should produce the output **`myfile23 exists`**. After **`myfile23`** was created by the **`touch`** command, the test condition **`[ -f myfile ]`** evaluated to **`true`**.*

Using the else option with the if Command

Most users don't like a script that dies without giving some kind of error message or warning message that explains what when wrong. In the previous lesson, when the script did not find the **`myfile23`** file, it ended without giving any type of messages.

To avoid that, the following lesson uses the **`else`** keyword to type an alternate message if the **`if`** test condition evaluates to **`false`**.

Lesson 12.17 Using the `else` Command

In this example the **`else`** command creates an alternate message if the original **`if`** test condition is **`false`**. This script is nicer to users than the last one, because it tells users why this script ended the way it did.

1. Log in as **user11** or as an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.

4. Type in the following script (remember to type it exactly as shown):

```
#!/bin/sh
if [ -f myfile25 ]
  then
    echo "myfile23 exists"
  else
    echo "myfile23 does not exist, ending script"
fi
```

5. Save the script with the name **myscript17**
6. Type the command **chmod 777 myscript17**
7. Type the command **rm myfile23**
   *The command **rm myfile23** deletes the **myfile23** file.*
8. Type the command **./myscript17**
   *The script should display the message **myfile23 does not exist, ending script**. This is because the test condition **[-f myfile23]** is **false**. In this case, the **if** keyword will not run the commands after the word **then**. Instead, the **else** keyword will run the commands that appear after it. In other words, the **else** keyword works when the **if** condition is **false**.*
9. Type the command **touch myfile23**
10. Type the command **./myscript17**
    *This time, the script should produce the output **myfile23 exists**. This happens because **myfile23** was created by the **touch** command. and the test condition **[-f myfile]** is now **true**.*

Creating an if/else if Tree

Sometimes, any of several different conditions could be **true**. When this is the case, it is helpful to use a construction technique with the **if** command that we will call an "**if/else if** tree." By using one **if** statement and multiple **else if** statements, a script can handle multiple possibilities with ease.

```
if [ test condition ]
  then
    command(s)
  else if [ test condition ]
    then
      command(s)
  else if [ test condition ]
    then
      command(s)
    fi
  fi
fi
```

With an **if/else if** tree, a condition can have numerous tests performed on it. No test has any higher precedence than any other. This structure simply gives the script a chance to perform multiple tests on a variable or condition.

Lesson 12.18 Creating an if/else if Tree

In this lesson a simple **if/else if** tree is created that tests the value of a variable. As we will see, the script will continue to look for a matching value if the first if statement is **false**.

1. Log in as **user11** or as an ordinary user

2. Open the Text Editor.
3. Open a Terminal window.
4. Type in the following script:
```
#!/bin/sh
echo "Type in the value of a: \c"
read a
if [ $a -eq 1 ]
  then
    echo " a = 1 "
  else if [ $a -eq 2 ]
    then
      echo " a = 2 "
  else if [ $a -eq 3 ]
    then
    echo " a = 3 "
      fi
    fi
  fi
```
5. Save the script with the name **myscript18**
6. Type the command **chmod 777 myscript18**
7. Type the command **./myscript18**
   *When the script runs, enter **1**, **2**, or **3** as the value for **a**. The script will move down the **if/elseif** tree. When it finds a value that it "knows about" it will print that value. None of the if or elseif checks has greater importance to the program than any of the others.*
8. Type the command **./myscript18**
   *This time, enter the value **4**. No output will be produced. Can you figure out why?*

Other Useful if Test Conditions

This section features a series of lessons that demonstrate some very useful **if** statements.

Lesson 12.19 if Statement Testing a Number For a Range of Values

In this lesson, the script tests a user's guess number for a range of values.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type in the following script:
```
#!/bin/sh
echo "pick a number between 10 - 30 ? \c"
read myvar1
if [ `expr $myvar1` -lt 10 -o `expr $myvar1` -gt 30 ]
  then
    echo " Number not chosen between 10 - 30 "
  else
    echo " Number was between 10 - 30 "
fi
```
5. Save the script with the name **myscript19**
6. Type the command **chmod 777 myscript19**
7. Type the command **./myscript19**
   *The test condition uses the **expr** command to find the numeric value of the variable **myvar1**. The test conditions are **-lt** (less than), **-gt** (greater than) and **-o** (or).*

Lesson 12.20 The Yes Tester

In this lesson, an answer is tested to see if it is **y** or **Y** answer.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type in the following script:

```
#!/bin/sh
echo "are you sure (y or Y)? \c"
read myvar1
if [ "$myvar1" = Y -o "$myvar1" = y ]
   then
   echo " The user typed yes "
fi
```

5. Save the script with the name **`myscript20`**
6. Type the command **`chmod 777 myscript20`**
7. Type the command **`./myscript20`**

The only complex part of this script that should be explained is the test condition. The variable ***`$myvar1`*** *is enclosed in double quote marks (") so that its alphanumeric value can be compared against* ***y*** *or* ***Y****. The* ***`-o`*** *option means "or". Notice that the script produces output only if the user types* ***y*** *or* ***Y*** *because there is no* ***`else`*** *or* ***`else if`*** *condition to tell it what to do otherwise.*

The case Command

The **`case`** command works just like the **`if/else if`** tree (but is more convenient to use). It checks a variable for various conditions. When a condition is met, the statements that go with it are executed. Each tested condition begins with the condition and ends with two semicolons (**;;**).

The **`case`** command uses the asterisk character (*****) as a catch-all for any other value than the tested values. The keyword **`esac`** (case spelled backward) is used to end the **`case`** command's structure.

```
case variable in
        value1)
                statements
        ;;
        value2)
                statements
        ;;
        *)
                statements
        ;;
esac
```

Lesson 12.21 Using the `case` Command

In this lesson the **`case`** command is used to ask about a user's car.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.

4. Type the following script:

```
#!/bin/sh
echo "who makes your car? \c"
read mycar
case "$mycar" in
Ford)
        echo "An American car"
        echo "Made in Detroit"
        ;;
Mercedes)
        echo "Yeah, sure you own one!"
        ;;
Hyundai)
        echo "An Asian car"
        echo "Made in Korea"
        ;;
*)
        echo "Never heard of that car"
esac
```

5. Save the script with the name **myscript21**
6. Type the command **chmod 777 myscript21**
7. Type the command **./myscript21**
   *Type in **Ford, Mercedes, Hyundai** or any other car company's name. Notice that any car other than the ones tested for gets the response "Never heard of that car."*

Lesson 12.22 Loosening Up the `case` Command

People have the bad habit of making typing mistakes. Instead of typing the word Ford, some people might type Foord or Fod or ford (all lower case) instead. To allow for this type of variation, the `case` command supports what are know as *regular expressions*. Most people know that the asterisk (*) represents any character or string of characters. As shown below, there are other regular expressions that can be used with the **case** command.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
echo "who makes your car? \c"
read mycar
case "$mycar" in
[Ff]o*)
        echo "An American car"
        echo "Made in Detroit"
        ;;
[Mm]erced*)
        echo "Yeah, sure you own one"
        ;;
[Hh]yun???)
        echo "An Asian car"
        echo "Made in Korea"
        ;;
*)
```

echo "Never heard of that car"
esac

5. Save the script with the name **myscript22**
6. Type the command **chmod 777 myscript22**
7. Type the command **./myscript22**

*The **case** statement uses several types of "wild cards" when doing comparisons:*

- *The asterisk (*****) matches any character or string of characters. It also matches "nothing"—if no characters are typed at the position where it occurs.*
- *The question mark (**?**) matches any single character.*
- *Enclosing more than one character in square brackets (**[ ]**) tells the **case** statement to accept any of the characters between the square brackets*

By using these wild cards, this script accepts a wider variety of input:

- *The **Ford** selection from the previous script was replaced with **[Ff]o***. This lets the user type **f** or **F** as the first letter. The second letter has to be an **o**. The asterisk means that any character or characters will be accepted from the third character onward (or that the user can type nothing here, leaving the input at two characters).*
- *The **Mercedes** selection was replaced with **[Mm]erced***. This lets the user type in **M** or **m** as the first character. The next five characters must be **erced** (lower case). After that, any character or string of characters will be accepted, or the user can type nothing here.*
- *The **Hyundai** selection was replaced with **[Hh]yun???**. This lets the user type **H** or **h** as the first character. The next three characters must be **yun** (lower case). The three question marks (**?**) at the end of the script let the user type in any combination of **exactly** three characters.*

*The wild cards in this script are not intended to be "perfect." For example, notice that for the **Ford** selection, **folderol** would be accepted, and that for the **Mercedes** choice, **merced** would be accepted, but **MERCEDES** would not. Experiment with different combinations to make the case command accept and incorrectly typed answers in the way that you want it to.*

Creating a Selection Menu with the Case Command

The easiest way to make a menu in a Bourne shell script is with the **case** command. Try not to make a menu that is too long for terminal connections. Limit the size of the menu to about 20 lines down to make sure even old terminals can use the menu.

Lesson 12.23 Creating a Menu in the Bourne Shell Script

In this lesson, an onscreen menu is created to guide the user.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:
   #!/bin/sh
   echo "select your vehicle "

```
cat << MENUEND
      1) Ford
      2) Mercedes
      3) Hyundai
MENUEND

read mycar

case "$mycar" in
          1)
          echo "An American car"
          echo "Made in Detroit"
          ;;
          2)
          echo "Yeah, sure you own one!"
          ;;
          3)
          echo "An Asian car"
          echo "Made in Korea"
          ;;
          *)
          echo "Never heard of that car"
esac
```

5. Save the script with the name **`myscript23`**
6. Type the command **`chmod 777 myscript23`**
7. Type the command **`./myscript23`**

This script uses the text between **cat << MENUEND** *and* **MENUEND** *to present a menu. By requiring the reader to enter a number rather than typing an answer, this script avoids problems with incorrect input. This script checks for the numbers **1**, **2** or **3**, rather than checking for the names **Ford**, **Mercedes** or **Hyundai**.*

The for Command

The **`for`** command is good for processing lists of names or lists of anything. The **`for`** command follows the following syntax:

```
for variable in <word list>
        do
                statements
        done
```

Lesson 12.24 Using the `for` command

In this lesson the **`for`** command is used to read in a list of animals and to use that list to populate a series of echo commands.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
for animals in dogs cats mice
```

do
echo "$animals have four legs"
done

5. Save the script with the name **myscript24**
6. Type the command **chmod 777 myscript24**
7. Type the command **./myscript24**

The following output will appear:

dogs have four legs
cats have four legs
mice have four legs

This example only echoes text to the screen. Of course, any type of operation can be put in the loop. For example, the **for** command is good for situations when a system administrator needs to perform the same repetitive task on some files and does not want to type in the same values again and again.

Lesson 12.25 Using the for Command With a Repetitive Command

In this lesson, the **for** command is used to update a text files. One unique feature of the for command is it's unique ability to take words from a command run within the grave accent (`) characters. Remember that the grave accent character (`) runs the command. In the example, the **cat** command is run to change the list of names into a word list that the **for** command needs.

On a Sun keyboard, the grave accent character is the key just above the Back Space Key. On a PC keyboard, this key is usually above the Tab key on the left side of the keyboard, on the same key as the tilde (~) key.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Create the following text file

Michael
Karen
Tom
Steven

5. Save the file as **namelists**
6. Create the following text file

This is the system administrator.
I have put this text file in your home
directory to let you know that you
are using too much disk space

7. Save the file as **warnings**
8. Create the following shell script:

```
#!/bin/sh
for a in `cat namelists`
  do
    cp warnings warningt
```

```
  echo "please delete some files $a" >> warningt
  cp warningt diskwarning.$a
 done
rm warningt
```

9. Save the script as **myscript25**
10. Type the command **chmod 777 myscript25**
11. Type the command **./myscript25**

*Each time through the **for** loop, this script reads a name from **namelists**. It then creates a file called **diskwarning.<user's name>**. When it finishes, the files **diskwarning.Michael**, **diskwarning.Karen**, **diskwarning.Tom** and **diskwarning.Karen** have been created.*

The files are saved in the same directory as the script. The script could have had the line **cp warningt /usr/$a/diskwarning.$a** *to place the warning message in the user's home directory.*

*Here is the text of **diskwarning.Karen:***

This is the system administrator.
I have put this text file in your home
directory to let you know that you
are using too much disk space

please delete some files Karen

*(The file **warningt** is used as a temporary file so that each user's message file will not have the previous user's name in it.)*

12. To remove the warning messages, type the command **rm diskwarnin***

*This example was created to illustrate how to use the **for** command for administration. In a real-world situation, the system administrator would make the warning messages appear on the users' screens or would send the warnings as messages.*

Using the while command

The **while** Control-Flow command tests a condition. The condition is checked at the start. As long as the condition is **true,** the loop between the **do** and the **done** statements will continue to run. When the test condition becomes **false**, the loop terminates and the script continues with the next line. If the condition is **false** when the **while** statement is first encountered, the loop is not executed at all.

In a nutshell:

- The **while** tests a condition.
- If the condition is initially **true**, the loop runs until the test condition becomes **false**.
- If the test condition is initially **false**, the loop does not run all.

The **while** command has the following syntax:

```
while (test condition)
do
      statements
done
```

Lesson 12.26 Using the `while` Command

In this example, the **`while`** command is used to create a loop in which the script asks the user to type in a letter. The letter is then displayed. When the letter **z** is typed, the loop ends, and the script prints a message before terminating.

1. Log in as **user11** or an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
echo "Type in a guess letter \c"
read myvar
while [ "$myvar" != "z" ]
do
  echo " you typed $myvar "
  read myvar
done
echo "That was the guess letter !”
```

5. Save the script with the name **`myscript26`**
6. Type the command **`chmod 777 myscript26`**
7. Type the command **`./myscript26`**

This script is very easy to understand. Within the ***while*** *loop, it simply asks the user to type in a letter (and press the Return key!). It checks to see if the letter is not* **z** (**!= "z"**). *If the user types anything other than* **z** *(lower case), the loop tells the reader what character was typed.*

Notice that the line **echo "Type in a guess letter \c"** is not inside the loop. Because of this, the script only prompts the user to enter a character the first time. After that, the user must remember to type in a new character, without being prompted.

Now modify the script so it looks like this

```
#!/bin/sh
while [ "$myvar" != "z" ]
do
  echo "type in a guess letter \c"
  read myvar
  if [ -n "$myvar" ]
then
  echo " you typed $myvar "
fi
done
echo "That was the guess letter !
```

8. Save the script with the name **`myscript26.2`**
9. Type the command **`chmod 777 wyscript26.2`**
10. Type the command **`./myscript26.2`**

This script improves on the previous one in a couple of ways:

- *Because the* ***echo "type in a guess letter \c"*** *statement is inside the loop, it prompts the user to enter a character each time through the loop.*

- *This script uses the **if** statement nested inside the **while** statement for some sanity checking. If the user presses the **Return** key without typing in a letter the statement "**you typed <letter>**" is not displayed. This happens because the statement **if [-n "$myvar"]** means "if the value of **$myvar** is not blank."*

  This shows that control statements can be nested within each other.

Using the while Command for Numerical Loops

The **while** command can be used when a list of numbers needs to be generated. Remember the while conditional test stops when the loop comes back up to the top and the condition is false.

Lesson 12.27 Using the while Command to Creating a List of Numbers

In this lesson the while command creates a list of numbers from 5 to 15.

1. Login as user11 or an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
numvar=4
while [ $numvar != 15 ]
  do
  numvar=`expr $numvar + 1`
  echo " The number is $numvar"
done
```

5. Save the script with the name **myscript27**
6. Type the command **chmod 777 myscript27**
7. Type the command **./myscript27**

   *The results are pretty obvious. This script prints the values of **numvar** from 5 to 15:*

```
The number is 5
The number is 6
    .
The number is 15
```

*When the **while** loop starts, the value of **numvar** is 4. Each time through, the loop increases the value of **numvar** by 1 and prints its value. When **numvar** reaches 15, the test condition **$numvar != 15** is false, and the loop stops.*

The Until Command

The **until** test condition is the exact opposite of the **while** test condition. The **until** Control-Flow command tests at the start of the loop to see if its condition is **false.** If the condition is **false**, the script executes the loop between the **do** and **done** statements until its test condition becomes **true**. At this point, the script continues with the next statement after the **done** statement. If the condition is **true** when the **until** statement is first encountered, the loop will not be executed.

In a nutshell:

- The **until** command tests a condition.

- If the test condition is initially **false**, the loop runs until the test condition becomes **true**.
- If the test condition is initially **true**, the loop does not run all.

The **`until`** Control-Flow command supports the following format:

```
unitl (test condition)
                  do
                        statements
                  done
```

Lesson 12.28 Using the `until` Command to Generate Numbers

The **`until`** command is used in this example for a simple loop demonstration. The number 5 is given to a variable. The script runs until the variable reaches 16.

1. Log in as **user11** or as an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
numvar=5
until [ $numvar != 16 ]
  do
    echo " The number is $numvar"
    numvar=`expr $numvar + 1`
  done
```

5. Save the script with the name **myscript28**
6. Type the command **`chmod 777 myscript28`**
7. Type the command **./myscript28**
   *This script produces no output! Why? Because the test condition is initially **true**. When the **until** statement is executed for the first time, the value of **numvar** is 5. The statement **$numvar != 16** is **true** here, so the loop will not run!*
8. Type in the following script

```
#!/bin/sh
numvar=5
until [ $numvar -gt 15 ]
do
  echo " The number is $numvar"
   numvar=`expr $numvar + 1`
done
```

9. Save the script with the name **myscript28.2**
10. Type the command **`chmod 777 myscript28.2`**
11. Type the command **./myscript28.2**

*In this script, the value of **numvar** is 5 the first time through the loop. The script prints **`The number is 5`** the first time. It then continues to print the number until it prints **`The number is 15`**.*

*When the **`numvar`** variable reaches 16 at the bottom of the loop, the **`until`** command goes to the top of the loop and checks the condition again. Now, **$numvar -gt 15** is **true**, and the loop stops. The **`until`** command always evaluates the test condition at the top of the loop.*

The `break` command

The **`break`** command is used to break out of a **`for`**, **`while`** or **`until`** loop, regardless of the test condition. The script then continues to run after the **`done`** keyword. The **`exit`** command is used to leave a script immediately. Don't confuse the **`exit`** command with the **`break`** command.

Lesson 12.29 Using the `break` Command

In this lesson the loop will run continuously until the **`break`** command is reached. The break command is inside of an **`if`** statement that checks the variable **goodbye** for the answer **y**. If the user types **y** the **`break`** command is executed, and the loop is terminated immediately, without testing its test condition.

1. Log in as **user11** or as an ordinary user
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
numvar=5
while [ $numvar != 16 ]
 do
  echo " Type y and press Return to leave the loop, any other key to stay in "
  read mykeystroke
   if [ "$mykeystroke" = "y" ]
    then
     echo "break out of the loop"
     break
    fi
  echo "loop running again"
 done
```

5. Save the script with the name **`myscript29`**
6. Type the command **`chmod 777 myscript29`**
7. Type the command **`./myscript29`**

*This script breaks out of the loop when the lower case **y** key is pressed.*

The `continue` command

The **`continue`** command returns to the top of the loop, ignoring anything that is below the **`continue`** command. This can be used on **`for`**, **`while`** and **`until`** loops.

Lesson 12.30 Using the `continue` Command

In this lesson the loop will run continuously until the break command is reached. The break command is inside an **`if`** statement that checks the variable goodbye for the answer "**`y`**".

1. Log in as **user11** or an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
numvar=5
while [ $numvar != 16 ]
```

```
do
 echo "top of the loop"
 echo " Type any letter except - y to leave - c continue command "
 echo "press Return after letter"
 read mykeystroke
  if [ "$mykeystroke" = "y" ]
    then
     echo "break out of the loop"
     break
    fi
    if [ "$mykeystroke" = "c" ]
    then
     echo "continue to top of loop"
     continue
    fi
 echo "bottom of the loop "
done
```

5. Save the script with the name **myscript30**
6. Type the command **chmod 777 myscript30**
7. Type the command **./myscript30**
   *When the **c** key is pressed, the script immediately returns to the top of the loop, without finishing the rest of the loop, because of the **continue** statement Also, this script breaks out of the **while** loop when the **y** key is pressed. The **bottom of the loop** statement does not appear at that point.*

Debugging Bourne Shell Scripts

Sometimes Bourne shell scripts can become very complex. When a Bourne shell script becomes complex, there are some limited debugging utilities that can be used.

set –x A line used in the script to start the debugging

set +x A line used in the script to end the debugging

Lesson 12.31 Using sh -v and set -x and set +x

In this lesson the **set -x** and **set +x** commands are used to check the **myscript30** script as it is run.

1. Log in as **user11** or as an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the command **cp myscript30 myscript31**
   This command copies myscript30 to myscript31.
5. Type the command **./myscript31**
   *The **myscript31** script will run. Notice that each line of the script is displayed as it is being run.*
6. Modify the **myscript31** script so it looks like this:

```
#!/bin/sh
set -x
numvar=5
while [ $numvar != 16 ]
 do
```

```
    echo " The number is $numvar"
     numvar=`expr $numvar + 1`
  done
set +x
myvar327=83
```

7. Type the command **./myscript31**

   *When the script runs, the results of each line between the **set -x** and **set + x** statements are shown as that line is executed. Here is what appears near the end of the loop:*

```
numvar=15
+ [ 15 != 16 ]
+ echo  The number is 15
 The number is 15
+ expr 15 + 1
numvar=16
+ [ 16 != 16 ]
+ set +x
```

   *Any time a script has problems just add the line **set -x** just above the suspected problem area. The script will be displayed line for line as it is being processed. The traced lines are shown with a plus (+) sign.*

   *Notice that the statement **myvar327=83** is not displayed as it is executed. This is because it is after the **set + x** statement. This line is in the script just to show this. Setting the variable **myvar327** has no real purpose..*

Lesson 12.32 Create a Custom Utility

Most system administrators have their personal "bag of tricks" when it comes to custom-made commands. This lesson shows how to create a custom-made command to automatically list directories with the **ls -l** command. The command should be put in the **/usr/bin** directory, because all users have this in the **PATH** variable. Any command in the **PATH** variable's list will automatically be executed when typed at a terminal or console.

1. Log in as the root user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:
   #!/bin/sh
   ls -l
   *Notice that the last line is **ls-l** (small **L**).*
5. Save the script with the name **longlist**
   *The **longlist** command displays a long listing of files and directories with the **ls -l** (small L) command.*
6. Type the command **chmod 555 longlist**
7. *The **chmod 555** sets read/execute permissions on the shell script **longlist***
8. Type the command **cp longlist /usr/bin/**
   *The **/usr/bin** directory is a directory that all the users have in their **PATH** variable. The command **longlist** will be available for all the users when they type the command **longlist** in a terminal.*
9. Log off as root.
10. Log in as user11.

11. Type the command **longlist**
    *The command **longlist** now acts the same as the **ls -l** command. It is available because the shell script **longlist** is in the **/usr/bin** directory. If all users need to use a command, put the command in the **/usr/bin** directory.*

Bourne Run Control Scripts

Bourne shell scripts are used all throughout the Solaris 9 operating system. When Solaris 9 starts, it executes what are known as Run Control Scripts. These scripts perform all of the housekeeping functions after the **init** process is started. For example, these scripts start the Apache web server, the CDE desktop, and start the **sendmail** daemon.

These scripts are located in the following directories:
/etc/rc0.d, /etc/rc1.d, /etc/rc2.d /etc/rc3.d rcS.d

Run Control Script names start with an **S** or a **K**. The scripts that start with an **S** are used to start something. For example, in the **/etc/rc3.d** directory are the scripts **S90samba** and **S50apache**. The **S90samba** script is used to start the Samba daemon **smbd** to allow the Samba service to run. The **S50apache** script is used to start the Apache web server. The script **/etc/rc1.d/K16apache** is used to kill the Apache web server.

Lesson 12.33 Create a Run Control Script

In this lesson, readers create a relatively harmless script that displays the message "my shell script from Chapter 12" when the system starts. The shell script is tested before the server reboots, so that it will not hang during the boot process. This could prevent the server from working.

1. Log in as the root user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Create the script
   echo "my shell script from Chapter 12"
   sleep 5
5. Save the file as **/etc/rc3.d/S25myscript**
6. Type the command **chmod 777 /etc/rc3.d/S25myscript**
7. Type the command **/etc/rc3.d/S25myscript**

   MAKE SURE THE SCRIPT IS CORRECT !!! IT SHOULD ONLY ECHO THE FIRST LINE OF THE SCRIPT ON THE TERMINAL, AND THEN SLEEP FOR 5 SECONDS BEFORE THE SYSTEM PROMPT REAPPEARS.

   SOLARIS 9 COULD HAVE PROBLEMS STARTING IF YOU MAKE A MISTAKE !!!

   Before a server is rebooted or shut down, make sure that any new run control scripts work. If the run control script hangs, the server will not start properly.
8. Type the command **init 6**
   Watch the startup messages carefully. The message **my shell script from Chapter 12** *should appear.*
9. Log in as a root user.
10. Open a Terminal window.
11. Type the command **rm /etc/rc3.d/S25myscript**

The command ***rm /etc/rc3.d/S25myscript*** *removes the* ***S25myscript*** *from the system. If you forget to remove this script, the* **my shell script from Chapter 12** message will appear each time you boot the system.

Users .profile File

Users of the Bourne shell use the **.profile** file to set their own environmental variables. This script can not be seen with the **ls** command because it is a hidden file. Any file that starts with a period (**.**) is a hidden file that will not be seen by the **ls** command. However, the **ls -A** command will show all the files in a directory, even hidden files.

Lesson 12.34 Working With the .profile File

The **.profile** file is used to set a user's environmental variables. In this lesson readers will set the variable **DOG** to the name of the author's dog, **zeppiman**. This variable will be available to the user after the user logs off the system and then logs back on.

1. Log in as **user11**.
2. Open a Terminal window.
3. Open the Text Editor.
4. Edit the file **/usr/user11/.profile**
   Add the following lines to any lines that already exist in the file:
   DOG=zeppiman
   export DOG
5. Save the file as **.profile**
6. Type the command
   . ./.profile
   This command is "dot-space-dot-front slash-dot" followed by "profile"
7. Type the command **echo $DOG**
   The variable ***DOG*** *should be set to* ***zeppiman****.*
8. Log off the system
9. Log back into the system as **user11**.
10. Open a Terminal window.
11. Type the command **echo $DOG**
    The variable ***DOG*** *should still be set to* ***zeppiman***

Lesson 12.35 Capturing a Key

In the previous lessons the user had to press the Return key to enter in a value. To make a more impressive shell script, create the following script, which lets users enter values without pressing Return.

Please note that there might be some problems if a user connects through a terminal session and the user's terminal is not fully recognized by Solaris 9. This script was tested from the keyboard of a SunBlade100 and in a telnet session from Microsoft Windows.

1. Log in as **user11** or an ordinary user.
2. Open the Text Editor.
3. Open a Terminal window.
4. Type the following script:

```
#!/bin/sh
CatchKey ( ) {
```

```
trap "" 2 3
oldSttySettings=`stty -g` # use grave accent marks not single quotes
stty -echo raw
echo "`dd count=1 2> /dev/null`" # grave accent marks here, not single quotes
stty $oldSttySettings
trap 2 3
}

PressKey=""
while [ "$PressKey" != "x" ]
do
echo "press the x key to exit \c"
PressKey=`CatchKey` # these are grave accent marks not single quotes
echo "You just typed $PressKey"
#
# set a variable to anything needed
#
myvar1=$PressKey
echo "myvar1 is set to $myvar1"

# Add your script material here
# Include the case command to make a menu

done
```

5. Save the script with the name **myscript35**
6. Type the command **chmod 777 myscript35**
7. Type the command **./myscript35**

The **CatchKey( )** function uses the **trap** command and the **stty** command to capture the keystrokes from the keyboard. The **PressKey** variable retrieves the value of the function **CatchKey ()**. Because the keys need to be re-read, the **CatchKey ( )** function needs to be within a **while** loop. The value inside the **$PressKey** variable can be used for any typical duties of a variable. For example, copy the contents of the **PressKey** variable into other variables. Use the secondary variables for nested commands, such as **if**, **while** and **until**.

Key Points to Remember

The Bourne shell is the default system administration shell available on Solaris 9. All of the run control scripts and most software scripts are written in the Bourne shell. It will take time and patience to learn how to make nice Bourne Shell scripts. Follow the examples in this chapter. Later, when you become more comfortable with Bourne shells, modify the scripts you created from this chapter to fit your needs. Make tiny changes to the scripts and test the scripts often. Do not make a huge, complex script and then try to debug the script. That is the worst way to do things.

Chapter 31 DHCP

Lessons in This Chapter

Introduction

One problem with a large network is that every system on a standard IPv4 network (such as network printers, Microsoft Windows 2000 workstations, Windows 98 workstations and Solaris workstations) needs to be configured with three key pieces of information to work on a network. Each system needs to have an IP address, a subnet mask and the IP address of the default router. If the network topology changes, these pieces of information need to be changed on all the systems affected by the change. Unfortunately it is a time consuming process to change all this information on each system. Every network device also requires that staff know how to safely change the network settings without accidentally disabling the network services on the device.

DHCP (Domain Handling Control Protocol) was invented to reduce the burden of configuring network devices. A DHCP server gives a client its IP address, subnet mask and default router. If the network's topology changes, the network engineer simply changes the DHCP server's information. The only configuration necessary on the client is to make the client refresh its DHCP information. Devices that have static IP address can live in a DHCP network segment, provided the IP address is not used and the DHCP server is properly configured.

For diskless clients, DHCP is an absolute necessity. When a diskless client starts, it sends out a BOOTP packet using the network broadcast address 255.255.255.255. Basically, this packet asks the question "This is my Ethernet address, what are my network settings?" on the network segment.

The DHCP server hears the request and then leases an IP address, subnet mask and default router to the diskless client. Other network information can also be given to the client through DHCP. Some routers can act as a BOOTP relay agent and direct the client's broadcast message to a DHCP server, even to servers not on the same network segment.

DHCP is also very useful for laptop users and roaming users. These users need different network settings, depending on where they attach their systems. This information can be provided by a DHCP server in the network segment. In most cases, not all the roaming users will have their computer turned on

at the same time. Because of this use pattern, most networks can assign a small pool of IP addresses to serve a much larger pool of actual clients.

Solaris 9 has a built in DHCP server. This server is an industry compliant DHCP IPv4 server. However, this DHCP server does not work with IPv6. Understand that IPv6 takes care of a lot of its own housekeeping.

There are two main tools used to configure DHCP. The DHCP manager is a Java-based GUI tool that has various configuration wizards and view panes. The command line tool **`dhcpconfig`** can perform most of the same functions as the DHCP manager, except that it works without the need for a graphical display system or X server.

Quick Tip

- Sometimes when readers set up a DHCP (Domain Handling Control Protocol) server they notice some DNS (Domain Name Service) server settings, such as the DNS server's IP address and domain. Understand that these settings only point to a DNS server. They do not mean that you are setting up a DNS server.

- If you type **ping <hostname>.<domain>** and you get the message **<hostname>.<domain> is alive**, this most likely indicates that a DNS server is active on this domain and is resolving this host and this domain.

- If you type **ping <hostname>.<domain>** and you get back the response: <**hostname>.<domain> not found**, this indicates that a DNS server is not actively resolving this workstation's DNS name.

- Understand that a DNS server and a DHCP server are different types of servers, and that DHCP settings only *point* to a DNS server. The DHCP server can not perform hostname/domain resolution; only a DNS server performs that role.

How DHCP Works

DHCP generally works as follows:

1. A client that does not have an IP address starts up. It sends out what is known as a **`BOOTP`** packet. This packet has the client's Ethernet card address (example **`0:3:ba:4:c1:3a`**). The IP address of this packet is the network broadcast address 255.255.255.255. This address is received by everything on the same network segment. The DHCP server examines the broadcast packet and notes the client's Ethernet address. If the Solaris 9 server is setup as a DHCP relay agent, it will send the contents of the broadcast packet to the DHCP server that is responsible for that network. Some advanced routers can also interpret the packet as a **`BOOTP`** packet and route the **`BOOTP`** packet to a DHCP server on another network segment.

2. If a DHCP server only has one Ethernet card installed on one network segment, it uses the appropriate network table to find an IP address for the DHCP client. If a Solaris 9 server has multiple Ethernet cards, it check to see which card the broadcast packet arrived through. This tells the DHCP server which network the packet came from. If the DHCP request came from a DHCP relay agent, the DHCP

server looks at the DHCP relay agent's IP address to determine which network the DHCP client resides on.

After determining which network the broadcast packet arrived from, the DHCP server checks its list of IP addresses from a network table. This table has a network number, a list of IP addresses that are being used, and a list of IP addresses that are not being used. The network table and lists of IP address must be set up by the system administrator during the installation of the DHCP server.

Once the server finds an IP address that seems to be unused, it uses the **`ping`** command to test the network to verify that the IP address is truly not being used.

3. If the candidate IP address is free, the DHCP sends back the DHCP information to the same router (if it came from a DHCP relay agent) or network segment from which the DHCP client made the request. The client looks for its Ethernet address in any broadcast message it sees. Understand that at this point in time, the DHCP client does not have any network configuration information. The DHCP client and the DHCP server must communicate through broadcast messages until the client can configure its network segments.

   If multiple DHCP servers have been installed, there is a possibility that the DHCP client can contact several DHCP servers through its broadcast message. This is rare, because most network engineers are very careful to have only one DHCP server assigned to a network segment. On the rare occasions when a client receives multiple DHCP packets from different servers, it will select the first DHCP packet that it receives.

4. While the client is reconfiguring itself with the new network information, the DHCP server places a temporary checkmark next to the assigned IP address. When the client receives its network configuration information, it sends back an acknowledgement packet (**`ACK`**) back to the DHCP server. If the client never responds back, the DHCP server will eventually make that IP address available for re-use.

   If there are two or more DHCP servers on a network segment (rare), one server will receive an acknowledgement and will mark off the IP address from its list of available IP addresses. The other DHCP servers will eventually discard the temporary IP assignment and return the IP address back into its lists of available IP addresses.

5. One of the pieces of information that comes with the DHCP packet is the *lease time* of the IP address. This specifies the amount of time that the network information is valid. At the end of the lease time, the client sends a request for a renewal of the lease to the DHCP server. If the original leasing DHCP server can not be contacted, the DHCP client sends out broadcast messages, looking for another DHCP server to give it an IP address. If a new DHCP lease can not be obtained, the DHCP client disables the Ethernet card that was configured with DHCP information.

6. If the client shuts down, it sends a message to the DHCP server, letting the DHCP server know that it does not need the IP address anymore. This is necessary so that valid IP address are not being held for a computer that might not ever return.

DHCP Key Points

- A DHCP server can act as a BOOTP relay agent or as a DHCP server, but not both. Either a DHCP server handles DHCP requests, or it passes DHCP requests to other servers.

- A DHCP server can handle a limited number of clients, based upon the database that the DHCP server uses:

  Binary Files 100,000 clients

NIS+ Database	40,000 clients
Text Files	10,000 clients

- Always monitor the network traffic around DHCP servers. DHCP servers and DHCP clients use the 255.255.255.255 network broadcast packet. This packet is broadcast to all network components on a network segment. Try to configure the DHCP server so it is in close proximity to the DHCP clients.

- Never set up any type of server as a DHCP client. DHCP is great for workstations and laptops, but a server needs to have a static IP address. If something happens to the DHCP server, it might send incorrect information that could bring down the entire server structure in a company. Also, there are plenty of software programs, routers and gateways, and that reference a server's IP address. Severs should never be DHCP clients for any reason.

- A DHCP server can not use DHCP itself. A DHCP server must always have a static IP address to ensure the packets can be sent by DHCP relay agents to a DHCP server.

- Be careful when setting up the pool of available IP addresses in a DHCP server. Make sure that the list of available IP addresses does not include the IP address of a server or router or other critical network device!

  Imagine what could happen if the static IP address of a server was included in the list of available DHCP addresses. Let's say that a server is taken down for maintenance. Now let's say that during the maintenance cycle, a DHCP server instructed a DHCP client to use that same IP address. When the server or network device came back to life, it would not be able to communicate on the network because its IP address would be in use by something else. To avoid this problem, either disable the IP address in the DHCP server, make the IP address a static address assigned to that device (without a lease expiration), or do not include the IP address in the server's list of available IP addresses.

WARNING

- The DHCP server in Solaris 9 has a feature that lets this server create client hostnames. This is a very bad feature, which should not be used. Imagine trying to track down hostnames and IP addresses that *both* change. This also causes problems when it comes to network sharing and other specific network settings.

- Do not use DHCP with a server. If something happens to the DHCP server, the client's Ethernet card will be disabled if it can not contact a DHCP server for an IP lease renewal. Why have another server go down with the DHCP server?

- If the number of clients exceeds the number of IP addresses, monitor the DHCP pool. If more than 90% of the IP addresses are leased out, consider increasing the IP address pool.

DHCP Files

Some critical DHCP run control files are used with DHCP. Only senior level administrators should work with these files. These files are presented here for reference only. If there is a problem with

DHCP starting, check to make sure these files exist. Also, understand that if the option Services, Disabled is chosen in the DHCP manager, the DHCP server will not automatically start when Solaris 9 is rebooted.

The run control scripts that relate to DHCP include the following:

`/etc/rc0.d/K21dhcp`	The shutdown script in the **`/etc/rc0.d`** directory that turns off the DHCP server gracefully when the operating system shuts down. The **`K21dhcp`** script is also in the **`/etc/rc1.d`** directory and the **`/etc/rc2.d`** directory.
`/etc/rc0.d/K43inetsvc`	The shutdown script in the **`/etc/rc0.d`** directory that is used to shut down TCP/IP services. This helps to turns off the DHCP server gracefully when the operating system shuts down. The **`K43inetsvc`** script is also in the **`/etc/rc1.d`** directory.
`/etc/rc0.d/K90dhcpagent`	The shutdown script in the **`/etc/rc0.d`** directory that turns off the DHCP agent. The DHCP agent is the client side of DHCP that requests information from a DHCP server. This script is used on DHCP clients to gracefully exit when the operating system shuts down. The IP address is released from the client when the server shuts down (provided that the IP address is not reserved).
`/etc/rc1.d/K42inetsvc`	A script that works with NIS and NIS+ maps. This script only affects the DHCP server if the DHCP server is working with a NIS or NIS+ database. Among other things, this script releases an Ethernet card from DHCP control.
`/etc/rc2.d/S72inetsvc`	A script that works with NIS and NIS+ maps. This script only affects the DHCP server if the DHCP server is working with a NIS or NIS+ database. Among other things, this script sets the variable **dnsservers** to DHCP if DHCP is enabled on a system. This ensures that the workstation gets its network settings from DHCP, and not from a NIS or NIS+ server.
`/etc/rc2.d/S69inetsvc`	A script that configures the **`sysidnet`** used for network parameters.
`/etc/inet/dhcpsvc.conf`	A text file in the **`/etc/inet`** directory that shows the basic configuration of the DHCP server.

Some common variables are:

DAEMON_ENABLED=TRUE/FALSE	DCHP server enabled or disabled.
RUN_MODE=server/relay	DHCP server set up as a DHCP server or a DHCP relay agent.
RESOURCE=SUNWfiles	DHCP information is in a text file.
SUNWbinfiles	DHCP information is in a binary file.
SUNWnisplus	DHCP information is in a NIS+ map.
PATH=/path/to/data/source	Location of the DHCP data file.
CONVER=1	Version of the public module. This variable should not be changed.

The DHCP Manager

The DHCP server can be configured with command line options, or by using a Java-based GUI known as the *DHCP Manager*. It is important to know how to use command line tools as well as the GUI tool, in case the video display does not work for some reason.

If at all possible, use the DHCP Manager to work with the DHCP server. The DHCP command line commands are extremely picky about syntax and only give vague error messages if something goes wrong.

The DHCP Manager has several functions. The first is to configure DHCP network tables, macros, and options. Other functions including starting and stopping the DHCP server, and enabling or disabling the DHCP server. The configuration of the DHCP server can be exported to a file. This file can then be imported into another DHCP server to duplicate the first server's configuration. The DHCP Manager can also be used to set up the DHCP server as a DHCP relay agent.

One of the benefits of using the DHCP Manager is that the documentation is excellent and there are several installation wizards that guide the administrator through the configuration of DHCP. The installation wizards also check the syntax of the data entered.

A DHCP relay agent is a server that sits on the same network segment as some DHCP clients. When a DHCP client becomes awake, it sends out a broadcast message trying to find a DHCP server. The DHCP relay agent receives the broadcast packets and packages them into IP packets. The IP packet is then sent to a DHCP server over the network. A DHCP relay agent is necessary because the DHCP client can not talk to a DHCP server over the network (because the client does not have an IP address, subnet mask or default router). The DHCP server then sends back the client's configuration information to the DHCP relay agent. The DHCP relay agent sends the information back to the DHCP client. Once the DHCP client has its network configuration set up, the DHCP relay agent is not needed anymore.

One benefit of a DHCP relay agent is that the DHCP server can be centrally located. This makes it easier to manage the DHCP servers in a company. Another benefit is that routers do not have to be set up to pass **BOOTP** packets. If the network topology changes, it's easier to reconfigure a DHCP relay agent than it is to reconfigure routers to handle the network traffic to the DHCP servers. Figure 31.1 shows the DHCP Manager, in which the system administrator can manage the DHCP server with a GUI interface.

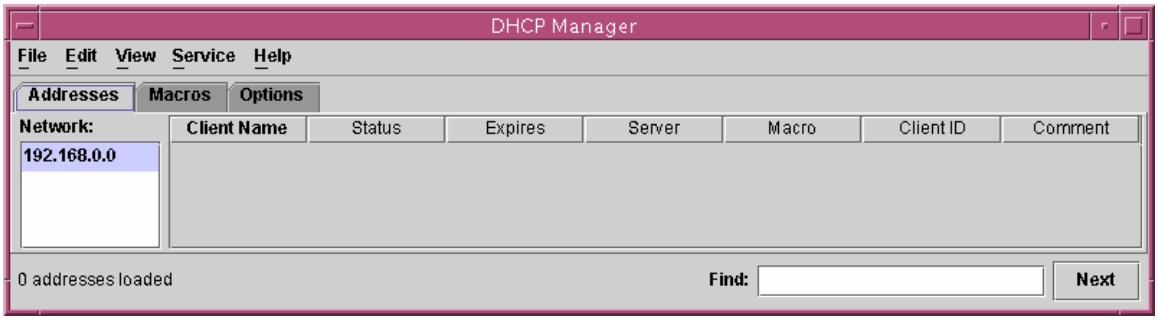


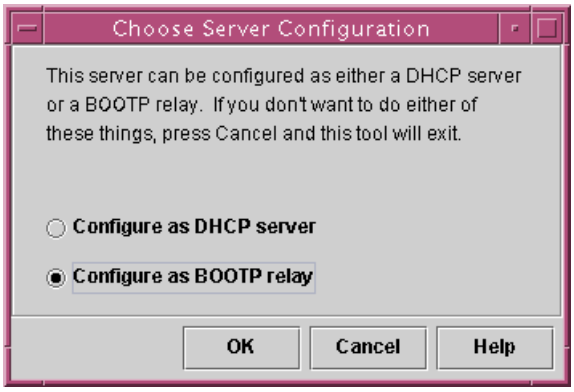
Figure 31.1 The DHCP Manager

Lesson 31.1 Set up a Server as a DHCP Relay Agent

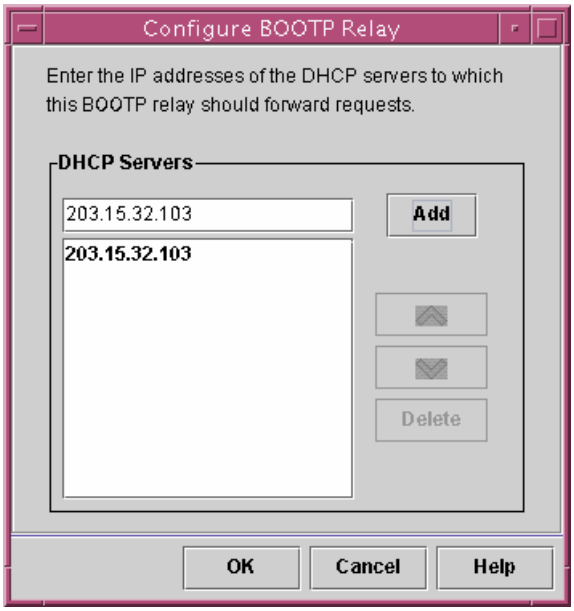
This lesson shows how to set up a DHCP relay agent. A relay agent is used to forward DHCP broadcast packets to a specific DHCP server. Understand that with Solaris 9, a DHCP server can only function as a DHCP server or a DHCP relay agent; it can not be both. In this lesson, the DHCP relay agent sends DHCP packets to a fictitious DHCP server. This will cause no harm, as long as the DHCP server is not connected to the Internet (in case there is a device on the IP address used in the lesson).

1. Log in as the root user.
2. Open a Terminal window.
   To open a terminal window, right click anywhere in unoccupied desktop space. The Workspace menu will appear. Left click on the Tools menu item, then left click on the Terminal window icon.
3. Type the command **`/usr/sbin/dhcpconfig -U -h -x`**
   This command is used to un-configure a DHCP server. If an error message appears that says **dhcpconfig: Error - reading DHCP configuration file. No such file or directory** *that only indicates that a DHCP server does not exist.*
4. Type the command **`/usr/sadm/admin/bin/dhcpmgr &`**
   The ***`dhcpmgr`*** *command starts the DHCP manager, the GUI tool used to manage a DHCP server in Solaris 9. The following window should now appear:*

5. Type the command **`/usr/sadm/admin/bin/dhcpmgr &`**
   The ***`dhcpmgr`*** *command starts the DHCP manager, the GUI tool used to manage a DHCP server in Solaris 9. The following window should now appear:*



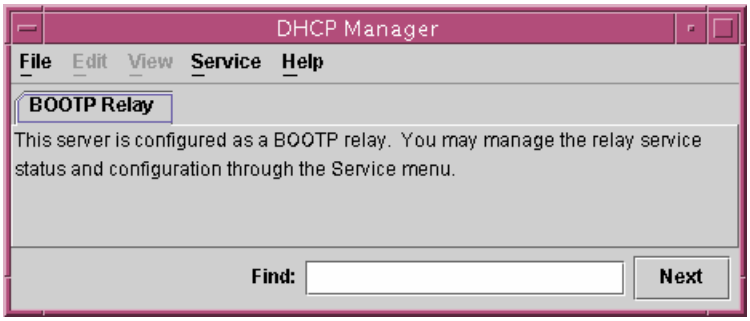
6. Select ◉ "Configure as BOOTP relay" and click OK.
   The following window should now appear:



7. Type in a fictitious IP address for a DHCP server, as shown above. Then left click the OK button.

   In the above figure, several menu items are grayed out, because they can not be used on a BOOTP relay agent. This is a rather chopped down version of the DHCP manager. This DHCP manager only has configuration information for the DHCP relay agent.

   The following window will now appear:



8. Left click on Service in the Menu bar.
   The Find text box at the bottom of the screen is used to find a particular IP address. This is a useful troubleshooting tool. Experiment with this feature if you want to; it can not harm your system or network.

   The Service menu has several options that should be understood:

Restart - Force the ***in.dhcpd*** *daemon to re-read all of its configuration information*
Stop - Stop the DHCP ***in.dhcpd*** *daemon*
Start - Start the DHCP ***in.dhcpd*** *daemon*
Disable - Don't start the DHCP server when Solaris 9 starts or when Solaris 9 is rebooted
Enable - Start the DHCP server when Solaris 9 starts or when Solaris 9 is rebooted
Modify - Modify the DHCP relay agent

9. Left click on Modify.
   The Modify Service Options window will appear.
10. Left click on the Options tab.
    The window shown in Figure 31.2 will appear.

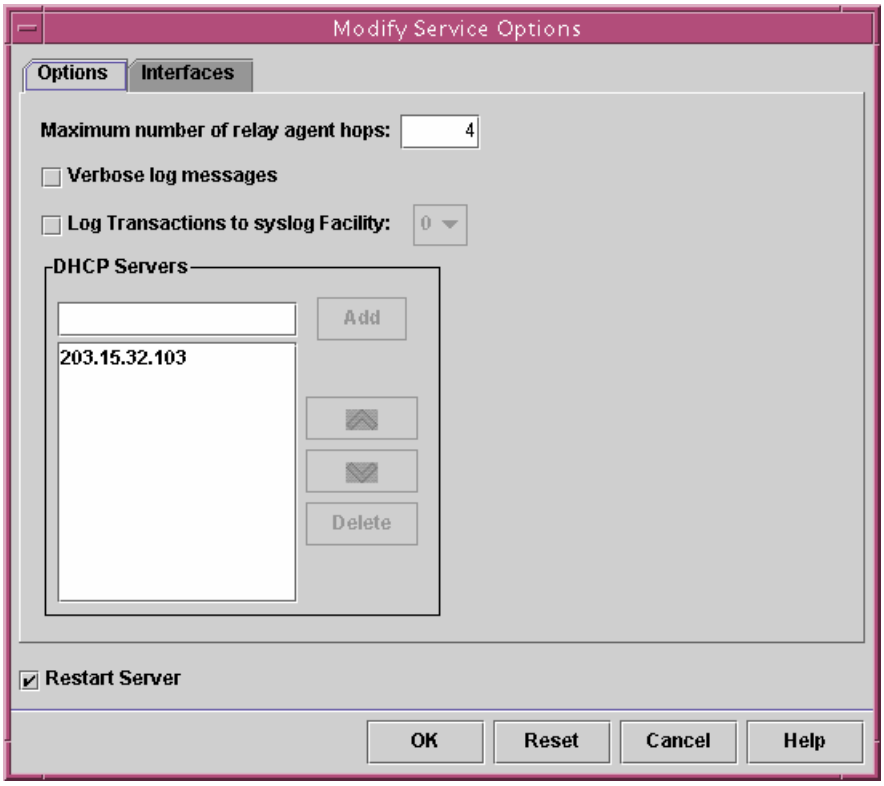


Figure 31.2 Options Tab of the Modify Service Options Window

The first selection item is “Maximum number of relay agent hops:" *This indicates the maximum number of times the DHCP relay agent will try to contact a DHCP server before it quits.*

- Leave the default value of 4 alone.
- Do not select the ☐ "Verbose log messages" or ☐ "Log Transactions to syslog Facility" choices.

11. Make sure that the checkbox ☑ "Restart Server" is checked (it is at the lower left of the window). *Then left click on OK.*
    Anytime a change is made to the DHCP server, restart the server. This only restarts the DHCP server, not the operating system. Because DHCP clients try several times to contact a DHCP server, there should not be any significant DHCP problems during a restart of the DHCP from the DHCP manager.
12. Left click on the Interfaces tab.
    The window shown in Figure 31.3 will appear. This window shows what Ethernet cards and network segments the DHCP relay agent should monitor (or ignore) for DHCP client broadcast messages. For servers with only one Ethernet card, this option really does not apply. For servers with multiple Ethernet cards, this is a convenient way to monitor different networks.

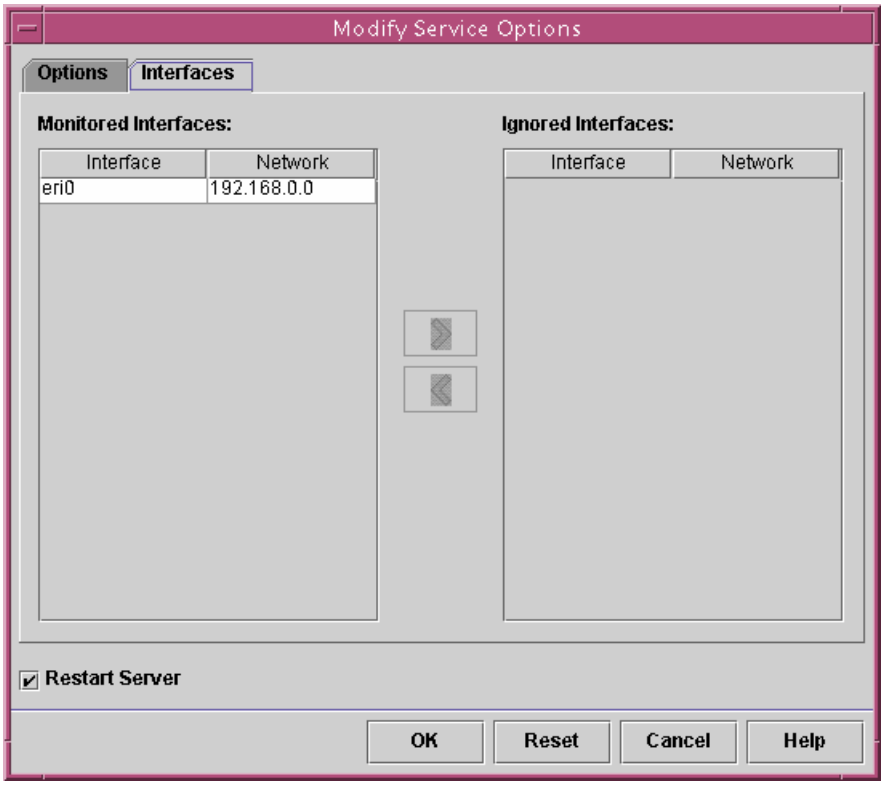


Figure 31.3 Interfaces Tab of the Modify Service Options Window

13. Left click on the OK button.
    Feel free to explore the DHCP relay agent, but do not make any permanent changes to the system. Whey you are done, click on Cancel so that any changes you have made are not permanent.
14. When you are finished, left double click on the Windows Menu button on the top right corner of the window. It looks like a dash inside a small square.

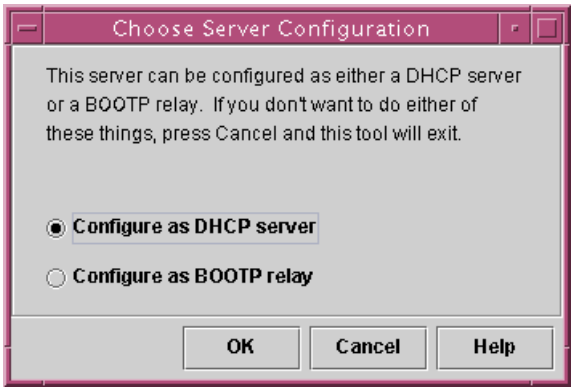
Lesson 31.2 Creating a DHCP Server with the DHCP Manager

WARNING

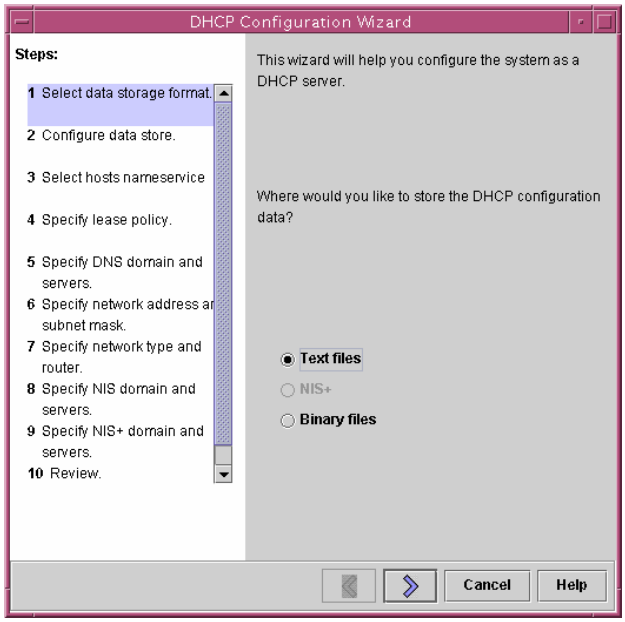
If your test system is on a live network, it could take DHCP requests from company clients. Make sure that the network segment is isolated from the rest of the network. If this system can not be disabled, make sure there are no other DHCP clients on the network segment, and that he router (which separates different network segments) has BOOTP disabled for this network segment. If DHCP is configured for this network segment, it will entertain DHCP requests.

This lesson focuses on the DHCP Manager. The DHCP manager is a Java-based GUI tool for managing the DHCP server. In this lesson the DHCP relay agent created in the previous lesson will be destroyed. The DHCP manager will then be used to set up this system as a DHCP server.

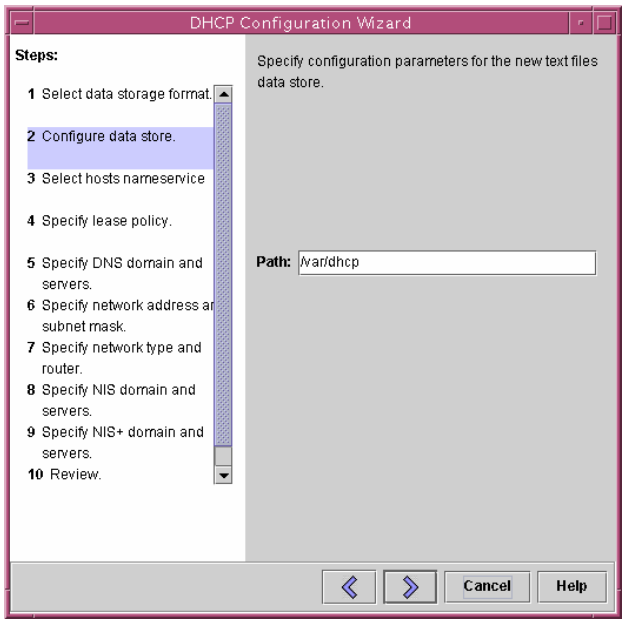
1. Log in as the root user.
2. Open a Terminal window.
3. Type the command **`dhcpconfig -U -h -x`**
   This command is used to un-configure a DHCP server. Don't worry if an error message appears that says **`dhcpconfig: Error - reading DHCP configuration file. No such file or directory`**; *this only indicates that a DHCP server does not exist.*
4. Type the command **`/usr/sadm/admin/bin/dhcpmgr &`**
   When the DHCP server starts, if and DHCP has never been set up, the following window pops up:



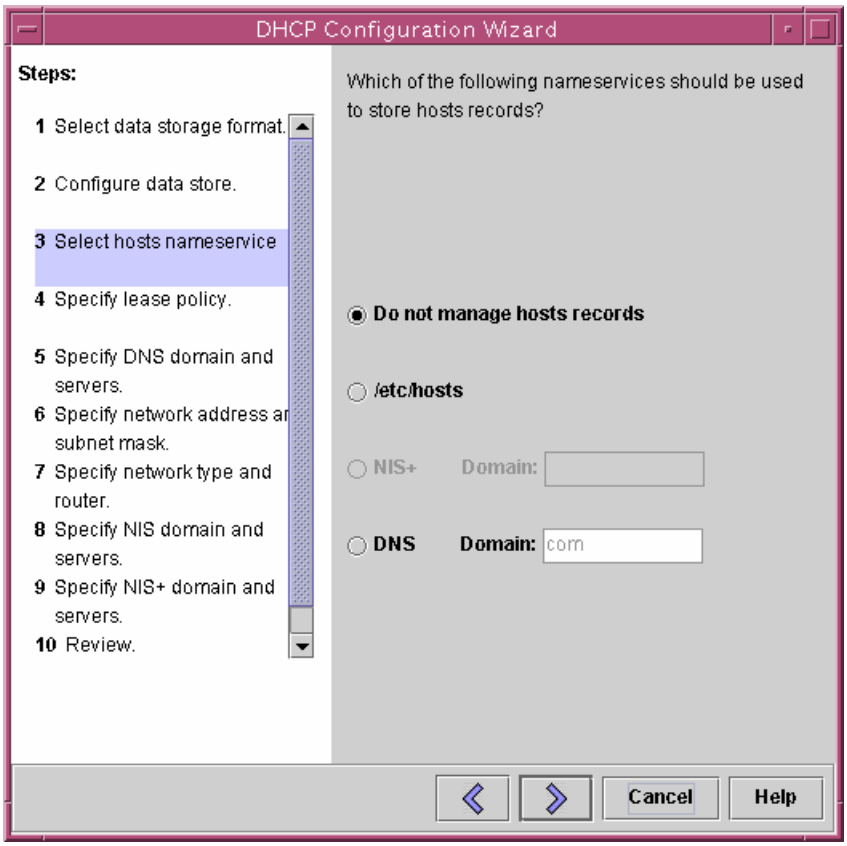
5. Select ◉ "Configure as DHCP server" and left click on OK.
   The DHCP Configuration Wizard starts. The first screen asks for the Data storage format. If NIS+ is not enabled, the NIS+ option is grayed out



6. Select ⦿ "Text files" and left click on ➢
   The next screen asks for the path location for the DHCP files. This is only asking for the directory structure.



7. Just click on ➢. *Don't add any file names to this path:* ***/var/dhcp*** *is the default option.*
   The third screen asks which name service should be used to store host records. A name service matches IP addresses to a hostname.

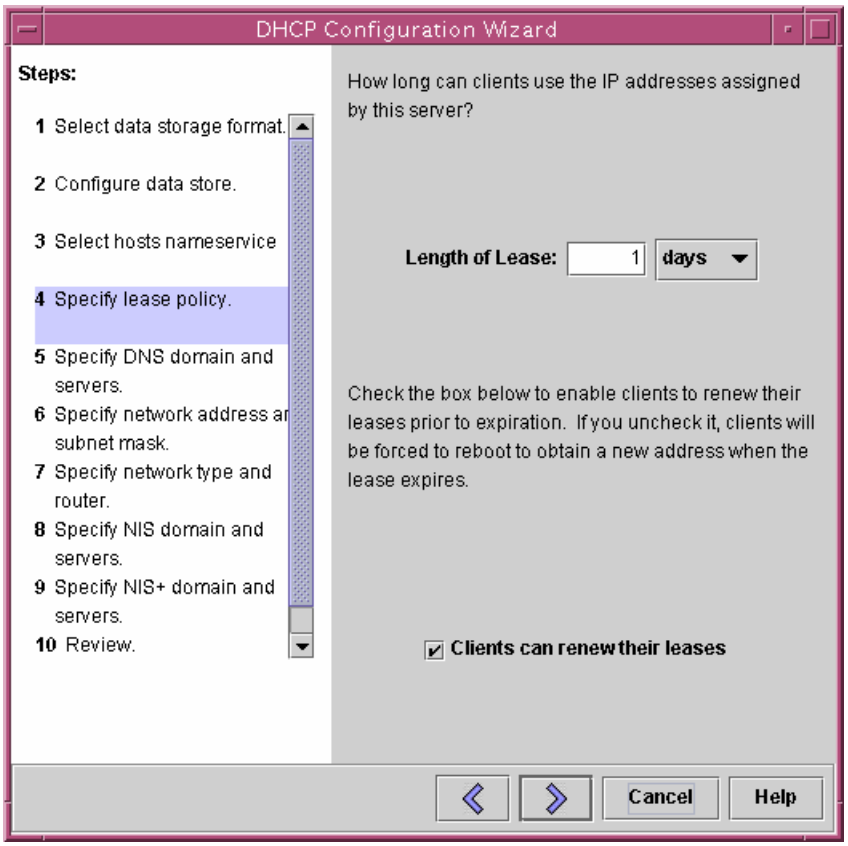


8. Select ⦿ "Do not manage hosts records" and click on ➢.

   Notes: *(1) If you have a very small network or are just working with a test workstation, you could select ⦿ /etc/hosts. When this option is selected, all configuration changes are saved in the* ***`/etc/hosts`*** *file.*

   (2) If you have a large network, you could select ⦿ DNS (provided that the DNS server can work with DHCP).

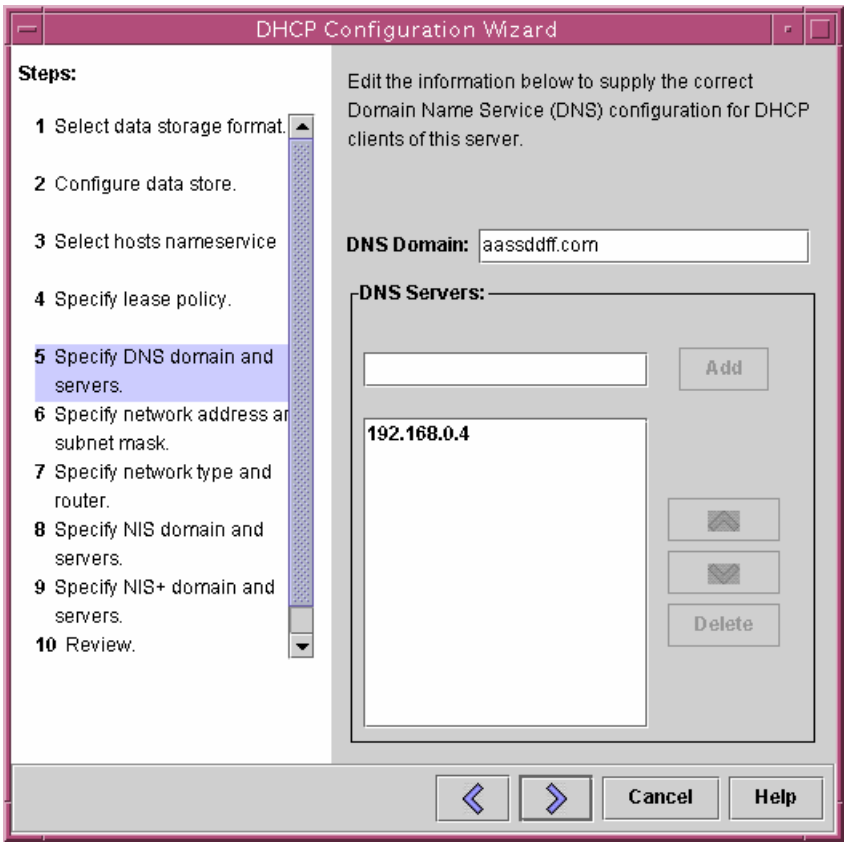
   The next box sets the amount of time a client can have an IP address before the lease has to be renewed.



9. Fill in this window as follows:
   a. In the "Length of Lease" text box, enter **1**.
   b. Make sure that the checkbox ☑ "Clients can renew their leases" is checked.
   c. Click on ➢.

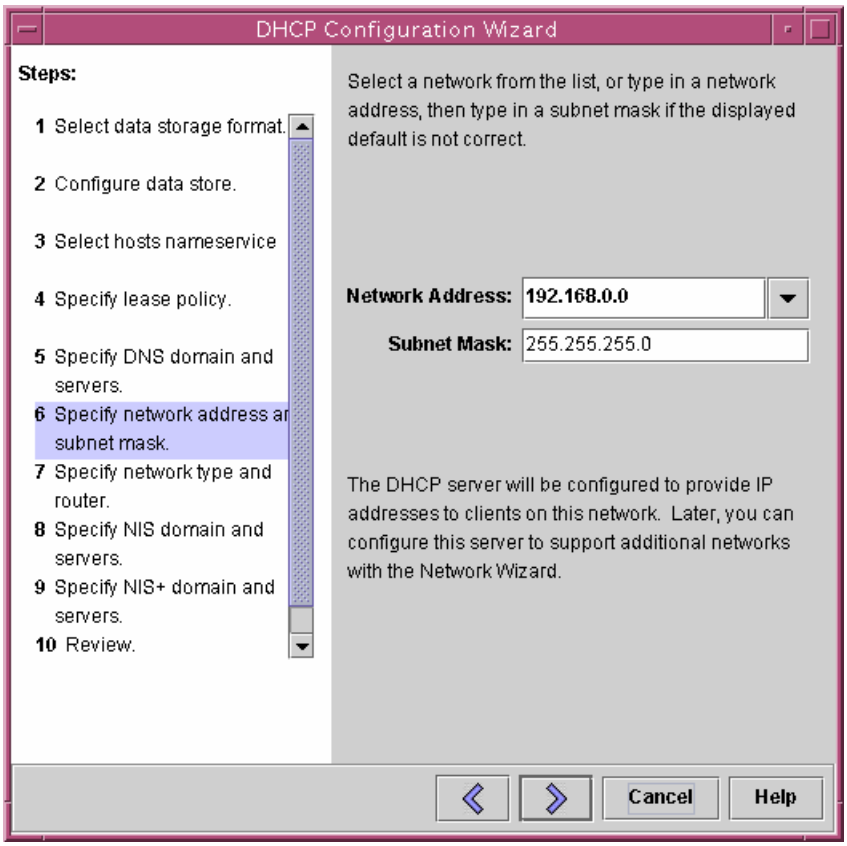
   It is not a good idea to require clients to reboot to obtain a new address. ISPs (Internet Service Providers) usually require this option to prevent customers from using DHCP to set up an IP address and creating a server from that IP address.

   The fifth screen asks for the name of DNS servers, among other information:



10. Fill in this window as follows:
    a. In the DNS Domain text box, type in your domain name (if you have one). If you do not have a domain name, leave this text box blank.
    b. For the DNS Servers area, proceed as follows:
        - If your network can connect to a DNS server, in the DNS Servers area, type the IP address of any DNS servers that are used to resolve hostnames to IP addresses for this workstation. Then left click on the Add button so that the IP addresses are recorded.
          This DNS information will be passed on to DHCP client workstations.
        - If your network cannot connect to a DNS server, leave this area blank.
    c. Click the ➢button.

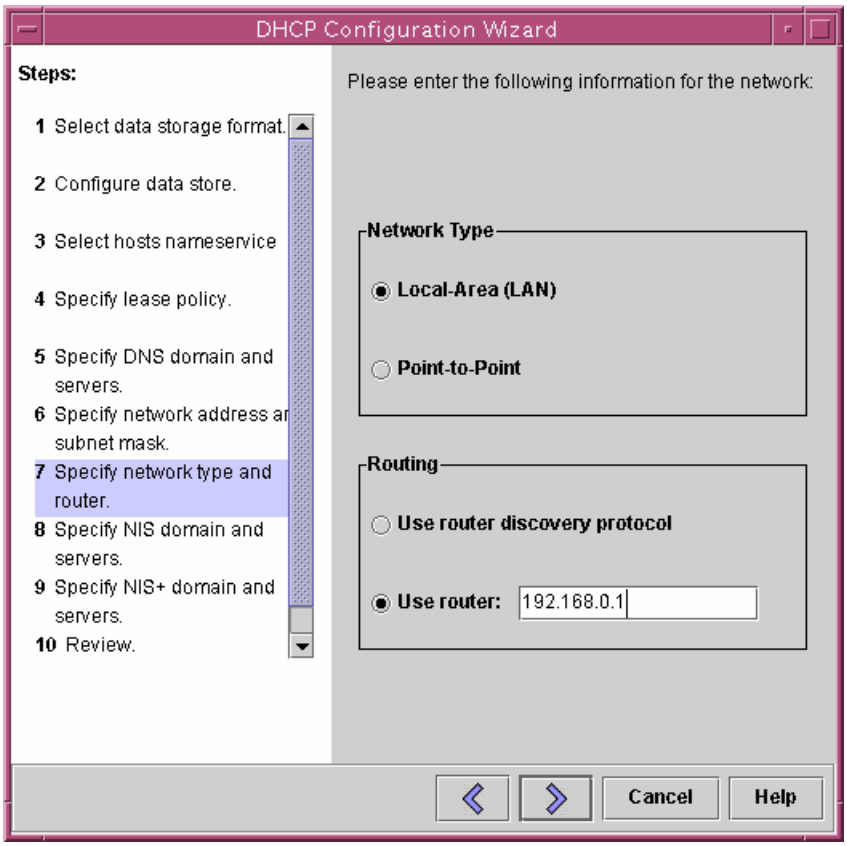
    The sixth screen asks the user for the Network Address and Subnet Mask that the DHCP server will work with.



11. *This window will try to display the correct Network Address and Subnet Mask, based upon file that it finds in the* ***`/etc`*** *directory.*
    - If these values are correct, click the ➢ button and continue with the next numbered step.

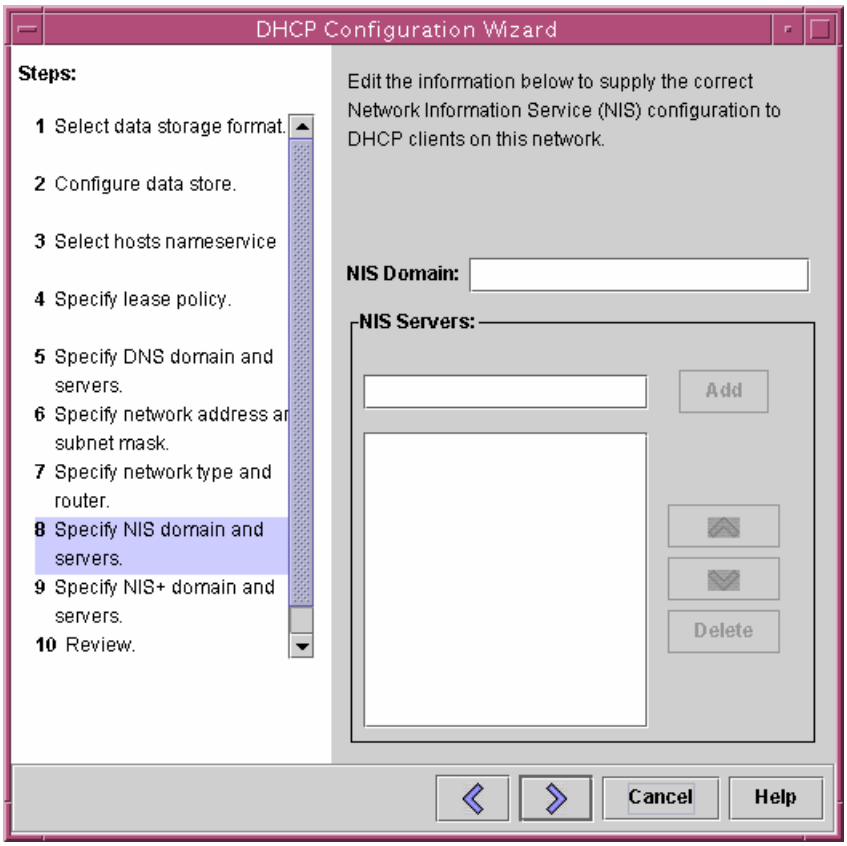
    - If these values are not correct:
        a. In the Network Address text box, select a network list, or type in a Network Address. If you have followed the previous lessons in this book type **`192.168.0.0`** here.
        b. In the Subnet Mask text box, type in the appropriate Subnet Mask, as follows:
            - Class A Subnet: 255.0.0.0
            - Class B Subnet: 255.255.0.0
            - Class C Subnet: 255.255.255.0
            - Variable length or non-standard subnet mask (a mask that uses any number other than 0 and 255), consult network subnet tables to determine your network address.
        c. Click the ➢ button.

The seventh screen asks for the Network Type. This can be a Local-Area Network (LAN) or Point-to-Point. If the computer is attached to the network, it is on the LAN If the computer connects through a modem or other terminal line service, it connects via Point-to-Point. It is also possible to use the router discovery protocol or to specify a router's IP address.

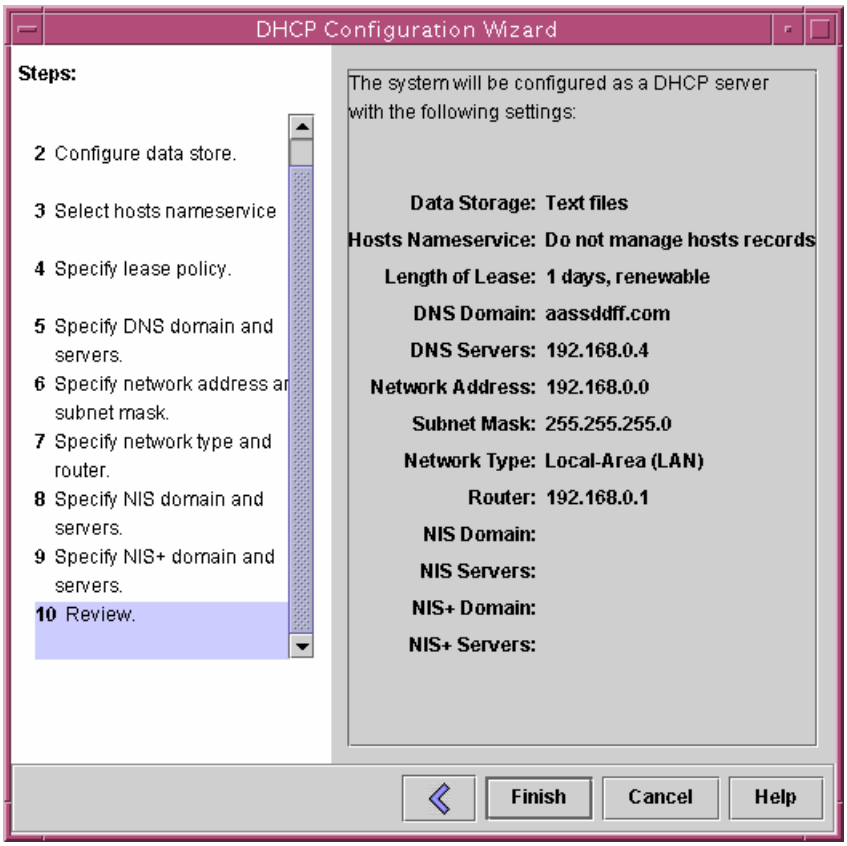


.

12. In this example, specify a router.
    The eighth and ninth screens ask for the location of NIS and NIS+ domain servers. These are NIS and NIS+ servers for the DHCP server only.



13. Leave these fields blank in both screens. Just left click on the ➢ button.
The tenth screen lets you review the information that you have entered.

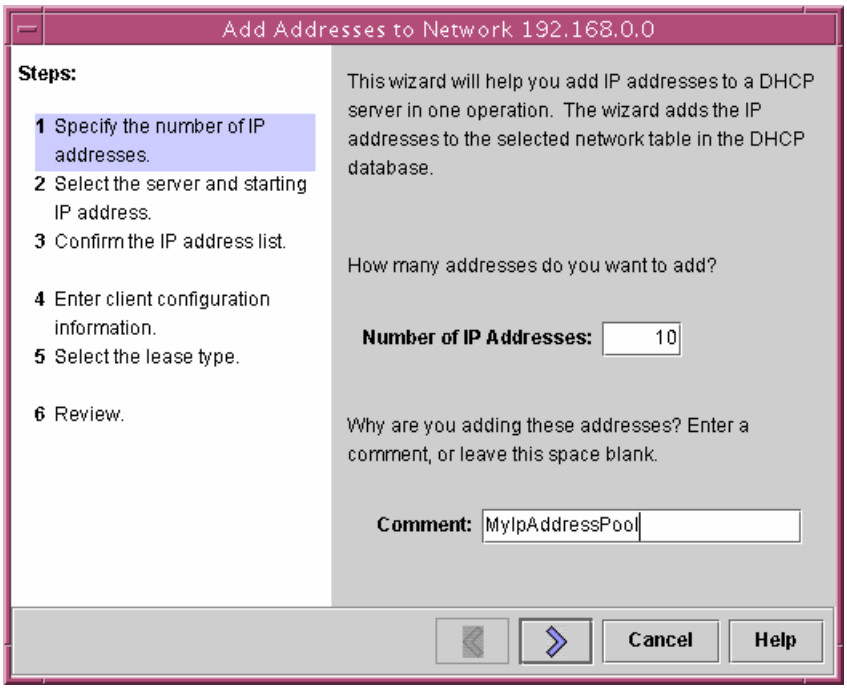


14. Carefully read the information in the right panel.

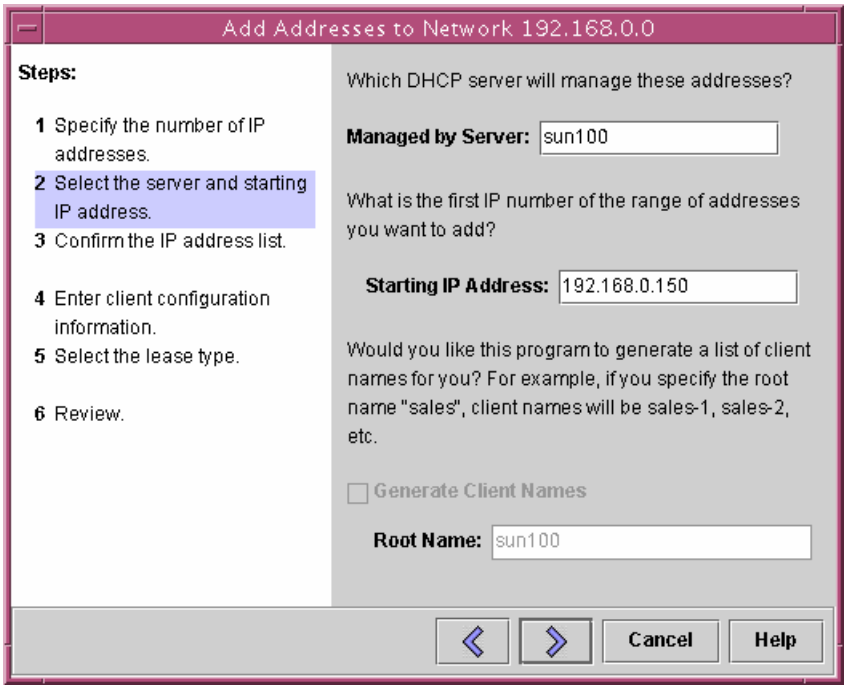
    - If any information is incorrect, left click on the ≺ button as needed to reach the screen for entering that information. After correcting the information, use the OK or ➢ button on each screen to navigate back to the screen shown above.

    - If all the information is correct, left click on the Finish button.
      After the DHCP server has its initial settings, it still needs to have the pool of IP addresses specified. The previous information only told the DHCP server what type of network it was on. When the DHCP Manager starts again, a popup window appears as shown bellow:

15. Press the Yes button.
    This wizard has six steps
    The first window asks for the number of IP addresses and for a comment for this IP address pool.



16. In the Number of IP Addresses box, type **`10`**.
    In the Comment box, type **`myIPAddressPool`**
    The second screen asks which DHCP server will manage these addresses.



17. Fill out this screen as follows:

a. In the Managed by Server text box, enter the hostname or IP address of this system. This is the DHCP server.

b. In the Starting IP Address text box, enter the Starting IP address.
*It would most likely be safe to use **`192.168.0.150`**, for the reasons noted below.*

Note: *Only the starting IP address is requested. This is used as the starting point for generating the number of IP addresses requested in the previous step. For example, if you specified 10 IP addresses there and specified a starting address of 192.168.0.150 here, you would be allocated the addresses from 192.168.0.150 through 192.168.0.159.*

*The starting IP address depends on your network topology. Most VPN networks that use the 192.168.x.x number scheme don't use IP addresses above **`192.168.0.100`**. Most routers and ISPs reserve the use of numbers less than 100 for internal use and dedicated functions. The address 192.168.0.150 should be a safe one to use.*

To see if an IP address is already in use, open a Terminal session and type

```
ping <IP_address_to_check>
```

Some networks disable ping responses to prevent ping denial of service attacks, so this might not be an accurate way of detecting IP addresses. You can also try the following commands:

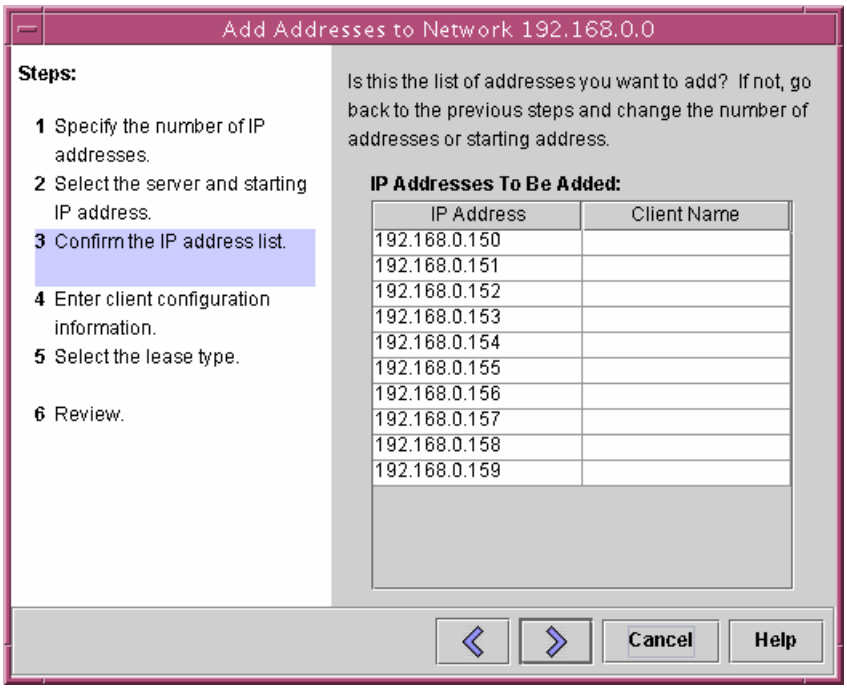
```
telnet <IP_address_to_check>
ftp <IP_address_to_check>
spray <IP_address_to_check>
```

Most likely the device will respond to one of these commands.

c. Make sure that the ☐ Generate Client Names box remains unchecked.
Do not let the DHCP server generate client names! Allowing it to do so can create a real problem when it comes to troubleshooting! It's hard enough to track down a hostname with DHCP, but trying to track down both a hostname and an IP address would be extremely difficult.

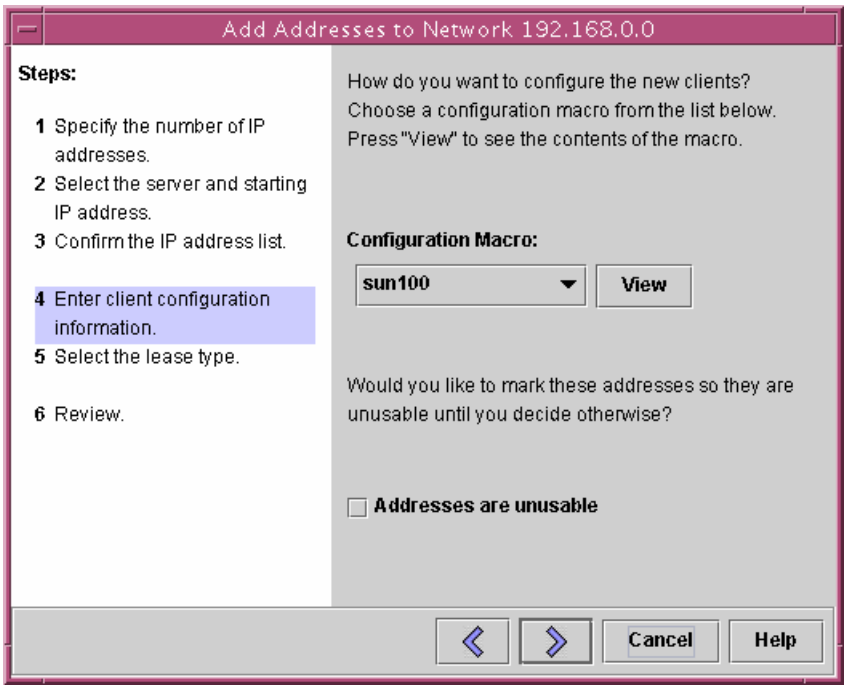
d. Click on ➢

The third screen shows the DHCP addresses that were created.



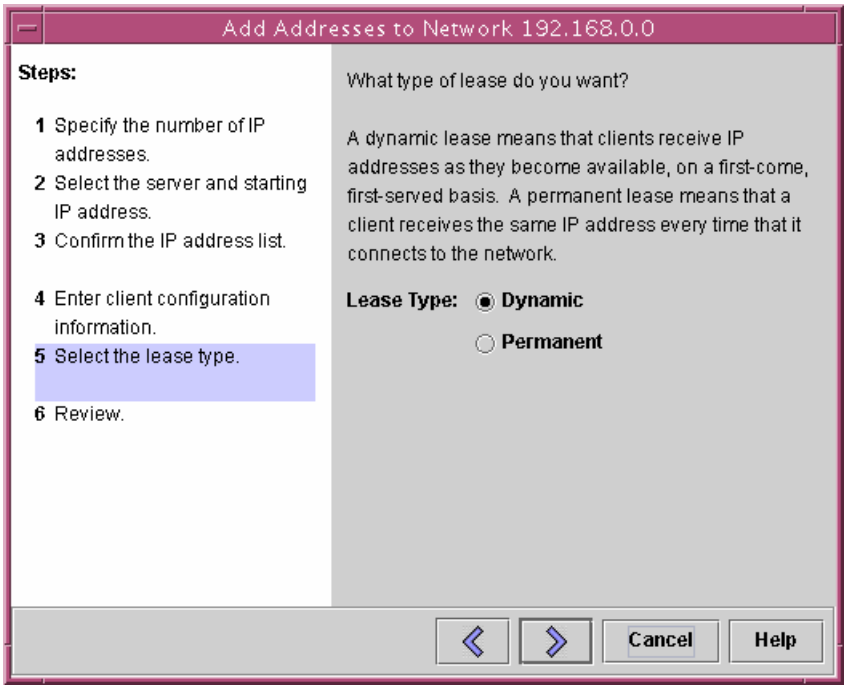
18. If the list of IP addresses is correct, click ➢ and go to the next step. If the list is incorrect, click ➢ and repeat the previous step.
    The fourth screen shows the Configuration Macro that will be associated with the pool of IP addresses. A macro has all the network settings related to DHCP. These settings can be changed at a later time.

    Notice the checkbox that says "Addresses are unusable." This is used when network servers and routers require static IP addresses. This pool of IP address is in DHCP, but nothing can use these IP addresses.

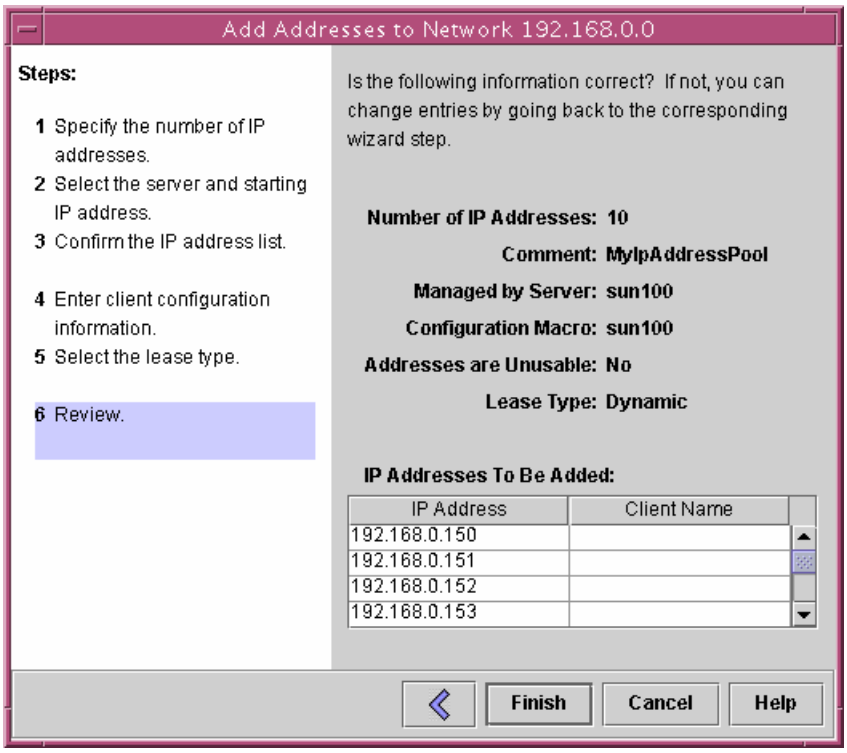


19. Select the configuration macro that matches the system's hostname and then click the ➢ button.
If a macro with the system's hostname is not visible, click on the ▼ button and then choose that macro.
Leave the ☐ "Addresses are unusable" box unchecked.

The fifth screen asks what type of DHCP lease should be presented.



20. Click on ◉ Dynamic.
The last screen is a confirmation screen, as shown below.



- If all the information is correct, left click on the Finish button.

21. In this screen:
    - If any information is incorrect, left click on the ⮜ button as needed to reach the screen for entering that information. After correcting the information, use the OK or ➢ button on each screen to navigate back to the screen shown above.
    - If all the information is correct, left click on the Finish button.

    For some reason, the DHCP server is really slow to create the DHCP addresses. If the screen seems to freeze, don't shut down the server or the DHCP manager. Just wait a short period of time. The DHCP manager should appear again.

Configuring a DHCP Client

Solaris 9 can be a DHCP client or a DHCP server, but it can not be both at the same time. A DHCP server needs to have a static IP address for itself. To make Solaris 9 system operate as a DHCP client, follow the steps in Lesson 31.3.

Lesson 31.3 Creating a DHCP Client

Note: To follow this lesson, you must have more than one SPARC workstation. This lesson is impossible with only one workstation, because a DHCP server can not also be a DHCP client.

1. Log in as the root user.
2. Open a Terminal window.
3. Type the command **`ifconfig -a`**
   This will show all the network interfaces on the system. The Ethernet card's network_interface name is the first word on the left side of the screen. Do not use the ***`lo0`*** *interface or any interface that starts*

with ***lo****. The* ***lo*** *interface is a loopback 127.0.0.1 virtual interface that can not be configured for DHCP.*

4. Type the command touch **/etc/dhcp.<network_interface>**
   Examples: **touch /etc/dhcp.hme0**
   touch /etc/dhcp.eri0
5. Type the command **/usr/sbin ifconfig <network_interface> dhcp start**
   Examples: **/usr/sbin ifconfig hme0 dhcp start**
   /usr/sbin ifconfig eri0 dhcp start
6. Type the command **/usr/sbin/ifconfig -a**
   The Ethernet card should take its network settings from the DHCP server. If for some reason the workstation does not reconfigure itself from the DHCP server, reboot the workstation. Once the workstation starts again, type the command ***/usr/sbin/ifconfig -a*** *to make sure the new network settings are from the DHCP server.*

Quick Tip

Microsoft Windows is compatible with the Solaris 9 DHCP server. To use a Microsoft Windows system as a DHCP client:

1. Set up the Windows system to "Obtain an IP address automatically."
2. Reboot the Windows system.
3. Open an MS-DOS window.
4. Type the command **ipconfig -all**
   This should show the IP address, subnet mask and other DHCP entries that were set up by the server.
5. Type **exit** to return to Windows from the MS-DOS prompt.

Note: You can also use the Windows command **winipcfg** to see these settings in a GUI window. Use the Release All and Renew All buttons to see DHCP in action.

Understanding Macros and Options

Once a DHCP pool of IP address has been created, it needs to have a macro attached to it. The macro contains the information that is assigned to the client. For example, the macro created in the last lesson sends the following variables and their values to a DHCP client

Include	Locale
Timeserv	192.168.0.4
LeaseTim	86400
LeaseNeg	
DNSdmain	aassddff.com
DNSserv	192.168.0.4

As you can see, a macro is a collection of settings for **Options =** variables, such as **Timeserv**, **LeaseTim** and **LeaseNeg**.

A macro's name determines which client or clients will receive the information. For example:

Macro Name	Clients That Receive the Information
192.168.0.5	The client with
192.168.0.0	All clients on the 192.168.0.0 network
SUNW,sun4u	All clients with the sun4u architecture
08002032B3A4	The client with this specific MAC address

Some of these macros may not exist. For example, a system administrator could only make a Network Address macro and not make any others.

There are four official Macro Classes:

Client Class Macro - Clients with the same hardware
Network Address Macro - Clients on a particular network
IP Address Macro - Client with a specific IP address
Ethernet Address Macro - Client with a specific Ethernet MAC address

These macros are processed in following order:
1 - Client Class
2 - Network Address
3 - IP Address
4 - Ethernet Address.

All the macros that apply to a client are processed. If an option is in more than one macro, the last macro that contains the option will set the option's value.

Lesson 31.4 Configuring Macros And Options

Now that the DHCP server has been created with 10 IP addresses, those DHCP clients are using a rather limited set of macros and options. Remember that a Macro holds all the configuration information for a client, such as DNS domain, the DNS server's IP address, and a Web server's IP address. Options that have been configured, such as Timeserv and LeaseTim, are passed to the DHCP client.

In this lesson the reader will add extra options to a macro. The first option, WWWservs is selected and given an IP address. The reader then creates a fictitious option.

Figure 31.4 shows a configured DHCP server.

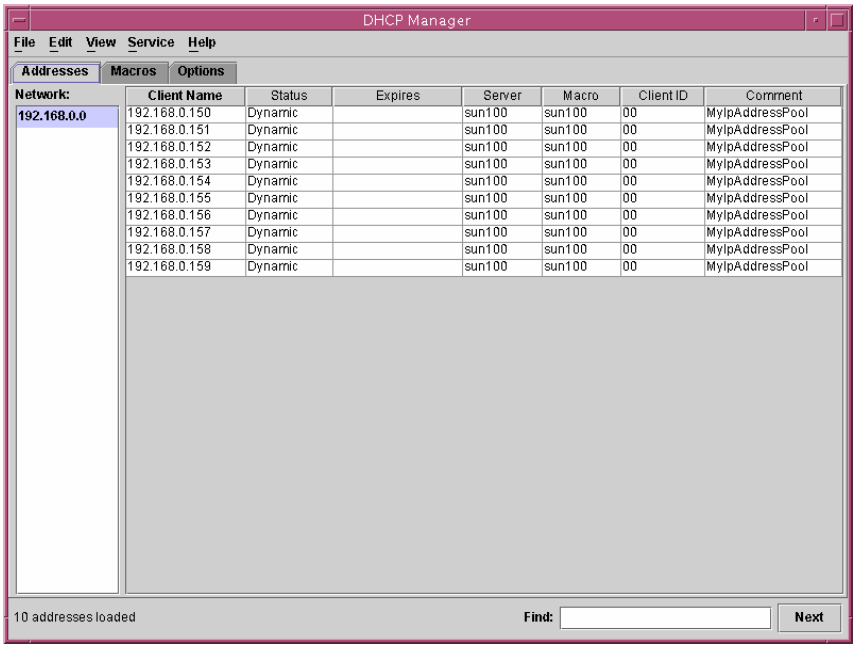


Figure 31.4 DHCP Manager With 10 Addresses

1. Log in as the root user.
2. Start the DHCP manager with the command
   `/usr/sadm/admin/bin/dhcpmgr &`
3. Left click on the Addresses tab.
   The macro assigned to this IP address is listed in the Macro tab. The macro on the author's computer is sun100. The hostname of the author's computer is sun100.
4. Left click on the Macros tab.
5. Left click on Edit.
6. Left click on Properties
   The Macro Properties window should appears, as shown in Figure 31.5.

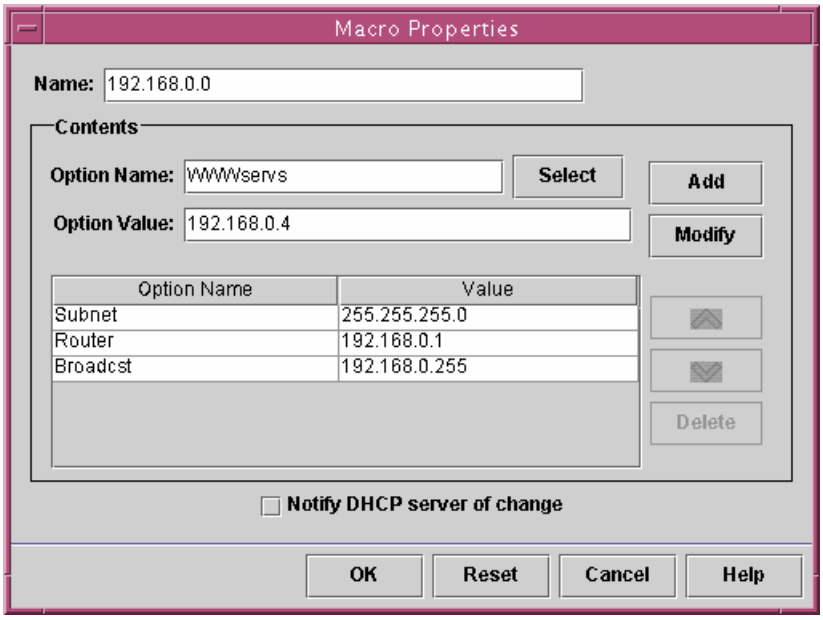


Figure 31.5 Macro Properties Window

7. Left click on the Select button next to the Option Name: text box.
8. Scroll down until the option WWWservs is displayed.
9. Left click on OK.
10. In the Option Value text box, type the IP address of your computer.
11. Left click on the Add button.
12. Left click on OK.
    The final results should look like Figure 31.5.
13. Left click on the Options tab.
14. Left click on Edit in the Menu bar.
15. Left click on Create.
    The Create Option window should appear, as shown in Figure 31.6

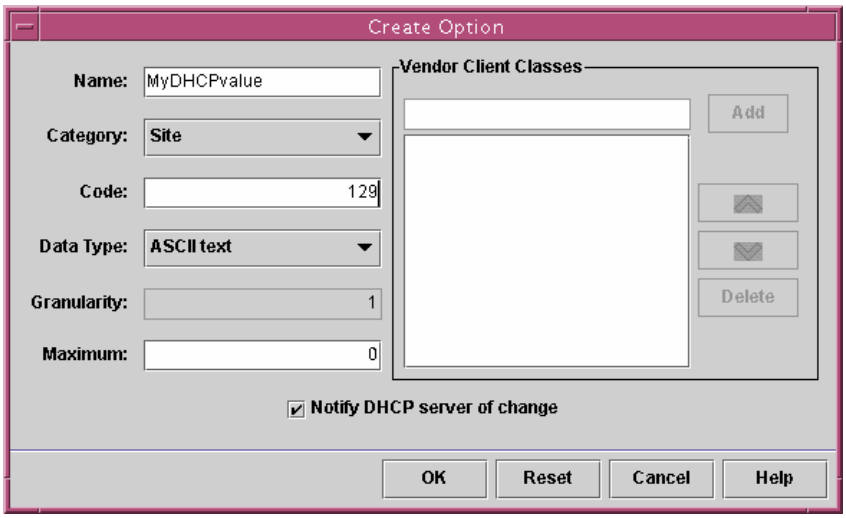
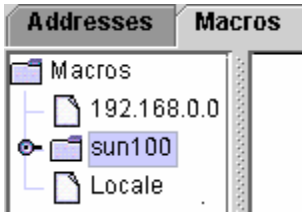


Figure 31.6 Create Option Window

16. In the Name: text box, type **`MyDHCPvalue`**.
    *This step and the next one simply create a dummy option called **MyDHCPvalue** with a dummy value of **129**. The point is simply to enter an option and its value.*
17. In the Code: textbox type **`129`**
18. In the Data Type: field, select ASCII text.
19. In the Maximum text box, type **`0`** (zero).
20. Make sure that the checkbox ☑ "Notify DHCP server of change" is checked.
21. Left click on OK.
    *From here to the end of the lesson, we will insert the **MyDHCPvalue** option into the sun100 macro.*
22. Left click on the Macros tab.
23. Highlight the Sun100 macro (or whatever macro has the same name as your system's hostname), as shown below.



24. Left click on Edit
25. Left click on Properties.
    The Macro Properties window should now appear, as shown in Figure 31.7.

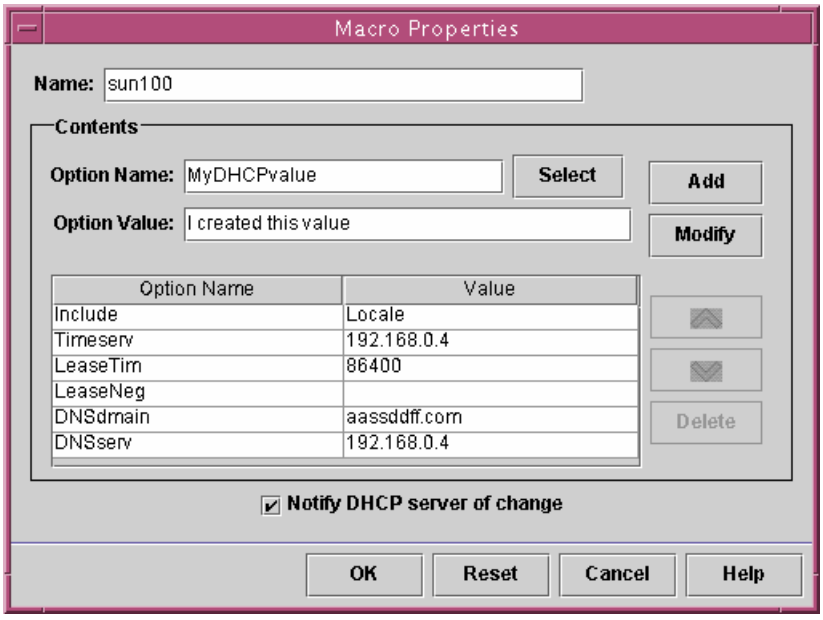


Figure 31. 7 Macro Properties Window

26. Left click on the Select button next to the Option Name: text box.
    The Select Option popup window should appear.
27. Left click on Category: at the top of the window.
28. Select Site: as the category.
    *The **MyDHCPvalue** option should appear.*

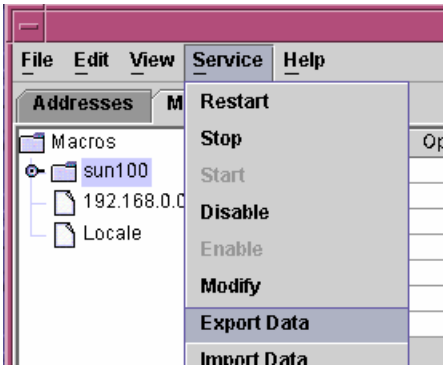
29. Highlight the ***MyDHCPvalue*** *option*.
30. Left click on OK.
    The Macro Properties window should appear again
31. In the Option Value: text box, type **`I created this value`**
    The Macro Properties window should now look like Figure 31.7.
32. Left click on Add.
33. Left click on OK to close the window.
    The macro should now have the custom DHCP Option "MyDHCPvalue"

Importing and Exporting DHCP Configuration Information

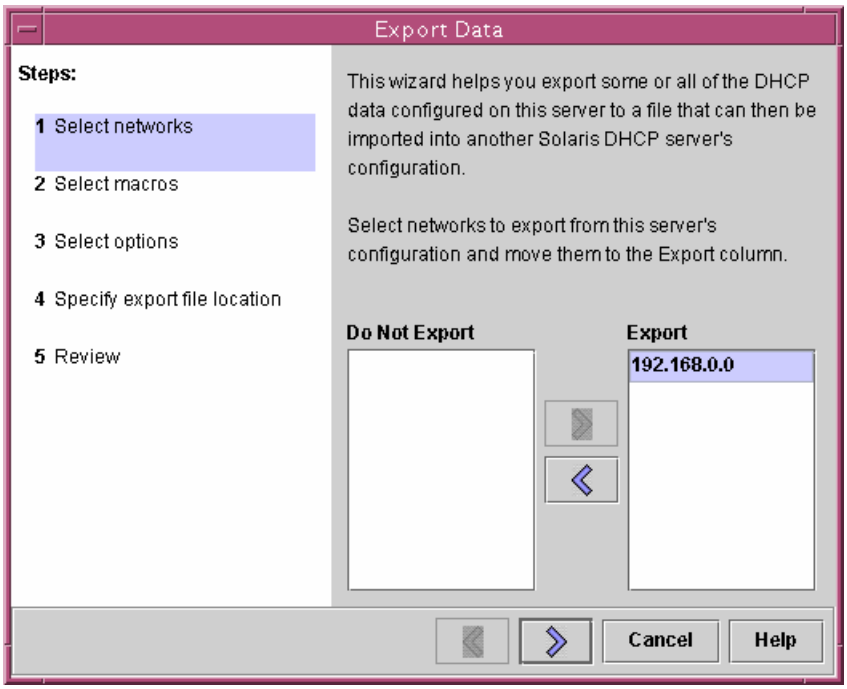
The DHCP manager has a feature that lets you export a DHCP server's settings to a file. That file can then be used to help set up a new DHCP server. This is a nice option if a DHCP server is being replaced by another one (such as a new and more powerful server). The old DHCP server's configuration file can be copied to the new server through a network connection. When the old server is taken off line, the new server can be started with the exact same configuration. This provides for seamless migration from one DHCP server to another one.

Lesson 31.5 Exporting DHCP Server Information

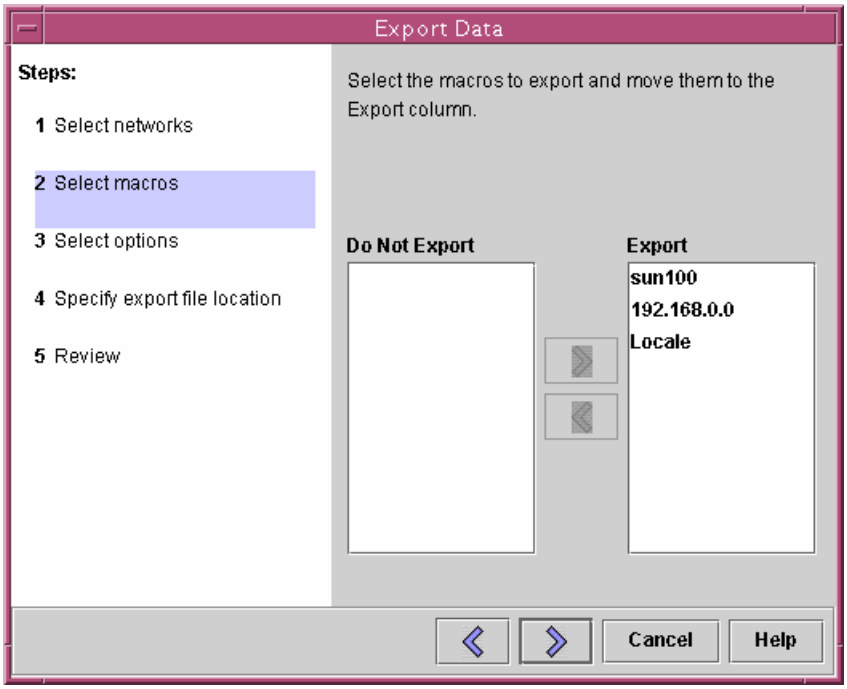
In this lesson, readers will save the configuration of the DHCP server to a file named **`myexportfile`** in the root (**`/`**) directory. This file could then be used to set up another DHCP server identically to this one.

1. Log in as the root user.
2. Open a Terminal window.
3. Start the DHCP manager with the command:
   `/usr/sadm/admin/bin/dhcpmgr &`
   In the first part of this lesson, we will export data from a DHCP server, using the Export Data wizard.
4. Left click on Service and then left click on Export Data (as shown below).
5. 

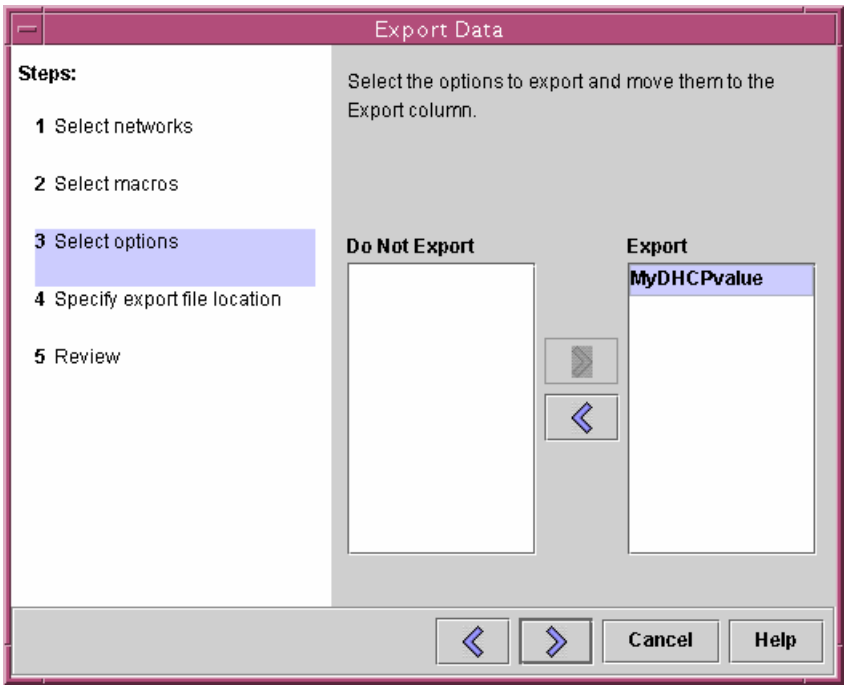
   The first screen asks what network should be exported. If you have followed the lessons in this book, there will be only one choice: 192.168.0.0.
6. In the Do Not Export field, highlight the 192.168.0.0 entry.
7. Left click on the ➢ button to the right of the Do Not Export field, to move that entry to the Export field.
8. Left click on the ➢ button at the bottom of the screen.



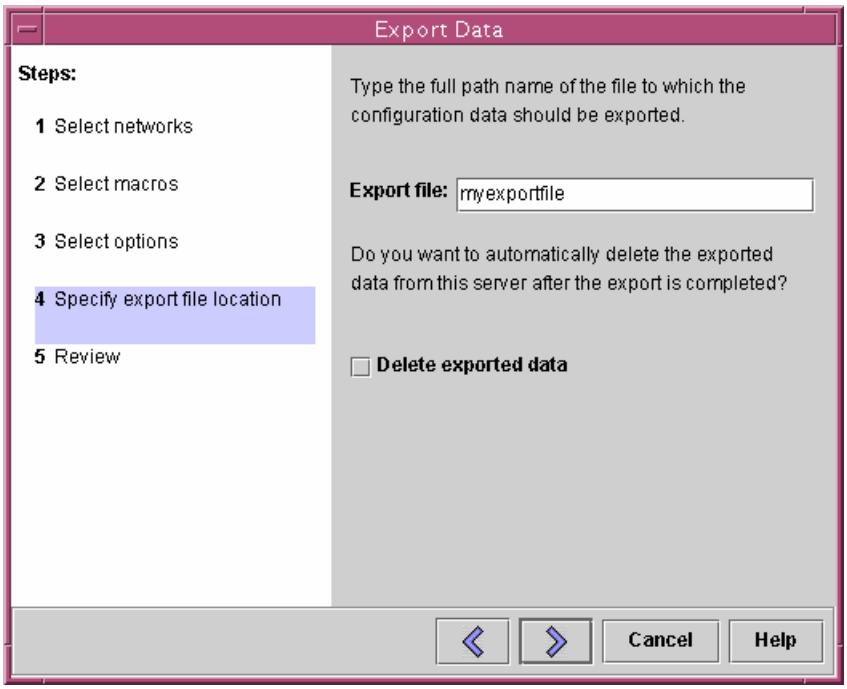
The next step lets you choose which macros to export. Remember, a macro contains a particular set of configured options.



9. In the Do Not Export field, highlight all the macros you want to export.
10. Left click on the ➢ button next to that field, to move those macros to the Export field.
11. Left click on the ➢ button at the bottom of the screen.
    The next window lets you select which options should be exported.



12. In the Do Not Export field, select the MYDHCPvalue.
13. Left click on the ➢ button next to the Do Not Export field to move it into the Export field.
14. Left click on the ➢ button at the bottom of the screen.
    The fourth window asks for the name of an export file.



15. In the Export File: text box, type **`myexportfile`**.
16. Make sure that the ☑ "Delete exported data" checkbox is checked.

The final screen is a confirmation screen.

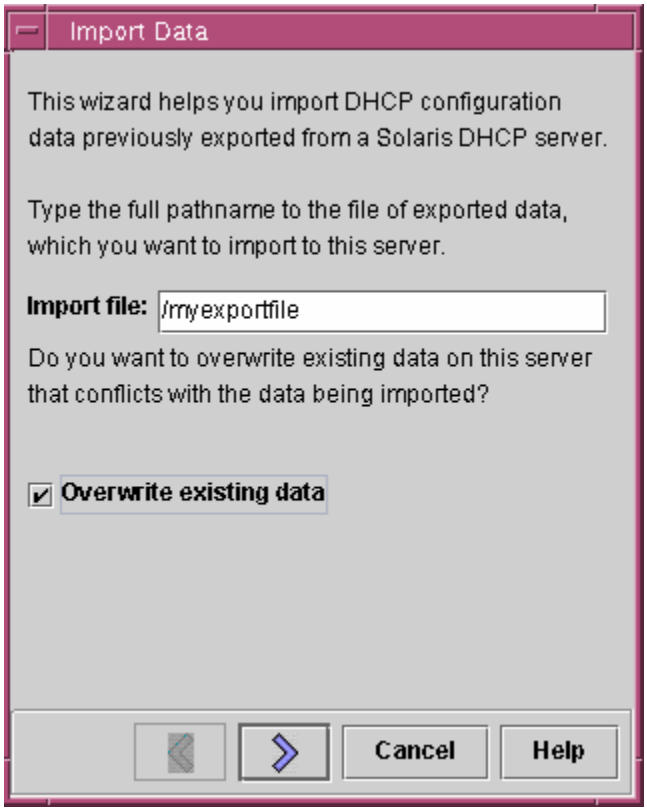
17. Left click on the Finish button.
18. Left click on File and then on Exit to exit

Lesson 31.6 Importing DHCP Server Information

As mentioned before Lesson 31.5, after you have exported a DHCP server's settings its settings to a file, The that server's configuration file can be copied to a new server, usually through the a network connection. This provides for seamless migration from one DHCP server to a new DHCP another one.

This lesson shows how to read in (import) the exported configuration information. Because you will be reading it back in to the same server from which you exported it, no changes will actually be made. So, this lesson is optional. It is a good idea to get used to using the import function, anyway.

1. Log in as the root user.
2. Open a Terminal window
3. Start the DHCP manager with the command:
   `/usr/sadm/admin/bin/dhcpmgr &`
4. Left click on Service in the Menu bar.
5. Left click on Import Data.



6. Type in the location **`/myexportfile`**
7. Make sure to select the ☑ "Overwrite existing data" checkbox
8. Left click on the ➢button.
9. Left click on the Finish button.

10. Left click on Service.
11. Left click on Stop.
12. Left click on Service.
13. Left click on Start.
14. Left click on File and then on Exit to exit the DCHP server.

Command Line Tools to Manage a DHCP Server

The DHCP manager is an easy tool to use when a graphical environment is available. Unfortunately, there are plenty of situations where a company does not install the CDE or OpenWindows environment on a server. In this case, the only way to manage a DHCP server is with command line tools.

DHCP commands are also used in scripts set up by the system administrator. For example, an administrator might create scripts that check the availability of IP addresses and then dynamically add or remove addresses from a DHCP server. By executing DHCP command line utilities, a program or script, can set up a very powerful, custom made DHCP server, something that can not be done with a GUI interface like the DHCP Manager.

There are three main command line tools used to manage a DHCP server:

- The **`dhcpconfig`** command is used to start, stop, enable and disable the DHCP server. It can also be used to export the DHCP server's data to a DHCP configuration file. In addition, this command can completely remove DHCP from a Solaris 9 server.

- The **`dhtadn`** command is used to configure macros and options. This command is very cryptic and difficult to learn.

- The **`pntadm`** command is used for day to day maintenance. It is mostly used to set up networks and IP addresses with DHCP.

The `dhcpconfig` Command

The **`dhcpconfig`** command is the primary command used to set up a DHCP server. This tool basically performs the same configuration and policy setup functions as the setup wizards in the DHCP Manager.

The following options are supported by the **`dhcpconfig`** command:

`dhcpconfig`

`-U` Un-configures a DHCP server. The following secondary options are supported:
- **`-f`** Do not display the "are you sure?" confirmation message
- **`-h`** Removes all the host names
- **`-x`** Removes the network tables and the **`dhcptab`** file

`-D` Specifies various parameters:
- **`-a`** IP addresses of one or more DNS servers, separated by the comma (**`,`**) symbol
- **`-h`** Where to place host data
- **`-l`** Lease time in seconds
- **`-n`** Non-negotiable leases for DHCP information
- **`-u`** Data ignored by **`dhcpconfig`** that is placed in the DHCP database
- **`-y`** NIS+ or DNS domain; only works if DNS or NIS+ is specified

-I Imports DHCP server configuration information from an export file.

-X Exports DHCP server configuration information to an export file.
- **-a** List of network and addresses to export; ALL is a recognized keyword
- **-m** List of macros to export; ALL is a recognized keyword
- **-o** List of options to export; ALL is a recognized keyword
- **-x** Clears out the DHCP server after the export file is created

The `dhtadm` command

The **`dhtadm`** command is used to configure macros and symbols in the DHCP table.

`dhtadm`

- **-A** Adds a macro or symbol to the DHCP table.
  - **-d** Defines a macro or symbol
  - **-m** Designates a macro name, must be used with **-d**
  - **-s** Designates a symbol name, must be used with **-d**
- **-B** Batch process of **`dhtadm`** commands from a text file or STDIN.
  - **-v** Shows verbose output as the batch file is being read
- **-C** Creates the **`dhcptab`** table.
- **-D** Deletes the **`dhcptab`** table or a **`dhcptab`** table entry.
  - **-m** Designates a macro name
  - **-s** Designates a symbol name
- **-M** Modifies existing macros or symbols.
  - **-d** Defines a macro or symbol
  - **-e** Edits a macro or symbol
  - **-m** Designates a macro
  - **-n** Renames a macro or symbol
- **-P** Prints the **`dhcptab`** table.
  - **-r** Specifies a different resource than the one specified in the **`/etc/init/dhcpsvc.conf`** variable **RESOURCE**.
- **-R** Removes the **`dhcptab`** table

The `pntadm` Command

The **`pntadm`** command works with networks and IP addresses.

`pntadm` This command manages the DHCP network tables. Common functions include showing IP addresses used, assigning new IP address pools and removing IP addresses.

- **-A** Adds a hostname and IP address.
  - **-c** Comment
  - **-e** Expiration date of lease, in MM/DD/YYYY format (rarely used)
  - **-f** Flag value, can use keywords or numbers

	DYNAMIC	00	Server's default assignment
	PERMANENT	01	Lease never ends (good for routers)
	MANUAL	02	Administrator specifies this
	UNUSABLE	04	Not a valid IP address (blocks IP address from being used; servers usually have this IP)
	BOOTP	08	Used with diskless clients and JumpStart installations

-h Hostname
-i Client identifier (hex)
-m Macro assigned to this client
-s DHCP server or IP name

-B Batch process of **dhtadm** commands from a text file or STDIN.
-v Shows verbose output as the batch file is being read

-D Deletes the hostname or IP address.
-y Deletes the hostname from the name service

-L Lists the DHCP table of network addresses.

-M Modifies the hostname or IP address.
-c Comment
-e Expiration date of lease, in MM/DD/YYYY format (rarely used)
-f Flag value, can use keywords or numbers

DYNAMIC	00	Server's default assignment
PERMANENT	01	Lease never ends (good for routers)
MANUAL	02	Administrator specifies this
UNUSABLE	04	Not a valid IP address (blocks IP address from being used; servers usually have this IP)
BOOTP	08	Used with diskless clients and JumpStart installations

-h Hostname
-i Client identifier (Hex)
-m Macro assigned to this client
-s DHCP server or IP name

-p Sets the location of the path to the DHCP database source. This is used instead of the PATH variable specified in the **/etc/init/dhcpsvc.conf** file.

-P Prints the IP addresses assigned and used in a **dhcptab** table.

-R Removes the network from DHCP.

-r Specifies the source for the data. This is the same as the **RESOURCE** variable used in the **/etc/init/dhcpsvc.conf** file.

-u Sends the data to a public module and tells **pntadm** to ignore the data.

Lesson 31.7 Using the pntadm Command

In this lesson readers will make a fictitious network inside the DHCP server. Several IP addresses will be added to the fictitious network. You can then play around with the fictitious network before deleting it from the server.

1. Log in as the root user.
2. Open a Terminal window.
3. Type the command **pntadm -P 192.168.0.0** (or any other valid network number).
   *The command **pntadm -P** <**network_number**> shows all the IP addresses that have been assigned to that network.*
4. Type the command **pntadm -C 100.100.0.0**
   This command creates a fictitious network segment with the IP address 100.100.0.0.
5. Type the command
   pntadm -r SUNWfiles -p /var/dhcp -A 100.100.0.101 100.100.0.102
   *This command uses **-r** to override the variable **RESOURCE=** from **dhcpsvc.conf** file. It then uses **-p** to display the **dhcptab** table, and **-A** to add two clients to the DHCP server.*
6. Type the command **pntadm -P 100.100.0.0**
   This shows all the IP addresses that have been assigned to that network.
7. Type the command **pntadm -A 100.100.0.102 -c "This is good" 100.100.0.0**
   *The command **pntadm -A** is somewhat tricky. Immediately after the **100.100.0.102** (new IP address) is one of the subarguments to **-A** (which in this case is **-c "comment"**). The last argument is the network that this IP address will be applied to, which in this case is **100.100.0.0**.*

8. Type the command **pntadm -P 100.100.0.0**
9. *Note the change in this output from before.*
10. Type the command **pntadm -M 100.100.0.1 -c "This is bad" 100.100.0.0**
    *The command **pntadm -M** modifies the comment on the **100.100.0.1** IP address to say "This is bad" instead of "This is good."*
11. Type the command **pntadm -P 100.100.0.0**
    Note the change in this output from before.

Lesson 31.8 Starting and Stopping the DHCP Server From the Command Line

This is a rather uneventful lesson. The DHCP server starts and stops from the command line. This is useful if the DHCP server is modified but for some unknown reasons the new changes do not seem to take effect.

1. Log in as the root user.
2. Open a Terminal window.
3. Type the command **/etc/init.d/dhcp stop**
4. Type the command **/etc/init.d/dhcp start**

Clearing a DHCP Server

Sometimes, it is useful to "clear" a DHCP server. When this is done, the server's configuration information is removed. The server will not be operational until a configuration is restored. Usually, the new configuration includes changes from the previous configuration.

When a DHCP server is cleared, its DHCP clients still keep their current network settings, until their leases expire. After a client's lease expires, the client will attempt to contact the previous server. If the DHCP server that initially set up the client can not be found, the client will try to contact other DHCP servers for DHCP information. Eventually, if no DHCP server can be found, the network will come to a grinding halt. Of course, you should avoid this situation.

When a DHCP server is cleared, the DHCP software is not removed from the system. Remember, though, that as long as the server has no configuration information, it will not try to answer DHCP requests. When the configuration information is restored, the server will be operational again.

Lesson 31.9 Using `dhcpconfig` to Clear and Restore a DHCP Server

In this lesson, you will save the configuration of the DHCP server to an export file. Then, you will stop the DHCP server and clear its configuration. Finally, you will stop the DHCP server, restart it, and restore its configuration from the saved export file.

Note: This lesson requires that you have a running DHCP server. If this is not the case, follow Lesson 31.2 to create a simple DHCP server. Then continue with this lesson.

1. Log in as the root user.
2. Open a Terminal window.
3. Type the command
   `dhcpconfig -X /myexportfile2 -a ALL -m ALL -o ALL`
   This command saves the DHCP server configuration to the export file **`/myexportfile2`**.
   The keyword ***`ALL`*** *is case sensitive, so make sure that you type it in all caps.*
4. Type the command **`dhcpconfig -U -h -x`**
   The options ***`-U -h -x`*** *are used to remove the hostnames from the DHCP server, from the DHCP network tables, and from the* ***`dhcptab`*** *file. At this point in time the DHCP server has been removed from the system. The software still exists, but the server has no configuration.*
5. Type the command **`dhcpconfig -D -r SUNWfiles -p /var/DHCP`**
   This command adds configuration information to the DHCP server.
6. Type the command **`dhcpconfig -I /myexportfile2`**
   This command restores the original DHCP server to its original condition by reading in the configuration information saved in ***`/myexportfile2`***..

Lesson 31.10 Using Command Line Utilities with a Solaris 9 DHCP Client

This lesson introduces readers to various Solaris 9 DHCP command line utilities that provide information about your DHCP setup. These utilities must be run on a DHCP client. Remember, a DHCP server can not also be a DHCP client. If you only have one Solaris 9 test system to work with, you can create a DHCP server on another system (such as Linux, FreeBSD or Windows 2000) and then configure Solaris 9 to be a DHCP client.

1. Log in as the root user.
2. Open a Terminal window.
3. Type the command **`ls /etc/dhcp*`**
   This command shows all files that start with the first four letters ***`dhcp`*** *. In this case there needs to be a file named* ***`/etc/dhcp.network_card`*** *for each Ethernet card used with DHCP. If the file* ***`/etc/dhcp.network_card`*** *does not exist, then DHCP is not running on this client.*
4. Type the command **`ifconfig -a`**
   This command shows the current configuration of the network card on the current workstation.
5. Type the command **`cat /etc/dhcp/if.dhc`**
   This shows the current DHCP configuration of the network card.
6. Type the command **`cat /etc/default/dhcpagent`**
   This shows the tunable parameters of DHCP. Do not change these by hand unless there is a very strong reason for doing so!
7. Type the command ***`dhcpinfo -c 3`***
   The identifier ***3*** *is for a router.*
8. Type the command **`dhcpinfo -c Ip`**

*The symbol **Ip** represents the IP address assigned to the system.*

9. Type the command **`dhcpinfo -c 72`**
   *The identifier **`72`** is used to identify a World Wide Web server. This might not be a DHCP parameter that has been given by the DHCP server*

Lesson 31.11 Working with the dhtadm Command

In this lesson, you will use the **`dhtadm`** command to change the **`dhcptab`** table from the command line. You will then print out the **`dhcptab`** table, add an entry to it, and then delete the entry from this table.

1. Login as the root user.
2. Open a Terminal window
3. Type the command **`/usr/sbin/dhtadm -P`**
   *This command prints the **`dhcptab`** table. This will look something like:*

```
Name            Type        Value
================================================
192.168.0.0     Macro       Subnet=255.255.255.0:Router=192.168.0.1
                             :Broadcst=192.168.0.255:
sun100          Macro       :Include=Locale:Timeserv=192.168.0.4:
                            LeaseTim=86400:LeaseNeg:
Locale          Macro       :UTCoffst=-25200:
```

   This command shows the following pieces of information about the DHCP table:

   Name — *Name of the symbol record.*
   Type — *Type of record being displayed.*
   Value — *Variables used with this record. In the above example, some of these include the DHCP subnet value, the router and the broadcast address. To obtain further information on DHCP tables, type the command **`man -s4 dhcptab`**. This is the man page on the DHCP table.*

4. Type the command **`dhtadm -A -m newdns -d ':DNSserv=192.168.0.5:'`**
   *This command adds a new macro to the **`dhcptab`** table with the name **newdns** that contains an entry for a DNS server.*
5. Type the command **`dhtadm -P`**
   *This command prints out the **`dhcptab`** table. Notice that this time there is a new macro with the name **newdns** that contains the value* **DNSserv=192.168.0.4**, *as shown below:*

```
                          Name  Type     Value
================================================
newdns          Macro       :DNSserv=192.168.0.4:
192.168.0.0     Macro       :Subnet=255.255.255.0:Router=192.168.0.1
                            :Broadcst=192.168.0.255:
sun100          Macro       :Include=Locale:Timeserv=192.168.0.4:
                            LeaseTim=86400:LeaseNeg:
Locale          Macro       :UTCoffst=-25200:
```

6. Type the command **`dhtadm -M -m newdns -d ':DNSserv=192.168.0.7:'`**
   *This command modifies the **newdns** macro. The variable **DNSserv** was changed from 192.168.0.4 to 192.168.0.7.*
7. Type the command **`dhtadm -P`**
   *This command prints the **`dhcptab`** table. The table should look the same as above, except that the macro named **newdns** has been modified.*

```
newdns          Macro       :DNSserv=192.168.0.7:
```

8. Type the command **`dhtadm -D -m newdns`**
   *This command deletes the **newdns** macro from the **`dhcptab`** table.*
9. Type the command **`dhtadm -P`**
   *This command prints the **`dhcptab`** table. Notice that the **newdns** macro is gone?*

Client Side DHCP Files

These files reside on the Solaris 9 DHCP client. They are created automatically when a Solaris 9 workstation becomes a DHCP client. These files contain the DHCP settings that the client has obtained from the DHCP server. If you believe that for some reason the DHCP settings are incorrect, examine your DHCP server. Do not automatically assume that the DHCP client is the problem.

`/etc/dhcp/if.dhc`
: This file shows the configuration of a DHCP interface on the client.

`/etc/default/dhcpagent`
: This file shows the tunable parameters for the DHCP protocol. Unless there is a severe problem with DHCP, these values should not be changed.

Client and Server Side DHCP Commands and Daemons

These commands and daemons reside on both the Solaris 9 DHCP client and server.

`in.dhcpd`
: This is the actual DHCP daemon. If clients can not receive DHCP information, type the command
  `ps -ef | grep in.dhcpd`
  to see if this daemon is running.

  If the daemon is not running, use the DHCP Manager to start the DHCP server.

`dhcpagent`
: This resides on the DHCP client. When Solaris 9 starts, the **`/sbin/ifconfig`** command looks for a file named **`/etc/dhcp.<network_adapter>`**. If this file exists, the **`dhcpagent`** contacts the DHCP server for the network configuration. The **`dhcpagent`** also obtains a lease time from the DHCP server. If the lease expires, the **`dhcpagent`** obtains an extension to the lease or new network parameters. If a new lease can not be obtained, the network interface is disabled.

`dhcpinfo`
: The **`dhcpinfo`** command is used to show the DHCP values that were given to the DHCP client by the DHCP server.

The /etc/init.d/dhcp Script

The **`/etc/init.d/dhcp`** command is called by the run control scripts when the Solaris 9 operating system starts. The command **`/etc/init.d/dhcp start`** begins the **`in.dhpcd`** daemon.

Key Points to Remember

DHCP is a very handy tool to have when dealing with a large scale network. The DHCP server can be set up to automatically assign IP addresses to a wide variety of networked devices. The Solaris DHCP server has a very nice GUI interface that lets the system administrator easily set up the DHCP server. Make sure to also practice using the DHCP command line tools, so that if you come across a Solaris server that does not have a GUI interface, you will still be able to save and modify the DHCP server configuration.

“

1

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V