AIX Operating System
for the PS/2 and System/370
Using the Operating System

Document Number SC23-2291-01

AIX Operating System
for the PS/2 and System/370

Using the Operating System

Document Number SC23-2291-01

Edition Notice

Third Edition (March 1991)

This edition applies to Version 1.2.1 of the IBM Advanced Interactive Executive for the System/370 (AIX/370), Program Number 5713-AFL, and for Version 1.2.1 of the IBM Advanced Interactive Executive for the Personal System/2 (AIX PS/2), Program Number 5713-AEQ, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 52QA MS 911
Neighborhood Road
Kingston, NY 12401
U.S.A.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Subtopics

Trademarks and Acknowledgments

Trademarks and Acknowledgments

The following trademarks apply to this book:

AIX is a registered trademark of International Business Machine Corporation.

IBM is a registered trademark of International Business Machine Corporation.

Portions of the code and documentation were developed at th Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

RT,Personal System/2 and PS/2 are registered trademarks o International Business Machines Corporation.

System/370 is a trademark of International Business Machine Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc. i the USA and other countries.

Ethernet is a trademark of Xerox, Inc

Personal Computer XT is a trademark of International Business Machine Corporation.

Personal Computer AT is a registered trademark of Internationa Business Machines Corporation.

DEC, VT100 and VT220 are trademarks of Digital Equipment Corporation

INed is a trademark of INTERACTIVE Systems Corporation

About This Book

This book is written for those who want to learn how to use the Advanced Interactive Executive (AIX) Operating System for the IBM PS/2 and IBM System/370. The AIX Operating System is based on UNIX System V, and it includes many features from another popular form of UNIX, the Berkeley Software Definition (BSD4.3).

If you have never used the AIX Operating System before, the beginning chapters teach the basic commands to get you started. If you are an intermediate or advanced user, this book provides an introduction to some of the more sophisticated features and commands.

This book includes the general information you need to be able to:

Start the AIX Operating System and use simple command

Display and print the contents of file

Use the file syste

Work with processe

Write shell program

Use internal and external communication facilities

Subtopics

Who Should Read This Book

If you are not familiar with the AIX Operating System, you can use this book as a training manual. If you are familiar with the AIX Operating System, you can use this book as a reference. The table of contents and index are useful for locating particular topics to review.

What You Should Know

Your AIX system is managed by a ***system administrator***. Depending on your organization, the system administrator may be responsible for hardware maintenance, software administration, and training. Your system administrator should provide you with a **user name** and possibly a password. You should know what type of terminal or work station you are using to run the AIX Operating System, and you should be familiar with any information specific to that machine, such as a keyboard diagram.

How to Use This Book

Each chapter in this book takes the same general approach to the topics it covers--a series of explanations and examples. The examples build upon each other; in many instances, an example uses a file created in a previous example. Therefore, if you intend to follow the examples on your terminal, it is important for you to work through each chapter from start to finish beginning with Chapter 1.

Appendix A, "Using Advanced Bourne Shell Features" at the end of the book provides information intended for the more knowledgeable user.

In the text, whenever you are told to ***enter*** a command or other information, you should type the information and then press the **Enter** key.

Subtopics
Highlighting
Quick Reference Boxes
For Japanese Locale Users

Highlighting

This book uses different type styles to identify certain kinds of information. General information is printed in the standard type style (the type style used for this sentence). Other type styles alert the reader to special information.

New terms
Each time a new term is introduced, its first occurrence is printed in this type style (for example, "the AIX Operating System ***file system***").

System parts
The names for keys, commands, files, file names, and other parts of the system are printed in this type style (for example, "the **cp** command").

Variable information
The names for information that you must provide are printed in this type style (for example, "type **yourname**").

Special characters
Any characters that have a special meaning are printed in this type style (for example, "the **&** and **&&** operators have different uses"). This type is also used for the names of files that you create as you work through this book (for example, "create a file named **afile**").

Information you are to type
Many examples in this book are designed for you to try them on your own terminal; the information that you should type is printed in this type style (for example, "type **ls text** and press **Enter**").

Quick Reference Boxes

Where appropriate, the chapters and major sections of this book begin with a box containing instructions necessary to perform a particular task. The boxed instructions are similar to this:

To Use a Quick Reference Box

1. Skip the quick reference boxes the first time you read a section.
2. Use the quick reference boxes as a fast path through the book.
3. Refer to the quick reference boxes to refresh your memory.

The boxes make a convenient ***fast path*** through the book, but they are not comprehensive and are not intended to take the placeof the explanatory material in each section. Use the boxes for reference after you are familiar with the material.

For Japanese Locale Users

If you are using a Japanese keyboard, it is recommended that you go through each exercise twice. The first time you do an exercise, enter all your responses in ASCII; then, go through the same exercise entering your responses in Japanese. This procedure has two benefits. First, it lets you try everything twice. This can reinforce the lessons being learned. Second, it shows you the results of each command in both ASCII characters and the various forms of Japanese script which may appear on the screen.

US English is the default mode of the AIX system. There may be times when, due to error conditions, the operating system produces its results in this mode. Even if you do not read the English language responses that the system may make from time to time, it is beneficial for you to become familiar enough with them to have some idea of what the system is doing.

Related Publications

For additional information, you may want to refer to the following publications:

AIX Access for DOS Users Administrator's Guide, SC23-2042, explains how to install and administer the AIX Access for DOS Users program on the IBM PS/2, RT, and System/370 computers running the AIX Operating System with the AIX DOS Server. It covers the responsibilities for installation, daily operation, and maintenance of the AIX Access program.

AIX Access for DOS Users User's Guide, SC23-2041, describes the AIX Access for DOS Users program and shows how to use the file services of an AIX host while running DOS applications.

AIX C Language Reference, SC23-2058, describes the C programming language and contains reference information for writing programs in C language that run on the AIX Operating System.

AIX C Language User's Guide, SC23-2057, describes how to develop, link, and execute C language programs. This book also describes the operating dependencies of C language and shows how to use C language-related software utilities and other program development tools.

AIX Commands Reference, SC23-2292 (Vol. 1) and SC23-2184 (Vol. 2), lists and describes the AIX/370 and AIX PS/2 Operating System commands.

AIX Guide to Multibyte Character Set (MBCS) Support, GC23-2333, explains the basic concepts of AIX multibyte character set support and refers to other AIX books that contain more detailed information.

AIX Japanese Support User's Guide, SC18-0834, describes the AIX Operating System for Japanese users.

AIX Library Guide, Glossary, and Master Index, SC23-2324, describes the publications in the AIX Operating System library and contains a glossary of terms used throughout the library. This book also includes a master index to the contents of each of the publications in the library.

AIX Messages Reference, SC23-2294, lists messages displayed by the AIX Operating System and explains how to respond to them.

AIX Programming Tools and Interfaces, SC23-2304, describes the programming environment of the AIX Operating System and includes information about operating system tools that are used to develop, compile, and debug programs.

AIX TCP/IP User's Guide, SC23-2309, describes the features of TCP/IP and shows how to install and customize the program. It includes reference information on TCP/IP commands that are used to transfer files, manage the network, and log into remote systems.

AIX Technical Reference, SC23-2300 (Vol. 1) and SC23-2301 (Vol. 2), describes the system calls and subroutines a programmer uses to write application programs. This book also provides information about the AIX Operating System file system, special files, miscellaneous files,

and the writing of device drivers.

AIX VS FORTRAN Reference, SC23-2050, describes the FORTRAN programming language as implemented on AIX RT, AIX PS/2, and AIX/370. This book describes all of the standard features of VS FORTRAN as well as the enhanced functions and capabilities incorporated into IBM AIX VS FORTRAN.

AIX VS FORTRAN User's Guide, SC23-2049, shows how to develop and execute FORTRAN programs on AIX RT, AIX PS/2, and AIX/370. This book also explains how to compile and execute programs that contain sections of code written in the VS Pascal and C programming languages.

AIX VS Pascal Reference, SC23-2054, describes the VS Pascal programming language as implemented on the IBM PS/2 or RT with the AIX Operating System installed. This book describes all of the standard features of Pascal as well as the enhanced functions and capabilities incorporated into IBM AIX VS Pascal.

AIX VS Pascal User's Guide, SC23-2053, shows how to develop and execute Pascal programs on the IBM PS/2 and RT using the AIX Operating System. This book also explains how to compile and execute programs that contain sections of code written in the VS FORTRAN and C programming languages.

AIX X-Windows Programmer's Reference, SC23-2118, describes the X-Windows licensed program and provides information on X-Windows library functions, FORTRAN subroutines, protocols, and extensions.

AIX X-Windows User's Guide, SC23-2017, describes the X-Windows licensed program and shows how to start, run, install, and customize this program.

AIX PS/2 DOS Merge User's and Administrator's Guide, SC23-2045, shows how to use DOS in the AIX environment, including running DOS and AIX programs simultaneously and running AIX commands from the DOS environment. It also shows how to install the DOS Merge software and how to perform essential system maintenance activities, such as adding user accounts, backing up the file system, and setting up terminals.

AIX PS/2 General Information, GC23-2055, describes the AIX PS/2 Operating System's functions and capabilities and the product's position in the AIX family of products.

AIX PS/2 INed, SC23-2001, shows how to use the INed editor to create, access, and store files. This book also includes reference information on INed commands and a listing of INed error messages.

AIX PS/2 INmail/INnet/INftp User's Guide, SC23-2076, describes the INmail/INnet/INftp/Connect programs and shows how to use these programs to send mail to and receive mail from local and remote computer systems. This book also shows how to transfer files to and from other computer systems installed on the network.

AIX PS/2 Interface Library Reference, SC23-2051, contains information about the library of system calls available with IBM AIX VS Pascal and IBM AIX VS FORTRAN as implemented for use with the IBM AIX PS/2 Operating System.

AIX PS/2 Keyboard Description and Character Reference, SC23-2037,

describes the characters and keyboards supported by the AIX PS/2 Operating System. This book also provides information on keyboard position codes, keyboard states, control code points, code-sequence processing, and non-spacing character sequences.

AIX PS/2 Text Formatting Guide, SC23-2044, describes the text formatting utilities available on the PS/2 and shows how to format text with NROFF and TROFF. This book also shows how to use the **vi** editor to create, revise, and store files.

AIX PS/2 Usability Services Reference, SC23-2039, lists and describes Usability Services commands.

AIX PS/2 Usability Services User's Guide, SC23-2038, shows how to create and print text files, work with directories, start application programs, and do other basic tasks with Usability Services.

AIX/370 Administration Guide, SC23-2088, describes such administrative tasks as updating the file system, backing up files, and fine-tuning and monitoring the performance of the operating system.

AIX/370 Diagnosis Guide, SC23-2090, describes procedures and tools that can be used to define and categorize symptoms of problems that may occur during daily operation.

AIX/370 General Information, GC23-2062, describes the functions and capabilities of AIX/370 and its position in the AIX family of products.

AIX/370 Planning Guide, GC23-2065, describes the functions and capabilities of the AIX/370 Operating System and lists the requirements for all supported hardware and software. This book also includes information to assist with planning for installation and customization of the operating system.

Installing and Customizing the AIX PS/2 Operating System, SC23-2290, provides step-by-step instructions for installing the AIX PS/2 Operating System and related programs. This book also shows how to customize the operating system to suit the user's specific needs and work environment.

Installing and Customizing the AIX/370 Operating System, SC23-2066, provides step-by-step instructions for installing the AIX/370 Operating System and related programs. This book also shows how to customize the operating system to suit the user's specific needs and work environment.

Managing the AIX Operating System, SC23-2293, describes such system-management tasks as adding and deleting user IDs, creating and mounting file systems, backing up the system, repairing file system damage, and setting up an electronic mail system and other networking facilities.

Using the AIX Operating System, SC23-2291, shows the beginning user how to use AIX Operating System commands to do such basic tasks as log in and out of the system, display and print files, and set and change passwords. It includes information for intermediate to advanced users about how to use communication and networking facilities and write shell procedures.

Table of Contents

Figures

Tables

1.0 Chapter 1. Getting Started on the AIX Operating System

Subtopics

1.1 CONTENTS

1.2 About This Chapter

This chapter introduces you to the basic tasks of using the AIX Operating System. Before you work through this book, the components of your system must be set up and the AIX Operating System must be installed. In addition, you may be required to have a user name and a password. See your system administrator for this information.

After you finish this chapter, you should next learn how to create and modify files with a ***text editing program***. (A ***file*** is simply a collection of data stored together under a given name.) The following editing programs are available on the AIX Operating System:

ed (see Appendix B, "Creating and Editing Files with ed" in topic B.0)

INed (see *AIX INed*)

vi (see *AIX Text Formatting Guide*).

Your system may have other editing programs as well.

Once you complete this chapter and learn how to use an editing program, you should have the basic skills necessary to start using the operating system.

1.3 Display Stations--Terms and Features

One of the AIX Operating System's strongest points is its ability to allow many different types of hardware to interact smoothly together. Consequently, it is difficult to predict what hardware you will be using as you learn the AIX Operating System.

The following terms appear frequently throughout this manual:

terminal The word ***terminal*** is often used to describe any input/output device equipped with a keyboard and a video screen. In this manual, ***terminal*** refers to a dumb terminal or ASCII terminal. This is a screen and keyboard unit that does little or no internal processing. It merely receives your input from the keyboard and passes it on to the host computer to which you are attached. Likewise, a terminal receives the ***host*** responses and passes them unchanged to your screen. For information about hosts, see Chapter 5, "TCF Overview."

work station A ***work station*** is sometimes called an "intelligent terminal". A work station is a single-user, high-performance microcomputer (or even a minicomputer) which has been specialized in some way, usually for graphics output. Such a machine has a screen and a keyboard, but it is also capable of extensive processing of your input before it is passed to the host. Likewise, the host's responses may be extensively processed before being passed along to your screen. A work station may be intelligent enough to do much or all of the processing itself. In this manual, work station refers to an IBM PS/2 running AIX PS/2 or an IBM RT running AIX RT.

display station A ***display station*** may be either a terminal or a work station. It refers to any physical machine where you enter your input and receive responses.

system Throughout this book ***system*** is used to refer to the entire complex of computing machinery with which a user interacts. Users of the AIX Operating System usually don't need to worry about which computer is servicing his or her requests; the operating system takes care of these details. When it is not important for you to know what hardware is servicing your needs, this book will use the term ***system***. The later chapters in this book deal with establishing connections between your computer and a ***remote system***. This may be a single computer of a group of machines tied together in a network. In this case, ***system*** refers to any distant machine(s) with which you communicate.

You may notice small differences between the descriptions of keyboard and screen given in this book and what you actually see on your own display station. The descriptions in this book are based on the console of a Personal System/2. If your own equipment is different, the differences are small and of no fundamental consequence. You can accomplish most of the tasks described in this book using the typewriter-style letter and number keys, plus a few other special keys located on the keyboard.

Subtopics
1.3.1 The Keyboard

1.3.1 The Keyboard

Figure 1-1 shows the standard IBM PS/2 Keyboard.

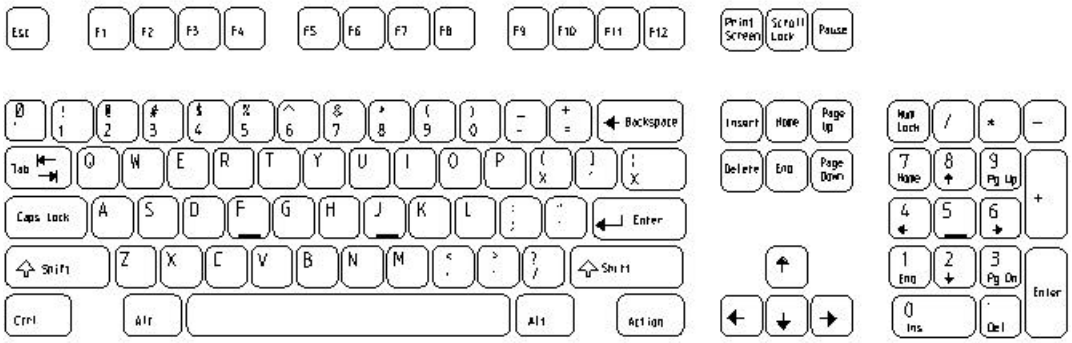


Figure 1-1. The IBM PS/2 Keyboard

Your keyboard may differ in some ways from the keyboard illustrated in Figure 1-1. Refer to the manual for your particular display station for keyboard information.

Subtopics
1.3.1.1 Special Keys
1.3.1.2 Performing Special Functions
1.3.1.3 Using Different Keyboards

1.3.1.1 Special Keys

Following is a list showing some of the special keys that you use in addition to, or in conjunction with, the regular typewriter-style keys on the keyboard. You often use these keys to perform certain specialized operations.

Key	**Action**
Ctrl	Used with other keys for special functions. **Ctrl-D**, for example, logs you off the system.
Enter	Sends your typed input from the keyboard to the system and moves the cursor from the end of one line to the beginning of the next.
Esc	Ends certain system activities. You also use this key in conjunction with other keys for special functions.
	(**Cursor Left**) Moves the cursor to the left one character at a time.
	Note: The ***cursor*** is the underscore or rectangle on your screen that moves as you type characters or press the cursor movement keys.
	(**Cursor Right**) Moves the cursor to the right one character at a time.
	(**Cursor Up**) Moves the cursor up one line at a time.
	(**Cursor Down**) Moves the cursor down one line at a time.

1.3.1.2 Performing Special Functions

To perform certain functions, you must often use two or more keys together. For example, to log out of the AIX Operating System, you can send the **END OF FILE** signal. To do so, you press and hold the **Ctrl** key and then press the **D** key.

The names of special functions are printed in this book in all uppercase letters. Following is a list of the special functions and the IBM PS/2 Keyboard keys you press to use them.

Special Function	Keys Used
END OF FILE	Ctrl-D
INTERRUPT	Del
NEXT WINDOW	Alt-Action
RESUME OUTPUT	Ctrl-Q
STOP OUTPUT	Ctrl-S
HORIZONTAL TAB	Ctrl-I
CARRIAGE RETURN	Ctrl-M (same function as **Enter**)
LINE FEED	Ctrl-J
CHARACTER KILL	Shift-#
WORD KILL	Ctrl-W
LINE KILL	Shift-@

Note: Some programs require a line feed. If you get unusual information on the screen or the system does not respond when you press **Enter**, see **stty** in *AIX Operating System Commands Reference* for information about setting the characteristics of your display station.

The keys you press to obtain special functions vary depending upon the type of terminal or computer that you are using. The **INTERRUPT** function listed earlier can be used to illustrate this variability. For example, an ASCII terminal may have a keyboard with a key named **Del**, **Delete**, or **Rubout** that you can use to interrupt a running process. If you are using a machine that is emulating an ASCII terminal when it connects to AIX, you probably use one of these keys. On a PS/2, however, you would use the **Ctrl-Backspace** combination, while on most other machines you would use **Ctrl-C** or **Ctrl-?**. This book uses the default value of the **INTERRUPT** function, which is **Del**. Your keyboard may use one of the other values.

AIX allows the system administrator to reconfigure these key sequences quite easily. Consequently, your workstation may have been reconfigured so that your keyboard functions according to a standard in use within your industry or organization. If you do not get the expected results when you press one of the function keys recommended by this book, your terminal or computer may be configured for a different key sequence. Ask your system administrator for help. If you are using a VT100 terminal or terminal emulation from a personal computer, consult your VT100 user's manual. If you are using a 3270 type display station, such as a 3278, consult the

user's manual for that display station.

Note: You can change many of the features of your PS/2. The display can show a variety of colors, line lengths and fonts. The keyboard offers a variety of delay rates and repetition rates. You can open multiple AIX sessions with a feature called Virtual Terminals. Controlling these features is explained in *Managing the AIX Operating System*.

1.3.1.3 Using Different Keyboards

You can use the AIX Operating System from any of the following display stations, each of which has a different keyboard:

The main PS/2 display station, sometimes called the ***console***, which includes the IBM PS/2 keyboard

The IBM 3151 ASCII Display Statio

The IBM 3161 ASCII Display Statio

The IBM 3163 ASCII Display Station (3161 mode

The DEC VT10

The DEC VT22

The following IBM personal computers using 3101 or VT100 emulation

- IBM Personal Computer
- IBM Personal Computer XT
- IBM Personal Computer AT
- IBM Portable Personal Computer
- IBM Convertible Personal Computer
- IBM Personal System/2 Models 50 and 60

IBM PS/5

IBM 5550

If you are using a Japanese keyboard, you may find Appendix C, "Working in a Japanese Locale" helpful. It describes the process of making text entries in ASCII, Romaji, Hiragana and Katakana modes.

Place your keyboard in ASCII mode for the first few exercises and make all responses in ASCII characters. You will learn to use the other text entry modes soon. If your keyboard is not already in ASCII mode, you can refer to Appendix C to learn how to select that mode.

1.4 Logging in to the AIX Operating System

When you start your display station, the system goes through a series of internal procedures before it is ready to use. When the system is ready, it displays a copyright notice and the following prompt:

```
login:
```

Before you can use the AIX Operating System, your display station must be running and you must be ***logged in*** (identified as a valid system user). If your system displays the prompt **login:** after going through its initial operations, log in with the procedure described in the following quick reference box. More detailed information about the **login process** follows the quick reference box.

If your system displays the prompt **autologin of name**, it logged you in automatically. You can now continue your work in this chapter at "Logging Out of the AIX Operating System" in topic 1.5.

Logging in to the AIX Operating System

If your system is not running, see your system administrator for assistance.

If your system is already running:

1. Type your user name following the **login:** prompt:

   ```
   login: username
   ```

2. Press the **Enter** key.

   If you do not have a password, you do not receive the next prompt.

3. If the **password** prompt appears, type your password, which the system does **not** display on the screen, and then press **Enter**:

   ```
   password: [your password]
   ```

When the prompt **login:** appears on your screen, type your user name and press **Enter**. This is the login process.

Note: If you are a user in a Japanese locale, your user name is always a string of ASCII characters. Be sure your keyboard is in ASCII mode when you enter it. Doublewide Roman characters look similar on the screen but the AIX system does not accept these characters as ASCII entries. If you cannot tell the difference, look at Appendix C, "Working in a Japanese Locale."

If your system displays the prompt **autologin of name**, it has logged you in automatically.

After you enter your user name, the system displays the **password:** prompt.

Enter your password. For security reasons, the system does not display your password as you type it.

Note: If you do not have a password, the system does not display the **password:** prompt. Although your system may not require you to have a password, it is usually a good idea to set one for yourself. For an explanation of how to set or change your password, see "Setting and Changing Your Password" in topic 1.7.

When the system completes the login procedure, it displays a prompt, usually a dollar sign followed by a space (**$** _), which is called the Bourne ***shell prompt***. The system is now ready to accept a command.

Note: The usual prompt is **$** (the Bourne shell prompt). Another standard prompt is **%**, the C shell (csh) prompt. However, it is possible that the prompt on your system is set to some other character or characters.

If your system does not display a prompt, you are not logged in. You may, for example, have typed your user name or your password incorrectly. Try to log in again. Remember to press the **Enter** key after you finish typing each entry. If you still cannot log in, see your system administrator.

1.5 Logging Out of the AIX Operating System

You generally log out and leave the operating system running for other users. Log out procedures may differ, depending on your configuration. If you are executing commands from a terminal, it may be shut down at any time without disrupting the operating system. If you are seated at a display station, you may or may not be able to shut your computer off without disrupting the system. Check with your system administrator.

Logging Out of AIX/370 and Cluster Systems

Press **END OF FILE (Ctrl-D)**.

Press **Enter**.

Turn off the power to your terminal or personal computer.

Logging Out on a Stand-alone AIX PS/2 System

Press **END OF FILE (Ctrl-D)**.

Press **Enter**.

If you have ***superuser*** authority (which usually means that you are a member of the group responsible for maintaining the system), you can also stop the operating system.

Warning: Shutting down the system improperly may result in a loss of data. In most cases, you should not stop the operating system unless you are a member of the system team. It is very important that the **shutdown** command be issued before turning the power off to the system.

Shutting Down the System

Type the following command, which writes (saves) anything that may still be in the buffer:

```
sync;sync;sync
```

Use the **shutdown** command to stop the operating system.

- Type **shutdown** and press the **Enter** key.

Turn off the power to the system when you see the following message:

```
....shutdown completed....
terminating processes
syncing disks
System halted, you may turn the power off now.
Type Enter to Reboot
```

For more information about **shutdown**, see *Managing the AIX Operating System*.

1.6 Using Operating System Commands

Operating system ***commands*** are programs that perform tasks. The AIX Operating System has a large set of commands, which are described in the remaining chapters of this book and in *AIX Operating System Commands Reference*.

In addition to using the commands provided with the system, you can also create your own personalized commands. Refer to "Writing and Running Shell Procedures" in topic 4.4.6 for information about creating these special commands.

When you work with the operating system, you typically enter command following the **$** (shell) prompt on the ***command line***. For example, to display a list of the contents, if any, of your current directory, enter the command **ls**:

```
$ ls
```

Remember to press the **Enter** key after you finish typing your entry.

If you make a mistake while typing a command, use the **Backspace** key to erase the incorrect characters and then retype them. The cursor movement key does not work for this purpose.

An option, called a ***flag*** in the command format, alters the way a command works. Most commands have several flags. For example, if you type the **-l** (**l**ong) flag following the **ls** command, the system provides additional information about the contents of the directory.

The following example shows how to use the **-l** flag with the **ls** command:

```
$ ls -l
```

An ***argument*** is a string of characters, usually the name of a file or directory, that follows a command name. An argument specifies what data the command is to work with. If you use flags with a command, arguments follow the flags on the command line.

In the following example, **/bin** (the name of a directory) is an argument:

```
$ ls -l /bin
```

The **ls -l /bin** command gives a long (detailed) listing of the contents of the directory **/bin**.

Note: Chapter 3, "Using the File System," contains a detailed explanation of files and directories.

If you start a command and then decide that you do not want t complete that action, press the **INTERRUPT** key sequence to cancel it.

INTERRUPT and other special function keys are described in "Performing Special Functions" in topic 1.3.1.2.

While a command is running, the system does not display the **$** (shell) prompt. When the command completes its action, the system displays the **$** prompt again, indicating that you can enter another command.

Subtopics
1.6.1 Japanese Language Support Information

1.6.1 Japanese Language Support Information

A Japanese Language user should eventually be able to do almost all his ordinary day-to-day work in Japanese characters. During the learning period, the user's responses should be made in ASCII characters.

As a general rule, command names and command flags must be entered in ASCII characters.

Your System Administrator may have set up a system of aliases which allow you to use Japanese characters to accomplish the same purpose. Standard AIX commands may have been copied to your system under new Japanese names, or even renamed in Japanese. New commands can be created with Japanese names. These are all techniques which you will learn later.

However, even if a whole system of such Japanese commands exist on your system, do not use them at this point. You need to learn to enter the standard AIX commands and command flags in the standard format required by the system.

A very extensive system of aliases can cover but a few of the functions available under AIX. Each command has many combinations of option flags and having separate aliases for all possible combinations would create a list of command names no one could ever keep track of. Furthermore, even commands which have been renamed with Japanese names still need their option flags entered in ASCII characters.

You need to learn the ASCII system first. You will never attain the full flexibility and power afforded by the AIX system if you allow yourself to become dependent upon the use of Japanese command names too early.

When you are familiar with command name and command flag functions, you can use the Japanese aliases as convenient shortcuts. You will then know enough to modify the aliases and develop new ones which reflect the way you actually use the system. This can only happen if you thoroughly learn the ASCII system first.

Note: As a general rule you may assume that anything this manual deals with can be done in Japanese characters unless specifically stated otherwise in the text.

1.7 Setting and Changing Your Password

A user name is a code that you use to identify yourself to the system. A ***password*** is a code that you use to verify your identity.

Your user name is public information and generally does not change. Your password, on the other hand, is private, and you should change it periodically with the **passwd** command to protect your data from unauthorized access.

If your account does not have a password, you can use the **passwd** command to set one.

Setting or Changing Your Password

1. Enter the following at the prompt:

   passwd

2. After the prompt **Old password:** appears, enter your old password:

   Old password: **your old password**

   Remember that the system does **not** display your password on the screen.

3. After the **New password:** prompt appears, enter your new password:

   New password: **your new password**

4. Enter your new password a second time in response to the prompt:

   Re-enter new password: **your new password**

To set or change your password, enter the **passwd** command:

```
$ passwd
```

Remember to press the **Enter** key after typing your entry. The system then responds with the following message and prompt:

```
Changing password for username
Old password:
```

Note: If you do not have an old password, the system does not display this prompt.

Type your old password following the prompt **Old password:**. For security reasons, the system does not display your password as you type it.

After the system verifies your old password, it is ready to accept your new password, and displays the following prompt:

```
New password:
```

Type your new password following the prompt. Remember that your entry does not appear on the screen.

Finally, to verify the new password (since you cannot see it as you type), the system prompts you to enter the new password again.

```
Re-enter new password:
$ _
```

When the $ (shell) prompt returns to the screen, your new password is in effect.

Keep the following points in mind when setting and changing passwords:

Try to remember your password; you cannot log in to the system withou it. However, if you do forget your password, see *Managing the AIX Operating System* to learn how to remove password protection from your account. Once you have removed the protection, promptly set a new password that you can remember.

You can vary the length and composition of passwords, although password must contain at least four characters. These characters can be letters, numbers, and punctuation marks. However, if your password contains only one type of character (for example, lowercase letters), it must be at least six characters long. The system recognizes only the first eight characters in a password.

On most systems, you can change your password as frequently, or a rarely, as you like. However, certain sites may set limits on how often users may change their passwords or on the length of time passwords remain valid.

If you are a user in a Japanese locale, your password must always be string of ASCII characters. Be sure your keyboard is in ASCII mode for this entry. Doublewidth Roman entries look similar, but AIX does not accept them. If your keyboard is not already in ASCII mode, you can refer to Appendix C to learn how to select that mode.

For information about setting password time limits, see *Managing the AIX Operating System*.

Subtopics

1.7.1 Setting Display Station Characteristics

You can use the **stty** command to modify the way in which a display station works. The general format of **stty** is:

```
stty flag
```

Following is a list of the **stty** options, called flags in the command format, that you may find useful in setting the characteristics of your display station.

Flag	Action
-a	Displays the current **stty** settings.
echo	Causes the **Backspace** key to erase characters when you backspace over them.
page	Causes the system to display information one screen at a time instead of scrolling through the entire output of a command without pause. When **page** is set, press another key (for example, **Enter**) to display the next screen of information. Use **page** in conjunction with **length** to set the number of lines displayed on the screen. To disable the **page** function, use the **-page** flag. The **-page** flag is the normal setting, called the ***default***, until you change it.
length n	Sets screen length to **n** lines, where **n** is a number from 1 to 255.

Note: The **length** flag works **only** in conjunction with the **page** flag.

In the following example, the **stty** command sets the system to display information one screen at a time and sets the length of the screen to 22 lines:

```
$ stty page length 22
$ _
```

For more information about **stty** flags, see **stty** in *AIX Operating System Commands Reference*. For information about running **stty** automatically and for additional information about display station features, see *Managing the AIX Operating System*.

1.7.2 Using Virtual Terminals

It may be easier to understand the concept of ***virtual terminal*** if you know something about how the AIX Operating System handles commands. Ordinarily, you enter a command, wait for the command to execute or complete its action, and then enter your next command.

The operating system, however, can actually run more than one command at the same time. For example, if your system has more than one display station, you can enter a command at one display station, move to the next display station and enter another command, and so on, until you have commands running at every display station on the system.

The AIX Operating System virtual terminal feature gives you the equivalent of having multiple display stations. However, rather than moving from one display station to the next to enter different commands, you remain seated at the main display station and use commands and keys to move from one virtual terminal to the next. In a way, a virtual terminal is like a window opening into the operating system, and you can perform different tasks in each of these windows.

Note: Only the console has the virtual terminal feature. If you try to use virtual terminals on the main display station and find that they do not work, see *Managing the AIX Operating System* for an explanation of how to make the virtual terminal feature available.

Using Virtual Terminals

1. To start a virtual terminal, enter:

   open sh

2. To move from one virtual terminal to another, press:

   NEXT WINDOW (**Alt-Action**)

3. To close (stop) a virtual terminal, press:

   END OF FILE (**Ctrl-D**)

4. Before you log out, close all virtual terminals you have opened.

The **open sh** command opens a virtual terminal with the standard operating system command interpreter (**sh**, or the Bourne shell). If you want to open a virtual terminal with a different command interpreter, substitute the name of that program for **sh**.

After you open a virtual terminal, you can enter a command just as yo normally would. If you have several virtual terminals open, **NEXT WINDOW** (**Alt-Action**) moves you from one to the next, in the order in which you opened them, as though they are connected in a ring. The maximum number of virtual terminals that the system can have open concurrently is 16.

For more information about virtual terminals and other features of the main display station, see *Managing the AIX Operating System*.

1.7.3 Before You Continue . . .

Before you continue with the remainder of this book, you should become familiar with a text editing program.

You can use any of the available AIX editing programs:

ed (see Appendix B, "Creating and Editing Files with ed")

INed editor (see *AIX INed*)

vi (see *AIX Operating System Commands Reference*).

Your system may have other editing programs as well. Use any text editor with which you are familiar, or follow the instructions given in "Creating Sample Files for This Chapter" in topic 2.3 for creating a file with the **ed** program.

Note: If you are using a Japanese keyboard, you need to be familiar with the layout and process of making text entries in ASCII, Romaji, Hiragana and Katakana. This information is found in Appendix C, "Working in a Japanese Locale."

2.0 Chapter 2. Displaying and Printing Files

Subtopics
2.1 CONTENTS
2.2 About This Chapter
2.3 Creating Sample Files for This Chapter
2.4 Displaying Files--The pg (page) Command
2.5 Printing Files--The print Command

2.1 CONTENTS

2.2 About This Chapter

This chapter explains how to display and print your files.

When you want to examine the contents of a file, you have two options. You can:

Display the file on the screen of your display station

Print the file on the system printer

You can choose to display or print the file so that its content is either ***unformatted*** or ***formatted***.

Unformatted The system displays or prints the file just as it is, without adding any special characteristics that govern the appearance of the contents.

Formatted The system displays or prints the file with particular characteristics that dictate the appearance of the contents, such as double spacing, a heading for each page, the number of characters per line, and the number of lines per page.

Note: This chapter requires you to work with three files that you create with a ***text editing program***. "Creating Sample Files for This Chapter" in topic 2.3 explains how to create these files.

Later chapters of this guide require you to be generally familiar with one of the text editing programs, which are listed in "About This Chapter" in topic 1.2.

2.3 Creating Sample Files for This Chapter

You need three practice files to work through the display and printing examples in this chapter. The editing example in this section shows how to use the **ed** program to create these files. If you are familiar with a different editing program, you can use that program to create the practice files. If you already have created three files with an editing program, you can use those files by substituting their names for the file names used in the examples.

In the following examples, you should type the text that is printed in **special characters, like this**. Do not, however, type the system prompts or responses.

The **ed** program is a ***line editor***, which means you can work on only one line of text at a time. You cannot use the cursor control keys (the arrow keys) to move the cursor up or down a line. Once you press **Enter** to start a new line of text, you cannot move back to a previous line to change it. You do not have to press **Enter**, however, unless you want to start a new line before the cursor reaches the end of the current line. You can just continue typing because **ed** automatically wraps around and continues the text on the following line.

Appendix B contains a full tutorial on the **ed** editor. Refer to this appendix for information on creating, editing, and saving text files with **ed**.

Note: If you are using a Japanese keyboard, practice this exercise with the keyboard in ASCII mode the first time through, even if the characters you enter are meaningless to you. When you finish, go back and repeat all the steps entering the various forms of Japanese text. Practice all the text entry modes, even the ones you think you will seldom use. Follow the same procedure with all of the upcoming exercises. Whenever you enter Japanese characters your locale must be set for Japanese and you must be running the **kjterm** program. You can refer to Appendix C, "Working in a Japanese Locale" for details. See "Using ed to Enter Japanese Text" in topic B.16 if you need instructions on using **ed** to enter Japanese text.

Subtopics
2.3.1 Using ed

2.3.1 Using ed

You start the **ed** program by typing the command **ed** and the name of a new or an existing file following the prompt, and then pressing the **Enter** key:

```
$ ed file1
```

This is a new file, so the system responds with a prompt in the form of a question mark (?) and the file name that you entered.

```
?file1
```

You indicate that you want to add text to the new file by typing the letter **a** on a line by itself and then pressing **Enter**. The entries on your screen now look like this:

```
$ ed file1
?file1
a
```

Now type the text of the file, beginning on the line following the **a**.

Note: It does not matter if you make typing mistakes when entering this text. However, if you type something incorrectly and want to correct it before you move to the next line, use the **Backspace** key to erase backward over the current line of text.

```
$ ed file1
?file1
a
You start the ed program by entering
the command ed, followed by the name
of a new or existing file.
```

That's all you need to type for the text of **file1**. Make sure you press **Enter** at the end of the last line of text.

Indicate that you have finished your current work by first typing a period (.) on the line following the last line of text and pressing **Enter**. Then enter the letter **w**.

Note: In the remainder of this book, the use of the word enter in an example or a quick reference box indicates that you should type the indicated information and then press the **Enter** key.

Entering the letter **w** indicates to the system that you want to write, or save a copy of the new file. Your screen will look like this:

```
$ ed file1
?file1
a
You start the ed program by entering
the command ed, followed by the name
of a new or existing file.
.
w
101
```

The number following the **w** indicates the number of characters entered while typing the file.

You are still in **ed**, so you can continue creating the sample files used in the remainder of this chapter. The process is exactly the same as the one you used to create **file1**; only the text is different.

```
e file2
?file2
a
If you are creating a new file, the ed
program first displays the prompt ?.  You
then type an a on a line by itself to indicate
that you are ready to add text to the file.
.
w
171
e file3
?file3
a
When you finish entering your text, enter a period
on a line by itself.  Then enter w to write (or save)
a copy of the new file.
.
w
129
q
```

Notice that the final entry in the example is the letter **q**. You can exit from the **ed** program in one of two ways. You can:

- Enter the letter **q** (remember that the use of enter means to type the material and then press the **Enter** key.)

- Use the **END OF FILE** (Ctrl-D) sequence.

2.4 Displaying Files--The pg (page) Command

If the **$** (shell) prompt is on your screen, you can use the **pg** (page) command to display the contents of one or more files.

Note: You can also use an editing program to display files.

Displaying a File

Use the following format to display the contents of a file. Enter the command and the name of the file at the prompt:

```
pg filename
```

The **filename** entry can be the name of one file or a series of file names separated by spaces:

```
pg filename filename filename
```

Subtopics

2.4.1 Displaying Files without Formatting--The pg (page) Command

In the following example, the **pg** command displays the contents of the file named **file1**:

```
$ pg file1
```

Notice that the command displays the file and then an END OF FILE message. Press **Enter** and the system prompt appears

```
You start the ed program by entering
the command ed, followed by the name
of a new or existing file.
(EOF):
$ _
```

Now use the **pg** command to view the contents of both **file1** and **file2**. Notice again that the command displays the first file and then an **END OF FILE** message. Press **Enter** and a new message appears:

```
(Next file:  file2)
```

Press **Enter** again to see **file2**:

```
$ pg file1 file2
You start the ed program by entering
the command ed, followed by the name
of a new or existing file.
(EOF):
(Next file:  file2)
If you are creating a new file, the ed
program first displays the prompt ?.  You
then type the letter a on a line by itself
to indicate that you are ready to add
text to the file.
(EOF):
$ _
```

Note: The **pg** command always displays multiple files in the order in which you listed them on the command line.

When you display files that contain more lines than will fit on the screen, the **pg** command pauses as it displays each screen. To see the next screen of information, press **Enter**.

2.4.2 Formatting Files for Display--The pr Command

Formatting is the process of controlling the way in which the contents of your files appear when you display or print them. The **pr** command formats a file in a simple but useful style. Used without any options, the **pr** command does the following:

Divides the contents of the file into pages

Puts the date, time, page number, and file name in a heading at th top of each page.

Leaves five blank lines at the end of the page to skip over th perforations between sheets of continuous form paper.

Formatting a File with the pr Command

Enter the following to apply simple formatting characteristics to a file:

pr **filename**

When you use the **pr** command to format a file for display, the contents of the file may scroll up and off of your screen too quickly for you to read them. In that case, you can view the formatted file using the **stty page** command, described in "Setting Display Station Characteristics" in topic 1.7.1. This command instructs the system to pause at the end of each page. You then press **Enter** to display the next page.

Sometimes you may prefer to display or print a file in a more sophisticated format. You can use a number of options, called flags in the command format, to specify additional formatting features. Table 2-1 explains several of these flags. A number of command flags enable you to control the way in which **pr** formats your files. Table 2-1 explains some of the most useful **pr** command flags.

Table 2-1. pr Command Flags

Flag	Action	Example
+num	Begins formatting on page number **num**. Otherwise, formatting begins on page 1.	**pr +2 file1**
-num	Formats page into **num** columns. Otherwise, **pr** formats pages with one column.	**pr -2 file1**
-m	Formats all specified files at the same time, side-by-side, one per column.	**pr -m file1 file2**

-d	Formats double-spaced output. Otherwise, output is single spaced.	**pr -d file1**
-wnum	Sets line width to **num** characters. Otherwise, line width is 72 characters.	**pr -w40 file1**
-onum	Offsets (indents) each line by **num** character positions. Otherwise, offset is 0 character positions.	**pr -o5 file1**
-lnum	Sets page length to **num** lines. Otherwise, page length is 66 lines.	**pr -l30 file1**
-h	Uses next string of characters, rather than the file name, in the header. If the string includes blanks or special characters, it must be enclosed in quote marks ' ' (single quote marks).	**pr -h 'My Novel' file1**
-t	Prevents **pr** from formatting headings and the blank lines at the end of each page.	**pr -t file1**
-schar	Separates columns with the character **char** rather than with blank spaces. You must enclose special characters in quote marks.	**pr -s'*' file1**

Note: For detailed information about the flags available with the **pr** command, refer to the description of the command in *AIX Operating System Commands Reference*.

You can use more than one flag at a time with the **pr** command. In the following example, you instruct **pr** to format the file **efile** with these characteristics:

in two columns **-2**)

with double spacing **d**)

with the title **My Novel** rather than the name of the file.

```
$ pr -2dh 'My Novel' efile
$ _
```

2.5 Printing Files--The print Command

Use the **print** command to send one or more files to the system printer. This command has the following form:

print filename

If your system has multiple printers, you can also use the following format to specify exactly where you want the file to print.

print queueargname filename

The **print** command actually places files in a ***print queue***, which is a list of files waiting to be printed. Once the **print** command places your files in the queue, you can continue to do other work on your system while you wait for the files to print.

If your system has more than one printer, there will be a corresponding queue to service each of them. Every queue has its own **queueargname**, which is the queue's identifier to be used with **print**. To print a file on a specific printer, simply include its corresponding **queueargname** as the second argument of the **print** command.

To display the status of each print queue along with its corresponding **argname**, and to see the current status of the printer devices which they service, enter the following command:

print -q

If your system has more than one printer, the system administrator establishes one of them as the default printer. Whenever you enter the **print** command without specifying any **queueargname**, the system automatically prints your file on the default printer.

Printing a File

Use the following format, entered at the $ prompt, to print a file on the default system printer:

print **filename**

You can print multiple files using the following format:

print **filename1 filename2 filename3**

If your system has more than one printer, specify the printer you want to use with the following format:

print **queueargname filename**

Printers often have names such as **lpd**. The following example shows how to

use the **print** command to print one or more files on a printer named **lpd** with a corresponding **argname** of **-lpd**:

```
$ print -lpd file1
$ print -lpd file2 file3
$ _
```

The first **print** command sends the file **file1** to the **lpd** printer and then displays the **$** prompt. The second **print** command sends **file2** and **file3** to the same print queue, and then displays the shell prompt before the files finish printing.

Several **print** command flags enable you to control the way in which your file prints. Following is the general format for using a flag with this command:

print flag filename

Table 2-2 explains some of the most useful **print** command flags.

Table 2-2. print Command Flags

Flag	Action	Example
-ca filename	Cancels a print request. (Do not specify **queueargname**.)	**print -ca file1**
-nc=num	Prints **num** copies of the file. Normally, only one copy is printed.	**print -nc=3 file1**
-no	Notifies you when the print job is completed by placing a message on your screen.	**print -no file1**
-q	Displays the status of the print queue and printer.	**print -q**
-tl=title	Places **title** on the first page of the printed document and displays **title** when you use the **-q** flag.	**print -tl=Book file1**
-to=name	Labels the printout for delivery to this person.	**print -to=fred afile**
-cp	Copies the file. If you want to change a file while you are waiting for the current copy to print, use the **-cp** flag.	**print -cp file1 file2**

You can specify more than one flag with the **print** command. In the following example, you instruct **print** to send a message to your screen

when **file1** has printed, and to label the printout for delivery to Fred.

```
$ print -no -to=fred file1
```

You can also specify a particular print queue, along with other options. The order of the options does not matter.

For detailed information on printing and printer control, refer to the following:

Managing the AIX Operating System

piobe in *AIX Operating System Commands Reference*

print in *AIX Operating System Commands Reference*.

3.0 Chapter 3. Using the File System

Subtopics

3.1 CONTENTS

3.2 About This Chapter

A ***file*** is a collection of data stored together in the computer. A ***file system*** is the arrangement of files into a useful order. This chapter explains the concepts of the file system and the commands you use to work with it.

The information in this chapter covers files, directories, and path names. Understanding this material can help you design a file system that is appropriate for the type of information you use and the way in which you work.

A good way to learn about the file system is to try the examples in this chapter on your work station; you can practice handling both files and directories. When you work through the examples, do them in order. That way, you make the information on your screen consistent with the information in this guide.

In the examples, the entries you should type are printed in **characters like this**. When the text instructs you to enter a command name or a string of characters, type the characters and then press the **Enter** key.

This chapter describes file systems that are stored on the operating system fixed disk. You also can create file systems on diskettes. For information about creating and using diskette file systems, see *Managing the AIX Operating System*.

Note: The examples in this chapter rely on the files created in "Creating Sample Files for This Chapter" in topic 2.3.

3.3 Understanding Files, Directories, and Path Names

A ***file*** is a collection of data stored together in a computer. A file on a computer is like a document stored in a filing cabinet. Every file in a computer has a ***file name***, which both people and the system use to refer to the file.

A ***directory*** is a collection of files that have been grouped together. A directory is like a folder in a filing cabinet. In a file folder, you store related documents; in a directory, you store related files.

A ***file system*** is a variable sized portion of contiguous space on a fixed disk. A file system is like a drawer in a filing cabinet; it contains one or more file drawers.

To better understand the file system, you should become familiar with each of the following parts of the system in detail:

Files and file name

Directorie

Tree structures and path names

Subtopics

3.3.1 Files and File Names

A file can contain the text of a document, a computer program, records for a general ledger, the numerical or statistical output of a computer program, or other data.

A file name, which can be up to 255 characters long, can contain letters, numbers, and underscores. File names should not include characters that have a special meaning to the shell, including **** (backslash), **!** (exclamation point), **&** (ampersand), and **@** (**at** sign). For information about characters with special meanings to the shell, refer to "Shell Reserved Characters and Words" in topic A.10.

Note: Unlike some operating systems, the AIX Operating System distinguishes between uppercase and lowercase letters in file names (that is, it is ***case sensitive***). For example, the following three file names specify three different files: **filea**, **Filea**, and **FILEA**.

It is a good idea to use file names that reflect the actual contents of your files. For example, a file name such as **memo.advt** indicates that the file contains a memo dealing with advertising. A file name such as **a..,,,..**, on the other hand, tells you little or nothing about the contents of that file. It is also a good idea to use a consistent pattern to name related files. For example, suppose you have a report, which is divided into chapters, with each chapter contained in a separate file. You might name these files in the following way: chap 1, chap2, chap3, and so on.

Subtopics
3.3.1.1 For Japanese Local Users

3.3.1.1 For Japanese Local Users

You can express file names in both ASCII and Japanese characters. When expressed in ASCII, file names can be up to 255 characters long. When expressed in Japanese characters, file names can vary in length depending on which Japanese characters you choose. (Some commonly used Japanese characters are encoded in one byte, but most take two bytes.) If you choose Japanese characters that are all encoded with two bytes, the maximum length of a file name is 127 characters. If you choose characters all encoded with one byte, the maximum length of a file is 255 characters.

An easy way to estimate the size of the Japanese characters you are entering is to look at the display width indicator at the bottom of the screen. The display width corresponds fairly closely with the size of the character being entered. Therefore, if the indicator reads "single width" you are usually entering one-byte characters, and if it reads "double width" you are usually entering two-byte characters.

Very few users need file names longer than the 14 bytes considered standard on most UNIX systems. And, if you want your files to be portable to (able to be read and used on) other UNIX systems, including earlier versions of the AIX Operating System, you should create files with names expressed only in ASCII characters and no longer than 14 characters long.

Japanese file names should not include characters that have a special meaning to the shell. For information about these forbidden characters, see "Shell Reserved Characters and Words" in topic A.10.

The ***single-width*** dot, backslash, ampersand, exclamation point and at sign should also not be used in file names. Their ***double-width*** counterparts are permitted.

3.3.2 Directories and Subdirectories

With the file system, you can organize your files into groups and subgroups that resemble the cabinets, drawers, and folders in a manual file system. These groups are called ***directories***, and the subgroups are called ***subdirectories***. A well-organized system of directories and subdirectories lets you retrieve and manipulate the data in your files quickly and efficiently.

Directories differ from files in two significant ways:

Directories are organizational tools; files are storage places fo data.

Directories contain the names of files, other directories, or both

Note: In order to distinguish between files and directories, you may find it convenient to use two different naming conventions when you assign names to these entities. If you can't tell by the name, you can determine whether a file is an ordinary file, a directory, or another type of file by executing either the **ls -l filename** command, the **file filename** command, or **ls -F**. For further information about the **ls** command, refer to *AIX Operating System Commands Reference*.

When you first log in, the system automatically places you in your ***login directory***, which was created for you when your computer account was established. However, a file system in which all files are arranged under your login directory is not necessarily the most efficient method of organizing your data.

As you work with the system, you may want to set up additional directories and subdirectories so you can organize your files into useful groups.

Note: "Creating a Directory--The mkdir (Make Directory) Command" in topic 3.4 discusses the process of creating new directories.

Once your files are arranged, you will probably need to move around between directories as you work first in File A, located in Directory X, and then in File B, located in Directory Y. The directory in which you are working at any given time is your ***current working directory***.

You can always find out the name of your current working directory by executing the command **pwd** (print working directory):

```
$ pwd
```

The system displays the name of your current working directory in a form such as the following:

```
/usr/msg
```

indicating that you are currently working in a directory named **/msg**, which is located under the **/usr** directory.

Note: The **/usr/msg** notation is called the ***path name*** of your current working directory. See "File System Structure and Path Names" in topic 3.3.3 for information about path names.

Whenever you are uncertain what directory you are working in, or where

that directory fits in the system, use the **pwd** command to find out.

3.3.3 File System Structure and Path Names

The files and directories in the file system are arranged hierarchically, in a structure that resembles an upside-down tree with the root at the top and the branches at the bottom. This arrangement is called a tree structure.

Figure 3-1 shows a typical file system arranged in a tree structure.

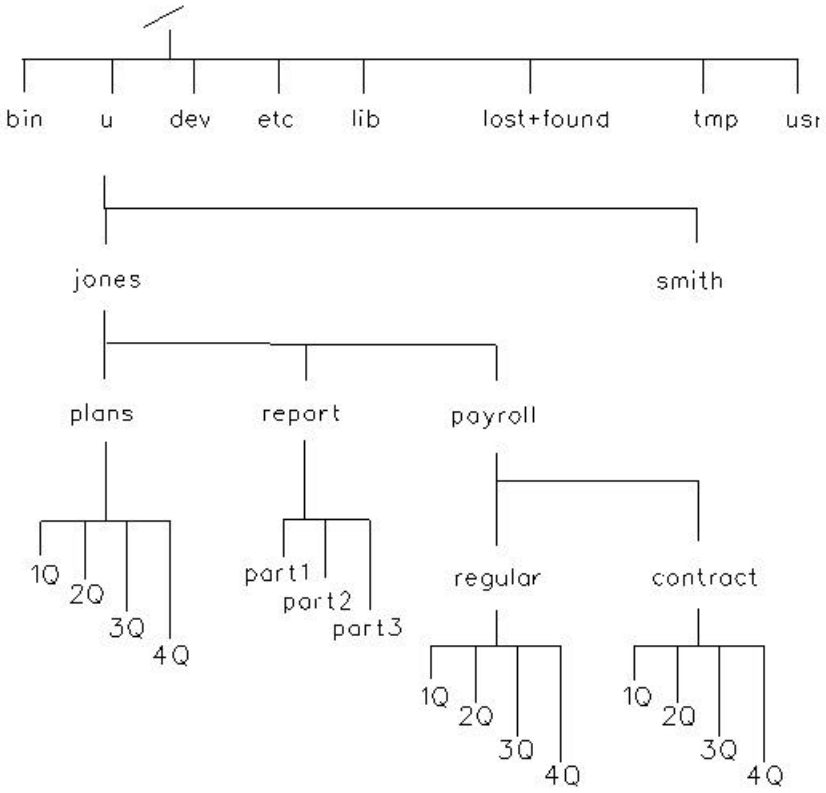


Figure 3-1. A Typical AIX Operating System File System

Subtopics
3.3.3.1 The Tree-Structure File System
3.3.3.2 Path Names

3.3.3.1 The Tree-Structure File System

At the top of the file system shown in Figure 3-1 in topic 3.3.3 (that is, at the root of the inverted tree structure) is a directory called the **root directory**. The symbol that represents this first major division of the file system is a slash (**/**).

At the next level down from the root of the file system are eight directories, each with its own system of subdirectories and files. The illustration, however, shows only the subdirectories under the directory named **u**. These are the login directories for the users of this system.

The third level down the tree structure contains the login directories for two of the system's users, **smith** and **jones**. It is in these directories that **smith** and **jones** begin their work after logging in.

The fourth level of the figure shows three directories under the **jones** login directory: **plans**, **report**, and **payroll**.

The fifth level of the tree structure contains both files and subdirectories. The **plans** directory contains four files, one for each quarter. The **report** directory contains three files comprising the three parts of a report. Also on the fifth level are two subdirectories, **regular** and **contract**, which further organize the information in the **payroll** directory.

A higher level directory is frequently called a ***parent directory***. For example, in Figure 3-1 in topic 3.3.3 the directories **plans**, **report**, and **payroll** all have **jones** as their parent directory.

3.3.3.2 Path Names

A ***path name*** specifies the location of a directory or a file within the file system. For example, when you want to change from working in File A in Directory X to File B in Directory Y, you enter the path name to File B. The AIX Operating System then uses this path name to search through the file system until it locates File B.

A path name consists of a sequence of directory names separated by slashes (**/**) that ends with a directory name or a file name. The first element in a path name specifies where the system is to begin searching, and the final element specifies the target of the search. The following path name is based on Figure 3-1 in topic 3.3.3:

```
/u/jones/report/part3
```

The first **/** represents the root directory and indicates the starting place for the search. The remainder of the path name indicates that the search is to go to the **u** directory, then to directory **jones**, next to directory **report**, and finally to the file **part3**.

Whether you are changing your current working directory, sending data to a file, or copying or moving a file from one place in your file system to another, you use path names to indicate the objects you wish to manipulate.

A path name that starts with a **/** (the symbol representing the root directory) is called a ***full path name***. You can also think of a full path name as the complete name of a file or a directory. Regardless of where you are working in the file system, you can always find a file or a directory by specifying its full path name.

Note: There are special file types (symbolic links) which allow multiple path names to refer to a single file. For more information see "Symbolic Links" in topic 3.10.5.

If there are other users on your system, you may or may not be able to get to their files and directories, depending upon the permissions set for them. For more information about file and directory permissions, see "Changing Permissions--The chmod (Change Mode) Command" in topic 3.14.2.

The file system also lets you use ***relative path names***. Relative path names do not begin with the **/** that represents the root directory because they are **relative** to the current working directory. You can specify a relative path name in one of three ways:

As the name of a file in the current working director

As a path name that begins with the name of a directory one leve below your current working directory

As a path name that begins with **..** (**dot dot**, the relative path name for the parent directory).

Note: Every directory contains at least two entries: **..** (**dot dot**) which refers to the parent directory, and **.** (**dot**) which refers to the current working directory.

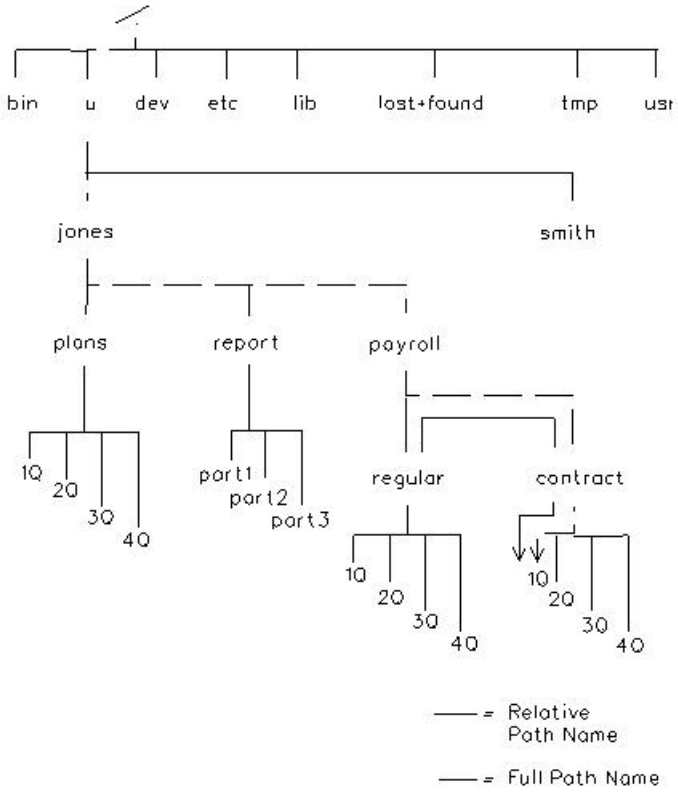


Figure 3-2. Relative and Full Path Names

In Figure 3-2, for example, if your current working directory is **jones**, the relative path name for the file **1Q** in directory **contract** is **payroll/contract/1Q**.

Note: Compare this relative path name with the full path name for the same file, **/u/jones/payroll/contract/1Q**. You can see that using relative path names may mean less typing and more convenience.

The rest of this chapter explains how to create and modify the file system, and how to use file system commands.

3.4 Creating a Directory--The mkdir (Make Directory) Command

The directories in a computer file system correspond to the file drawers in a manual filing system. Like file drawers, directories enable you to organize individual files into useful groups. For example, you could put all the sections of a report in a directory named **report**, or the data and programs you use in cost estimating in a directory named **estimate**. A directory can contain files, other directories, or both.

Note: Before you can work through the examples in the rest of this chapter, you must be logged in to the system and have in your login directory the three files created in "Creating Sample Files for This Chapter" in topic 2.3: **file1**, **file2**, and **file3**. If you are using files with different names, make the appropriate substitutions as you work through the examples. To produce a listing of the files in your current working directory, enter the **ls** command, which is explained in detail in "Listing Directory Contents-- The ls (List) Command" in topic 3.5. For more information on the **ls** command, see *AIX Operating System Commands Reference*.

Your login directory is created for you when you first begin using the operating system, but you will probably need additional directories for the files you create and edit while working with the system. Creating a new directory with the **mkdir** (make directory) command is a very simple process.

The form of the **mkdir** command is

mkdir dirname

where **dirname** is the name of the new directory.

The system creates **dirname** as a subdirectory of your current working directory. This means that the new directory is located at the next level below your current directory.

+--- **Creating a Directory** ----------+
¦ Enter the **mkdir** command in the following form: ¦
¦ mkdir **dirname** ¦
¦ The **dirname** entry is the name you want to assign to the new directory. ¦
+----------+

To create a directory, enter the **mkdir** command followed by the name you want to assign to the new directory. The following example shows how to create a directory named **project**:

```
$ mkdir project
$ _
```

The new directory, **project**, is located one level below the current working directory in the file system tree structure, as shown in Figure 3-3.

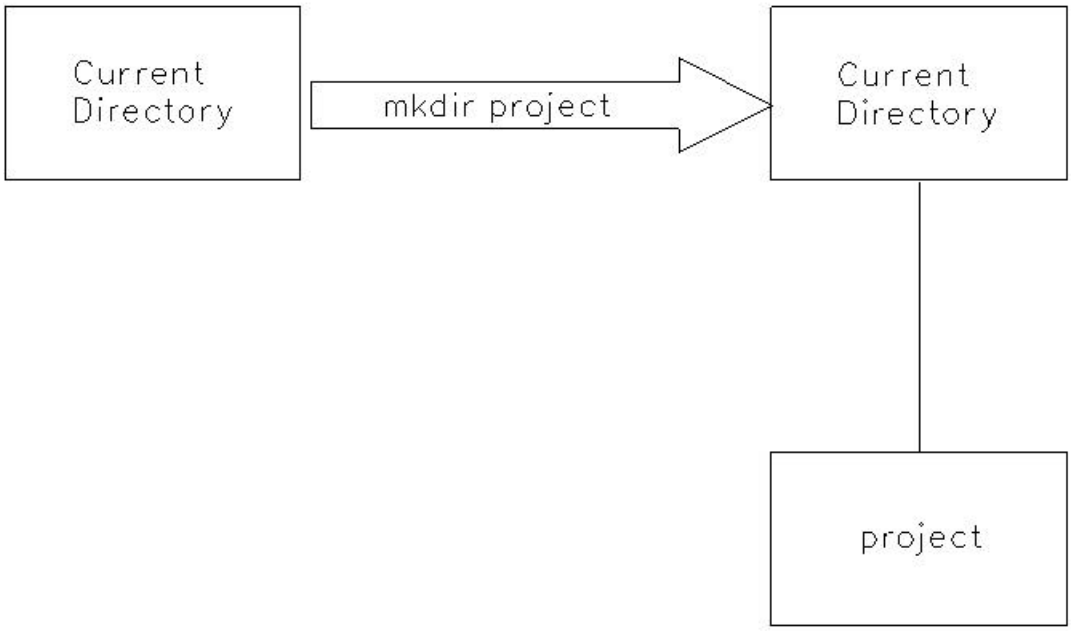


Figure 3-3. Relationship between a New Directory and the Current Working Directory

Like file names, directory names can be up to 255 characters in length. These names can contain letters, numbers, periods, commas, and underscores. The system does not have a symbol or notation that automatically distinguishes between a file name and a directory name, so you may find it useful to establish your own naming conventions to designate files and directories.

Note: If you are a user in a Japanese locale, you are free to name directories with Japanese characters. (A directory is basically a file, so directories can also be named with Japanese strings.) However, the path delimiter (the symbol used by the shell to tell where a file or directory name starts or ends) must remain an ASCII slash. It is possible for you to have full pathnames that look like this:

/Japanese-characters/Japanese-characters/Japanese-characters

When typing such a string you **must** switch your keyboard back to the ASCII mode to type the slash. The Japanese yen sign is often used in a way similar to the slash, but the shell does not recognize it as the path name delimiter symbol.

3.5 Listing Directory Contents-- The ls (List) Command

You can display a listing of the contents of one or more directories by executing the **ls** (list) command. The general format of this command follows:

ls

This simple command, which you will use frequently, produces a list of the files and subdirectories (if any) in your current, or working directory. You can also display other types of information with the **ls** command, such as listing the contents of directories other than your working directory.

The **ls** command has a number of options, called flags in the command format, that enable you to display different types of information about the contents of a directory. Refer to "Flags Used with the ls Command" in topic 3.5.3 for information about these flags.

Listing the Contents of a Directory

Enter the **ls** command in the following format:

ls **dirname**

The **dirname** entry is the name of the directory whose contents you want to display. If you omit **dirname**, you will see a listing of files in the current directory.

Note: For information about using the **ls** command with the **-l** flag to check the access permission for individual files, refer to "Protecting Files and Directories" in topic 3.14.

Subtopics
3.5.1 Listing the Contents of Your Current Working Directory
3.5.2 Listing the Contents of Other Directories
3.5.3 Flags Used with the ls Command

3.5.1 Listing the Contents of Your Current Working Directory

Use the **ls** command to list the contents of your current directory. Enter the following at the $ prompt:

```
$ ls
```

The system displays the following type of listing:

```
$ ls
file1     file2     file3     project
$ _
```

Used without flags in this format, the **ls** command simply displays a list of the names of the files and directories in your current directory.

3.5.2 Listing the Contents of Other Directories

To display a listing of the contents of a directory other than your current working directory, use the **ls** command followed by the path name of the target directory.

In the following example, the current working directory is your login directory. Use the **ls** command to display a list of the contents of a directory named **/bin**. Notice that the name of the **bin** directory is preceded by a slash (/), which indicates that the system should begin searching in the root directory.

Note: The example below shows only a partial listing of the files in the **/bin** directory.

```
$ ls /bin
Rsh     STTY     actman     ar      as
at      awk      basename   batch   bellmail
cat     cb       cc         chgrp   chmod
chown   chtcb    cmp        cp      cpio
$ _
```

As you can see from the example, the **ls** command ordinarily sorts directory and file names alphabetically. Your listing may contain a different set of system files, depending on the way in which your system is customized.

3.5.3 Flags Used with the ls Command

In its simplest form, the **ls** command displays only the names of files and directories contained in the specified directory. However, **ls** has several options, or flags, that provide additional information about the listed items or change the way in which the system displays the listing.

When you want to include flags with the **ls** command, use the following format:

ls -flagname(s)

The **-flagname(s)** entry is actually the first letter of the name of the flag. For example, the **-l** flag produces a **long** listing of the directory contents, as shown in the table below. Notice also that all the **ls** flags are preceded by a hyphen (-).

If you want to use multiple flags with the command you can type the flag names together or list them separately. The first method requires a single hyphen preceding the string. The second method accepts one hyphen for each flag.

ls -ltr or **ls -l -t -r**

The following table lists some of the most useful **ls** command flags.

Table 3-1. ls Command Options

Flag	Action
-l	Lists in **long** format. An **-l** listing provides the type, permissions, number of links, owner, group, size and time of last modification for each file or directory listed.
-t	Sorts the files and directories by the **time** they were last modified (latest first), rather than alphabetically by name.
-a	Lists **all** entries. Without this flag, the **ls** command does not list the names of entries that begin with **.** (period), such as relative directory names, **.profile**, and **.login**.
-r	Reverses the order of the sort to get **reverse alphabetic** order (**ls -r**), or **reverse time** order (**ls -tr**).

The following example shows a long (**-l**) listing of a current directory (your login id will replace the **user** entry):

```
$ ls -l

total 0
-rw-r--r--   1 user1  group1   101   Jun 5  10:03  file1
-rw-r--r--   1 user2  group1   171   Jun 5  10:03  file2
-rw-r--r--   1 user1  group1   130   Jun 5  10:06  file3
drwxr-xr-x   2 user1  group1    32   Jun 5  10:07  project
$ _
```

The following table explains the information displayed on your screen

after you execute the **ls -l** command.

Table 3-2. ls -l Command Information

Field	**Information**
total 0	Number of 1024-byte blocks taken up by files in this directory. The number is zero because the files you created are very small. A file less than 384 bytes is written within the inode itself and requires no additional blocks of storage. For more information see *Managing the AIX Operating System*.
drwxr-xr-x	File type and permissions set for each file or directory. The first character in this field indicates file type: **-** (hyphen) for ordinary files **d** for directories **b** for block special files **c** for character special files **p** for pipe (first in, first out) special files. **l** for symbolic links **s** for sockets **h** for hidden directories The remaining characters (**rwxr-xr-x**) indicate what **r**ead, **w**rite, and e**x**ecute permissions are set for owner, group, and others. For more information on permissions, see "Changing Permissions--The chmod (Change Mode) Command" in topic 3.14.2 and the **mknod** command in *AIX Operating System Commands Reference*.
1	Number of links to each file. For an explanation of ***file links***, see "Linking Files--The ln (Link) Command" in topic 3.10.
uname	Login ID of the file's owner.
gname	Group to which the file belongs.
101	Number of characters in the file.
Jun 5 10:03	Date and time the file was created or last modified.
file1	Name of the file or directory.

There are other **ls** command flags that you may find useful as you gain experience with the system. Refer to the explanation of the **ls** command in *AIX Operating System Commands Reference* for detailed information about the **ls** command flags.

3.6 Changing Directories--The cd (Change Directory) Command

The **cd** (change directory) command changes your current (working) directory. You can access any directory in the file system from any other directory in the file system by executing **cd** with the proper path name.

Note: You must have permission to access a directory before you can use the **cd** command to make that directory your current directory. For information about directory permissions, see "Protecting Files and Directories" in topic 3.14.

Following is the general form of the **cd** command:

cd pathname

You can use either a full path name or a relative path name with the **cd** command, depending on the location of the directory.

Note: If you enter the **cd** command without a path name, the system returns you to your login directory. If you would prefer to have this command return you to a directory other than your login directory, see "Special Shell Variables" in topic A.5. To check the name of your current directory, enter the **pwd** (print working directory) command.

Changing Your Current Working Directory

Enter the **cd** command in the following form:

cd **pathname**

The **pathname** entry can be either the full path name or the relative path name of the directory that you want to set as your current working directory.

Returning to Your Login Directory

Enter the **cd** command in the following format:

cd

Subtopics
3.6.1 Changing Your Current Working Directory
3.6.2 Returning to Your Login Directory
3.6.3 Using Relative Directory Names (. and .. Notation)

3.6.1 Changing Your Current Working Directory

In the next example, you first enter the **pwd** command to display the name (which is also the path name) of your current working directory.

Note: Unless you have already entered the **cd** command, you are currently in your login directory, which is represented in the examples by the notation **/u/username**. This may or may not be the actual name for this directory. Check with your system administrator.

```
$ pwd
/u/username
```

Now enter the **cd** command with the relative path name **project** to change to the **project** directory.

```
$ cd project
```

The system makes **project** the current directory. Enter **pwd** again to verify that the **cd** command has executed successfully.

```
$ pwd
/u/username/project
$ _
```

3.6.2 Returning to Your Login Directory

To return to your login directory from any other directory in the file system, use the **cd** command without a path name:

```
$ cd
$ pwd
/u/username
$ _
```

To change your current directory to a directory that is either above your current directory or on a different branch of the file system tree structure, enter the **cd** command with a full path name:

```
$ pwd
/u/username
$ cd /u
$ pwd
/u
$ _
```

3.6.3 Using Relative Directory Names (. and .. Notation)

Every directory contains at least two entries represented by . (dot) and .. (dot dot). These entries refer to directories relative to the current directory.

. (dot) This entry refers to the **current** directory.

.. (dot dot) This entry refers to the **parent** directory of your working directory. The parent directory is the directory immediately above the current directory in the file system tree structure.

To display the relative directory names, use the **-a** flag with the **ls** command. (For more information about the **ls** command and its flags, see "Listing Directory Contents-- The ls (List) Command" in topic 3.5.)

Note: In the following example, type the name of your login directory instead of the entry **username**.

```
$ cd username/project
```

In the following example, the first **ls** command does not return any information about the contents of the directory **project** because you have not yet created any files or subdirectories in **project**.

```
$ ls
```

Now execute the **ls** command with the **-a** flag to list the directory entries that begin with a **.** (dot)--the relative directory names.

```
$ ls -a
.
..
$ _
```

You can use the relative directory name **..** to refer to files and directories located above the current directory in the file system tree structure. That is, if you want to move up the directory tree one level, you can use the relative directory name for the parent directory rather than using the full path name.

In the following example, the **cd ..** command changes the current directory from **project** to your login directory, which is the parent directory of **project**. Remember that the **username** entry represents your login directory; therefore, the **cd ..** command in this example is the same as the command **cd /u/username**

```
$ pwd
/u/username/project
$ cd ..
$ pwd
/u/username
$ _
```

To move up the directory structure more than one level, you can use a series of relative directory names, as shown in the following example. The slash (/) entry here represents the root directory and the example shown moves you up two parent directory levels to the root directory.

```
$ cd ../..
$ pwd
/
$ _
```

3.7 Removing Files--The rm (Remove File) Command

When you no longer need a file, you can remove it with the **rm** (remove file) command. You use this command to remove a single file, multiple files, and, with the appropriate flag, multiple files and directories.

Following is the general format of the **rm** command:

rm filename

The **filename** entry can be simply the name of the file, the relative path name of the file, or the full path name of the file. The format you use depends on where the file is located in relation to your current directory.

Removing a File

Enter the **rm** command in the following form:

```
rm filename
```

The **filename** entry can be a simple file name, a full or relative path name, or a list of file names.

Subtopics
3.7.1 Removing a Single File
3.7.2 Removing Multiple Files--Matching Patterns
3.7.3 Removing Multiple Files and Directories--the -r Flag

3.7.1 Removing a Single File

In the following example, you remove the file called **file1** from your login directory.

First, return to your login directory with the **cd** (change directory) command. Next, enter the **pwd** (print working directory) command to verify that your login directory is your current working directory, and then list its contents. Remember that the system substitutes the name of your login directory for the notation **/u/username** in the example.

```
$ cd
$ pwd
/u/username
$ ls
file1     file2     file3     project
```

Now enter the **rm** command to remove **file1**, and then list the contents of the directory to verify that the system has removed the file.

```
$ rm file1
$ ls     file2     file3     project
```

You must have write and execute permissions to access a director before you can remove files from it. For information about directory permissions, see "Protecting Files and Directories" in topic 3.14.

You also can remove files with the **rm -i** command, which prompts you for verification before deleting a file.

Note: You can ***link*** multiple file names to a single file. In addition to removing one or more files, **rm** also removes the links between files and file names. The **rm** command actually removes the file itself only when it removes the last file name for that file. For information about links, see "Linking Files--The ln (Link) Command" in topic 3.10.

3.7.2 Removing Multiple Files--Matching Patterns

You can remove more than one file at a time with the **rm** command by using the following ***pattern-matching*** characters. Such characters can represent another character, or a string of characters.

***** Matches any string of characters in a file name.

? Matches any character in a file name.

[...] Matches any of the characters enclosed between the brackets.

For example, the pattern-matching string ***.jun** matches any of the following file names: **receivable.jun**, **payable.jun**, **payroll.jun**, and **expenses.jun**. You could remove all four of these files from your current directory with the **rm *.jun** command.

Warning: Be certain that you understand how the ***** pattern-matching character works before you use it. For example, the **rm *** command removes **every** file in your current directory. Instead of using **rm** to remove a file, you may prefer to use **rm -i**, which prompts you for verification before deleting a file or files.

You can also use the pattern-matching character **?** with the **rm** command to remove files whose names are the same except for a single character. For example, if your current directory contains the files **record1**, **record2**, **record3**, and **record4**, you could remove all four files with the **rm record?** command.

For detailed information about pattern-matching characters, see "Matching Patterns" in topic 4.4.5.

3.7.3 Removing Multiple Files and Directories--the -r Flag

Ordinarily, the **rm** command removes only files, not directories.

Note: For information about removing directories with the **rmdir** command, see "Removing Directories--The rmdir (Remove Directory) Command" in topic 3.8.

You can, however, remove directories with the **rm** command by entering it with the **-r** (recursive) flag. When used with **rm**, the **-r** flag removes the files from a directory and then deletes the directory itself. It also finds any dependent directories and removes them, along with the files in them. Following is the format for the **rm** command with the **-r** flag:

rm -r pathname

Warning: Be certain that you understand how the **-r** flag works before you use it. For example, entering the **rm -r *** command would delete all files, subdirectories, and the files in the subdirectories to which you have access under the current working directory. If you have superuser authority, this command could delete all system files.

Another flag used with the **rm** command, the **-i** (interactive) flag, enables you to remove files selectively. When using the **rm -r** command to remove files or directories, it is a good idea to include the **-i** flag in the command line, in the following form:

rm -ri pathname

When you enter the command in this form, the system prompts you for verification before actually removing the specified item(s). In this way, by answering **y** (yes) or **n** (no) in response to the prompt, you control the actual removal of a file or the directory.

3.8 Removing Directories--The rmdir (Remove Directory) Command

When you no longer need a particular directory, you can remove it from the file system with the **rmdir** (remove directory) command. This command removes only empty directories, which contain no files or subdirectories. You cannot remove your current (working) directory with the **rmdir** command.

Note: For information about removing files from directories, see "Removing Files--The rm (Remove File) Command" in topic 3.7.

Following is the general format of the **rmdir** command:

rmdir dirname

The **dirname** entry is the name, or path name, of the directory you want to remove.

Removing a Directory

1. Make sure the directory is empty.

   a. Enter the following to list the directory's contents:

      ```
      ls -a
      ```

      Note: See if there are any files beginning with a **dot** (.). You will certainly see one called **dot** (.) and one called **dot dot** (..), but there may be others. Common ones are .profile and .login.

   b. Use the **rm** command to remove any files. Remember that entering this command with the * pattern-matching character removes **all** files from your current directory except those that begin with a**dot** (.). Be sure you have no valuable data in any of the files you remove.

      ```
      rm *
      ```

   c. Use **rmdir** to remove any directories.

      rmdir **dirname**

      Note: You can also use the **rm -r** command to remove both files and subdirectories from your current directory. You may choose to enter the command with the **-i** (interactive) flag (**rm -ri**) in order to have the system prompt you for verification before actually removing any entries.

2. Use the **cd** command to move to a different directory (usually the parent directory):

   ```
   cd ..
   ```

3. Enter the following, where **dirname** is the name or path name of the directory you want to remove:

   rmdir **dirname**

Before working through the examples in this chapter, create three subdirectories in the directory **project**.

First, use the command **cd project** to set **project** as your current directory. Next, use the **mkdir** command to create the directories **schedule**, **tasks**, and **costs**. Finally, use the **cd** command to return to your login directory.

```
$ cd project
$ pwd
/u/username/project
```

Now create the three new subdirectories and then list the contents of the **project** directory.

```
$ mkdir costs schedule tasks
$ ls
costs     schedule     tasks
```

Return to your login directory.

```
$ cd
$ pwd
/u/username
$ _
```

Subtopics
3.8.1 Removing a Directory
3.8.2 Removing Multiple Directories
3.8.3 Removing Your Current Working Directory

3.8.1 Removing a Directory

The **rmdir** command removes only empty directories. If you try to remove a directory that contains any files or subdirectories, the **rmdir** command gives you an error message, as the following example shows:

```
$ rmdir project
rmdir:    project not empty
$ _
```

Before you can remove the directory **project**, you must first remove the contents of that directory. In the following example, the **cd** command makes **project** your current directory, and then the **ls** command lists the contents of **project**:

```
$ cd project
$ ls
costs     schedule     tasks
```

Now remove the directory **schedule** from the current directory, and then list the remaining contents of the **projects** directory:

```
$ rmdir schedule
$ ls
costs     tasks
$ _
```

3.8.2 Removing Multiple Directories

You can remove more than one directory at a time with the **rmdir** command by using pattern-matching characters--characters that represent other characters or a string of characters. Following are two of the most frequently used pattern-matching characters:

***** Matches any string of characters in a file name.

? Matches any character in a file name.

For example, the characters ***s** (a pattern-matching string) match the names of both the **tasks** and **costs** directories, but do not match the name **schedule**. (Remember that you actually removed the **schedule** directory in the previous example.) The **rmdir *s** command would therefore remove both **tasks** and **costs**.

Note: Entering the **rmdir** command with the * character (**rmdir ***) removes **all** empty directories from your current directory.

The string **??s??** also matches the names of both the **tasks** and **costs** directories. Each of these names consists of two characters matched by **??**, the character **s**, and two more characters, again matched by **??**. The **rmdir ??s??** command would therefore also remove both of these directories from your current directory.

You can use the ***** and **?** pattern-matching characters together. In the following example, the ***s?s** string matches the directories **tasks** and **costs**. Both names consist of a string of characters matched by *****, an **s**, another character matched by the **?**, and a final **s**.

```
$ rmdir *s?s
```

Now enter the **ls** command to verify that the **project** directory contains no entries:

```
$ ls
$ _
```

For detailed information about pattern-matching characters, see "Matching Patterns" in topic 4.4.5.

3.8.3 Removing Your Current Working Directory

You cannot remove your current working directory while you are still working in it. You can remove it only after you move into another directory. You generally enter the **cd ..** (**dot dot**) command to move into the parent directory of your current working directory, and then enter **rmdir** with the path name of the target directory.

The directory **project** is empty. To remove **project**, first move to your login directory, which is the parent directory of **project**. Then use the **rmdir dirname** command to remove **project**, and enter **ls** to confirm the removal.

```
$ cd
$ rmdir project
$ ls
file2     file3
$ _
```

Your login directory now contains only **file2** and **file3**.

3.9 Copying Files--The cp (Copy) Command

The **cp** (copy) command copies files either within your current working directory, or from one directory into another directory.

The **cp** command is especially useful in making backup copies of important files. Because the backup and the original are two distinct files, you can make changes to the original while still maintaining an unchanged copy in the backup file. This is helpful in case something happens to the original version. Also, if you decide you do not want to save your most recent changes to the original file, you can begin again with the backup file.

Copying Files

To copy a file, enter the **cp** command in the following form:

```
cp source destination
```

The **source** entry is the name of the file to be copied. The **destination** entry is the name of the file to which you want to copy **source**.

The **source** and **destination** entries can be file names in your current working directory, or path names to different directories.

To copy files to a different directory, enter:

```
cp source destination
```

In this case, **source** is a series of one or more file names and **destination** is a path name that ends with the name of the target directory.

Subtopics
3.9.1 Copying Files in the Current Working Directory
3.9.2 Copying Files into Other Directories

3.9.1 Copying Files in the Current Working Directory

The **cp** command creates the destination file if it does not already exist. If a file with the same name as the destination file does exist, however, **cp** copies the source file over the existing destination file.

Warning: If the destination file exists, the **cp** command erases the contents of that file before it copies the source file. Be certain that you do not need the contents of the destination file, or that you have a backup copy of the file, before you use it as the destination file for the **cp** command.

In the following example, the destination file does not exist, so the **cp** command creates it. First list the contents of the directory, which is still your login directory unless you have entered the **cd** command since working through the last example.

```
$ ls
file2     file3
```

Now copy the source file, **file2**, into the new destination file, **file2x**:

```
$ cp file2 file2x
```

List the contents of the directory to verify that the copying process was successful:

```
$ ls
file2     file2x     file3
$ _
```

3.9.2 Copying Files into Other Directories

If you have followed the examples to this point, you do not have a subdirectory in your login directory. You need a subdirectory to work through the following example, so create one called **extra** with the **mkdir** command:

```
$ mkdir extra
$ _
```

In the next example, first enter the **cp** command to copy the file **file2** into directory **extra**:

```
$ cp file2 extra
```

Now list the contents of **extra** to verify that it contains a copy of **file2**:

```
$ ls extra
file2
$ _
```

You can also use the **cp** command to copy multiple files from one directory into another directory. Enter the command in the following form:

cp filename1 filename2 dirname

In the following example, enter the **cp** command to copy both **file2** and **file3** into the **extra** directory, and then list the contents of that directory:

```
$ cp file2 file3 extra
$ ls extra
file2     file3
$ _
```

To change the name of a file when you copy it into another directory, enter the name of the source file (the original file), the directory name, a slash (/), and then the new file name. In the following example, copy **file3** into the **extra** directory under the new name **notes**. Then list the contents of the **extra** directory:

```
$ cp file3 extra/notes
$ ls extra
file2     file3     notes
$ _
```

3.10 Linking Files--The ln (Link) Command

A ***link*** is a connection between a file name and the file itself. An ordinary file usually has one link--a connection to its original file name. However, you can use the **ln** (link) command to connect a file to more than one file name at the same time. This is convenient when you need to use the data in a particular file in more than one location in the file system.

You can link files across directories in the same file system using the **ln** command in the following form:

ln filename1 filename2

The **filename1** entry is the name of an existing file. The **filename2** entry is a new file name to be linked to the existing **filename1**.

Note: Compare the **ln** command, which creates multiple names for the same file, with the **cp** command, which actually copies files. Refer to *AIX Operating System Commands Reference* for additional information about the **cp** and **ln** commands.

If you want to link files and directories across file systems, you can create ***symbolic links***. To create a symbolic link, add an **s** flag to the **ln** command sequence and specify the full path names of both files. The **ln** command for symbolic links takes the following form:

ln -s **/dirname1/filename1 /dirname2/filename2**

Symbolic links are discussed in greater detail later in this section.

+--- **Linking Files** ----------+
¦ Enter the **ln** command in the following form: ¦
¦ ln **filename1 filename2** ¦
¦ The **filename1** entry is the name of an existing file. ¦
¦ The **filename2** entry is the name of the file that you want to link to **filename1**. ¦
¦ If you want to create a symbolic link, use the **-s** flag and the path names of the files with the command. ¦
+---------------------------------+

Subtopics

3.10.1 Using Links

Links are convenient whenever you need to work with the same data in more than one place. For example, suppose you have a file containing assembly-line production statistics. You use the data in this file in two different documents--in a monthly report prepared for management, and in a monthly synopsis prepared for the line workers.

You can link the statistics file to two different file names, for example **mgmt.stat** and **line.stat**, and place these file names in two different directories. In this way you save storage space because you have only one real copy of the file. More importantly, you do not have to update multiple files. Because **mgmt.stat** and **line.stat** are linked, editing one automatically updates the other, and both files always contain the same data.

In the following example, the **ln** command links the new file name **checkfile** to the existing file named **file3**:

```
$ ln file3 checkfile
$ _
```

Now use the **pg** command to verify that **file3** and **checkfile** are two names for the same file:

```
$ pg file3
```

The system displays the following:

```
When you finish entering your text, enter a period
on a line by itself.  Then enter w to write (or save)
a copy of the new file.
```

Now display the text of **checkfile**:

```
$ pg checkfile
When you finish entering your text, enter a period
on a line by itself.  Then enter w to write (or save)
a copy of the new file.
$ _
```

Notice that both **file3** and **checkfile** contain the same information. Any change that you make to the file under one name will show up when you access the file by its other name. Updating **file3**, for example, will also update **checkfile**.

If your two files were located in directories that are in two different file systems, you need to create a symbolic link to link them. In this example, **file3** is in the **/charts** directory, and **checkfile** is in the **/summary** directory. The symbolic link between the files and their file system is created by the following:

```
$ ln -s /charts/file3 /summary/checkfile
```

The information in both files is still updated in the same manner as explained above.

3.10.2 How Links Work--Understanding File Names and i-numbers

Each file has a unique identification number, called an ***i-number***. The i-number refers to the file itself--data stored at a particular location--rather than to the file name.

A directory entry is simply a link between an i-number representing a physical file and a file name. It is this relationship between files and file names that enables you to link multiple file names to the same physical file--that is, to the same i-number.

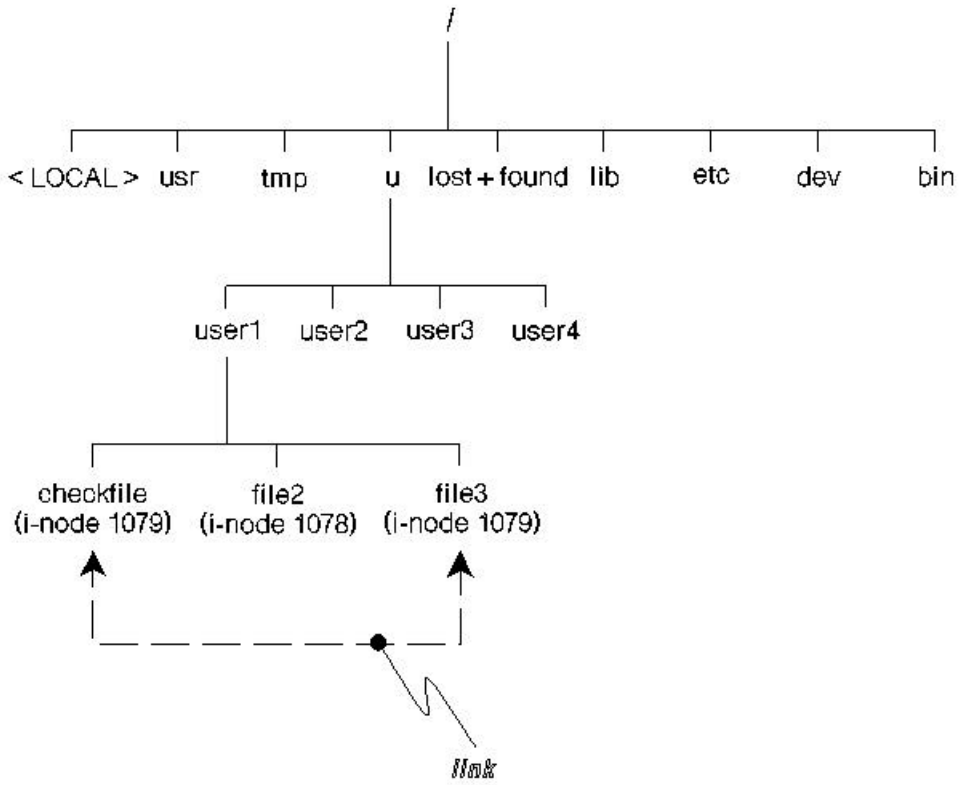


Figure 3-4. Two File Names Within a File System Linked to the Same inode

To display the i-numbers of files in your current working directory, use the **ls** command with the **-i** (print i-number) flag in the following form:

ls -i

Now examine the identification numbers of the files in your login directory, which is your current working directory unless you have entered the **cd** command since working through the last example. The number preceding each file name in the listing is the i-number for that file.

```
$ ls -i
1079 checkfile     1078 file2     1079 file3
$ _
```

The i-numbers in your listing will probably differ from those shown in this example. However, the important thing to note is the identical i-numbers for **file3** and **checkfile**, the two files linked in the previous example. In this case, the i-number is **1079**.

In the case of symbolic links, the link becomes a new file with a new i-number. It does not add another directory entry to the link's i-number but instead has its own i-number for identification in both file systems.

3.10.3 Removing Links

The **rm** (remove file) command does not always remove a file. For example, suppose that a file--that is, an i-number--is linked to more than one file name. In that case, the **rm** command removes the link between the i-number and that file name but leaves the physical file intact. The **rm** command actually removes a physical file only after it has removed the last link between that file and a file name, as shown in Figure 3-5.

Note: For detailed information about the **rm** command, refer to "Removing Files--The rm (Remove File) Command" in topic 3.7.

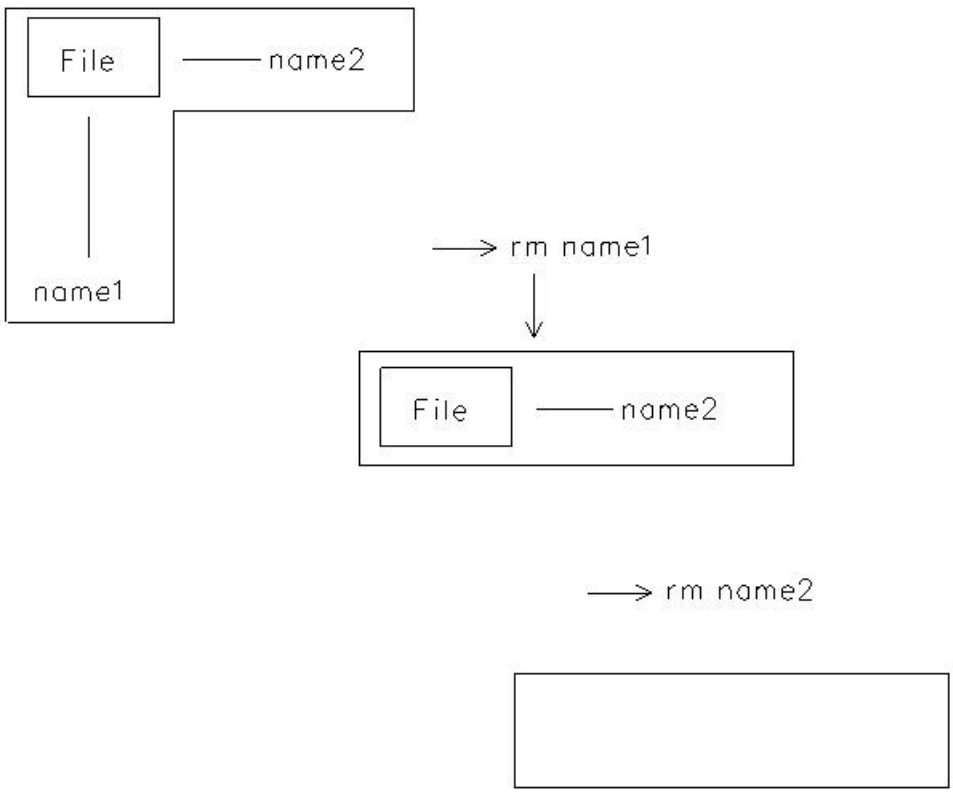


Figure 3-5. Removing Links and Files

To determine the number of file names linked to a particular i-number, use the **ls** command with the **-i** (print i-number) and the **-l** (long listing) flags, in the following form:

ls -il

Now examine the links in your login directory. Remember that the i-numbers displayed on your screen will differ from those shown in the example.

```
$ ls -il
total 0
 1079 -rw-r--r-- 2 user1  group1  130 Jun 5 10:06 checkfile
 1078 -rw-r--r-- 1 user1  group1  171 Jun 5 10:03 file2
 1079 -rw-r--r-- 2 user1  group1  130 Jun 5 10:06 file3
$ _
```

Again, the first number in each entry shows the i-number for that file

name. The second element in each line shows the file permissions, which "Protecting Files and Directories" in topic 3.14 describes in detail. The third field for each entry, the number to the left of the user name, represents the number of links to that i-number. Notice that **file3** and **checkfile** have the same i-number, 1079, and that both show two links. Each time the **rm** command removes a file name, it reduces the number of links to that i-number by one.

In the following example, use the **rm** command to remove the file name **checkfile**.

```
$ rm checkfile
```

Now list the contents of the directory with the **ls -il** command. Notice that the **rm** command has reduced the number of links to i-number **1079**, which is the same i-number to which **file3** is linked, by one.

```
$ ls -il
total0
 1078 -rw-r--r-- 1 user1  group1  171 Jun 5 10:03 file2
 1079 -rw-r--r-- 1 user1  group1  130 Jun 5 10:06 file3
$ _
```

Only one link to i-number 1079 remains. Therefore, another **rm** command would now remove the last remaining link between that i-number and a file name. There would then be no file name associated with that inode. A file which loses its last remaining link cannot be found because it has no name.

The file data is not actually erased, but the inode which marked its existence is returned to the list of inodes which are considered free to be assigned to new files. As far as the system is concerned, it is gone and cannot be recovered.

3.10.4 File Systems

A file system is a single, contiguous span of disk space which carries a hierarchical structure of files upon it. You could create a suitable branching tree structure of files and directories on a floppy disk and use the **mount** command to append it to the existing root file structure of an AIX PS/2 system. (See *Managing the AIX Operating System* for information on **file systems** and the **mount** command.)

One of the deficiencies of the standard **link** command is that it cannot link files across file system boundaries, but only within the same file system. In a large, complex system like the AIX Operating System, it is often desirable to link a filename in your own workspace to a large program or file residing on another file system. This saves storage space by allowing you to share large files and programs with others without having to copy them into your own file directory.

3.10.5 Symbolic Links

Symbolic links differ from the links created by the standard **link** command in that they allow you to create links to directories as well as files. You can also create links across ***file system boundaries***.

Creating a regular file enters a new name in the directory, which is then associated with a particular inode. The inode points to the actual file and its data. A **link** inserts a new name in a directory which is then associated with that inode number.

A ***symbolic link*** is quite different. Although a new file name entry is created in the directory, this entry contains a full path name pointing to a file or directory that resides somewhere else. When the AIX Operating System encounters a ***symbolic link***, it reads the contents as if it were a standard file. Then it follows the pointer to the indicated location where it finds a standard file or directory name. This is referenced to the inode, which is used as a pointer to the actual data block containing the information.

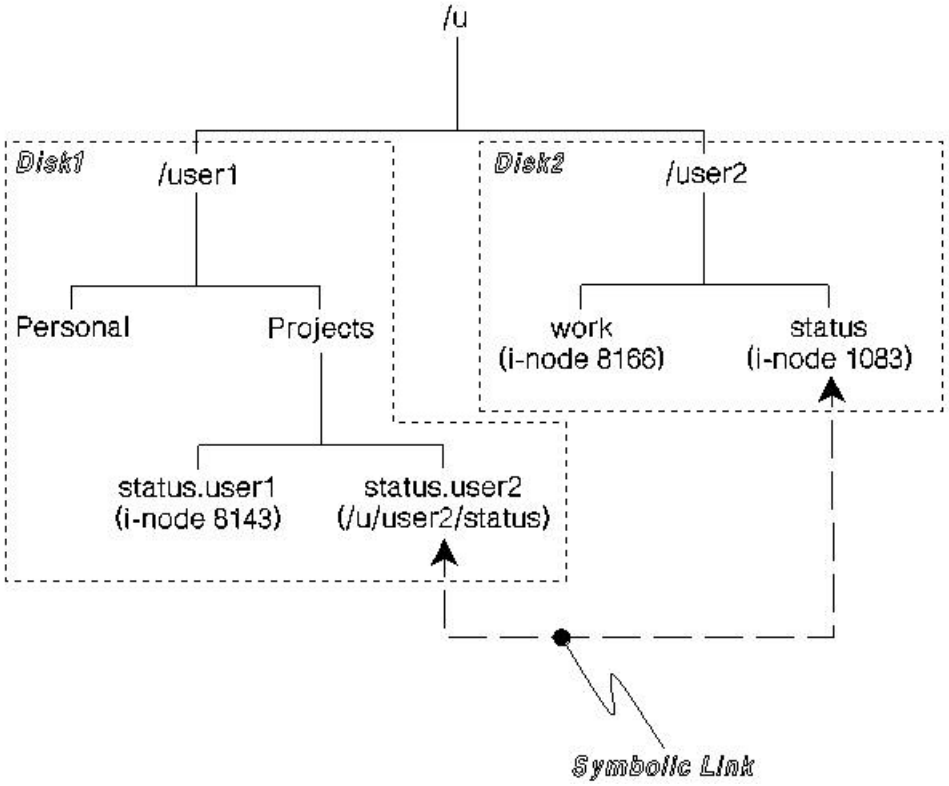


Figure 3-6. File Name Referencing a File in Another Directory by Symbolic Link

Figure 3-6 represents two users. Both users have their own separate file systems on separate floppy disks which have been mounted on the replicated directory tree at the **mountpoint** called **/u**.

The supervisor of this two-person group is **user1**. One of his jobs is to collect status reports for himself and **user2** and combine them into a group status report.

user1 has a file called **/u/user1/projects/status.user2**. This file is

actually a symbolic link containing the character string **/u/user2/status**. This string is a destination path name used to link the first filename to an actual file called **/u/user2/status**. The content of this file is a status report for **user2**.

When **user1** wants to read the status report for the entire group, he moves to his directory called **/u/user/projects** and types:

```
$ cat /u/user1/projects/status.user2
```

When the system reads this file, it finds the text string **/u/user2/status**. It uses this text string as a pointer to the actual file, and the **cat** command sends the contents of the status report to **user1**'s terminal screen.

Note: To see the symbolic links, use the **ls** command with the **-l** flag.

Subtopics
3.10.5.1 Using Symbolic Links To Share Information
3.10.5.2 Accessing Directories Through Symbolic Links
3.10.5.3 Removing Symbolic Links

3.10.5.1 Using Symbolic Links To Share Information

You can use symbolic links to share file information since the linking process updates all the linked files when you update one of them. For example, suppose **user1** and **user2** are working together on project which requires them to use the same files. By using a symbolic link, both users have subdirectories in their home directory that contain the files they need. When **user1** updates any of the files, **user2** can look at the files and see the updates.

user1 creates a directory called **/u/user1/project** and **user2** can create the symbolic link to it by changing to the home directory (using **cd**) and typing the following:

```
$ ln -s /u/user1/project project
```

3.10.5.2 Accessing Directories Through Symbolic Links

It is important to note that accessing a directory through a symbolic link is different from accessing it through its **real** name. In particular, using the **..** (**dot dot**) component of the directory results in accessing the directory's real parent directory, not the parent directory of the symbolic link.

For example, suppose **user2** is working on a file in the **/u/user2/project** directory, which is the symbolic link to **/u/user1/project**. In order to change to the parent directory (**/u/user1**), **user2** types the following:

```
$ cd /u/user2/project
$ cd ..
$ pwd
/u/user1
```

Instead of being in the **/u/user2** directory, **user2** is now in the directory called **/u/user1**.

3.10.5.3 Removing Symbolic Links

Symbolic links also differ from other links when they are removed. When you remove a file with the **rm** command, you eradicate the connection between a file name and the inode that tells the system where to find the file. When you remove a regular link, you are doing the same thing. Any inode may have multiple links; that is, it may be associated with multiple file names. Only when the last link between an inode and a file name is removed will all the data of the file become unfindable by the system.

When you remove a symbolic link, you are removing the connection between a certain file name and an associated path name. What is lost is simply a reference to the original file. The data of the file, the inode telling where to find that data, and the original file name all remain intact.

If you were to remove the original file without first removing the symbolic link to it, the symbolic link would be left dangling. The second file name would still be associated with the same path name, but the path name would now refer to a file which no longer exists.

3.10.6 The <LOCAL> Alias

The string **<LOCAL>** is the AIX Operating System notation for the name of the host system you are connected to. As a user, you may occasionally see this text string, especially when you do a long listing (**ls -l**) of a symbolic link.

3.11 Renaming or Moving Files and Directories--The mv (Move) Command

You can use the **mv** (move) command to perform the following actions:

Move one or more files from one directory into another directory

Rename files or directories

Note: You can use the **mv** command only to rename directories; you cannot use it to move one directory into another directory.

Following is the general format of the **mv** command:

mv oldfilename newfilename

The **oldfilename** entry is the name of the file you wish to move or rename. The **newfilename** entry is the new name you want to assign to the original file.

The **mv** command links a new name to an existing i-number and breaks the link between the old name and that i-number. It is useful to compare the **mv** command with the **ln** and **cp** commands, which are explained in "Linking Files--The ln (Link) Command" in topic 3.10 and "Copying Files--The cp (Copy) Command" in topic 3.9. Refer also to the descriptions of these commands in *AIX Operating System Commands Reference*.

Note: When you use **mv** to move files into other directories, be very careful to type the name of the destination directory correctly. If you do not supply a valid destination directory name, **mv** could simply rename the file within the same directory, using the invalid directory name as a file name.

+--- **Renaming Files and Directories** ----------+
¦ ¦
¦ Enter the **mv** command in the following form: ¦
¦ ¦
¦ mv **oldfilename newfilename** ¦
¦ ¦
¦ The **oldfilename** and **newfilename** entries can be names of files in the ¦
¦ current working directory, or path names to files in a different ¦
¦ directory. ¦
¦ ¦
+---+

Subtopics
3.11.1 Renaming Files

3.11.1 Renaming Files

In the following example, first list the i-number of each file in your current directory with the **ls -i** command. Then enter the **mv** command to change the name of file **file2x** to **newfile**. (The i-numbers displayed on your screen will differ from the numbers in the example.)

```
$ ls -i
1085 extra     1078 file2     1088 file2x     1079 file3
$ mv file2x newfile
```

Again, list the contents of the directory:

```
$ ls -i
1085 extra     1078 file2     1079 file3     1088 newfile
$ _
```

Notice two things in this example:

First, the **mv** command removes the entry **file2x** and adds the entry **newfile**.

Second, the i-number for the original file **file2x**) and **newfile** is the same--1088.

The **mv** command removes the link between i-number 1088 and file name **file2x**, replacing it with a link between i-number 1088 and file name **newfile**. However, the command does not change the file itself.

3.12 Moving Files into a Different Directory

You can also use the **mv** command to move one or more files from your current working directory into a different directory.

Note: Type the directory name carefully because the **mv** command does not distinguish between file names and directory names. If you enter an invalid directory name, the **mv** command simply takes that name as a new file name. The result is that the file is renamed rather than moved.

In the following example, the **ls** command lists the contents of your current working directory. Then the **mv** command moves **file2** from your current working directory into the **change** directory.

```
$ ls
change     file2     file3     newfile
```

Now enter the **mv** command to place **file2** into the **change** directory and then list the contents of your current directory to make sure the file is gone:

```
$ mv file2 change
$ ls
change     file3     newfile
```

Finally, list the contents of the **change** directory to verify that the command has moved the file:

```
$ ls change
file2     file3     notes
$ _
```

Subtopics

3.12.1 Renaming and Moving Directories

3.12.1 Renaming and Moving Directories

You can use the **mv** command to rename directories.

As shown in Figure 3-7, you could use the **mv** command to change the name of the directory **/u/jones/plans** to **/u/jones/schemes** because both directories have the same parent directory, **/u/jones**.

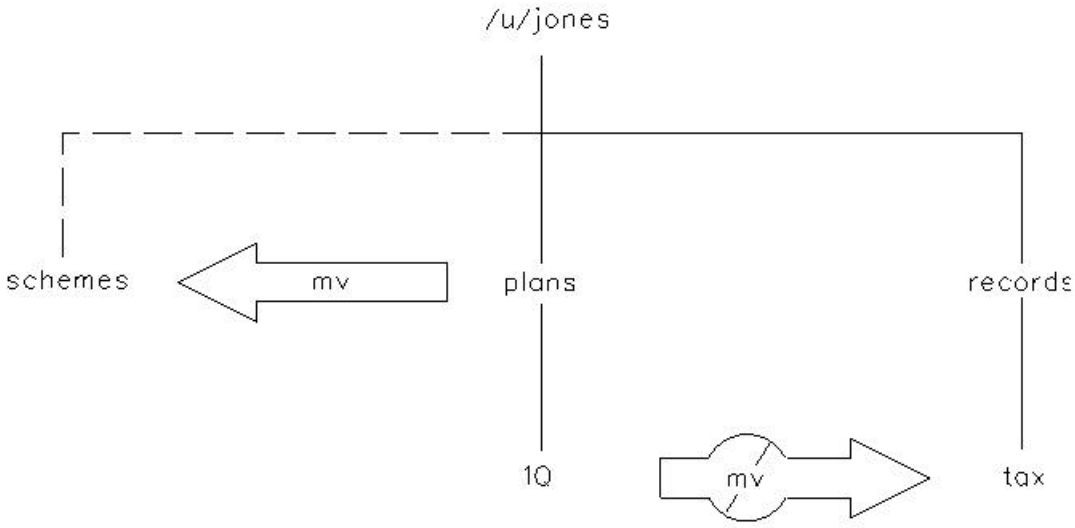


Figure 3-7. Directories That Can and Cannot Be Renamed

However, you could not change the name of the directory **/u/jones/plans/1Q** to **/u/jones/records/tax** with the **mv** command because the two directories have different parent directories.

In the following example, the first **ls -i** command lists the i-numbers for all entries in the current working directory. Notice the i-number of the **extra** directory.

```
$ ls -i
1085 extra     1078 file2     1079 file3     1088 newfile
```

Now enter the **mv** command to change the name of **extra** to **change** and list the contents again:

```
$ mv extra change
$ ls -i
1085 change     1078 file2     1079 file3     1088 newfile
$ _
```

Notice that the second **ls -i** command does not list the original directory name **extra**. However, it does list the new directory name, **change**, and it displays the same i-number (**1085** in this example) for the new directory as

for the original **extra** directory.

3.13 Backing Up and Restoring Files

Note: The **backup** and **restore** operations are highly dependent upon the configuration of your cluster and the type of machine you are logged on to. The following description is for the simplest possible case; a standalone PS/2 with no other machines involved. Ask your System Administrator for the **backup** and **restore** procedures specific to your system.

Files and directories represent a significant investment of time and effort. At the same time, all computer files are potentially easy to change or erase, either intentionally or by accident. A ***backup copy*** of a file is a duplicate of that file normally stored on a removable storage medium (diskette or tape).

When you have a recent backup copy of a file, you have a good place to start over if your original file is damaged or lost. To make backup copies of individual files, use the **backup** command, which has the following basic format:

backup

The **backup** command has a number of flags that enable you to back up various parts of the file system in various ways. You will be probably be most concerned with backing up individual files, so you will generally enter the **backup** command with the **-i** (individual) flag:

backup -i

The **backup -i** command backs up the specified files onto a ***formatted diskette***. Use the **format** command to prepare a diskette as a backup medium.

Warning: Formatting **erases** any data stored on a diskette. Be sure you do not need the data stored on the diskette you format as a backup medium.

+--- **Formatting a Diskette** ----------+

1. Enter the **format** command in the following form:

   format

   The system displays the following message:

   Insert a new diskette for /dev/fd0
   and strike ENTER when ready

2. Insert a diskette in diskette drive 0 (also referred to as drive A) and press **Enter**.

   The system displays the following message:

   Formatting . . .

When the diskette is formatted, the system displays the message:

```
Formatting . . . Format completed
```

The diskette is now ready for use.

Backing Up Individual Files

1. To back up individual files, enter the **backup -i** command in the following form:

   ```
   backup -i
   ```

2. When the system displays the prompt

   ```
   Please mount volume 1 on /dev/rfd0
   ... and type return to continue _
   ```

   make sure a **formatted** diskette into diskette drive 0 (also referred to as drive A) and press **Enter**.

3. After the cursor returns to the left side of the screen, enter **one file name per line** until you have entered the names of all the files you want to back up. Remember to press the **Enter** key after typing each file name, including the last one.

4. Once you have typed the last file name and pressed **Enter**, use the **END OF FILE** (**Ctrl-D**) sequence to begin the backup process.

5. When the backup is complete, the system displays the following type of message:

   ```
   Done     at day date time year
   n blocks on n volume(s)
   ```

   Remove the diskette, label it clearly, and store it in a safe place.

The **backup** command copies files in a compact form that is not directly usable. To make the backup file copies usable, enter the **restore** command to transfer them from the diskette to the system.

Note: The output from this operation is always sent to the diskette drive of the host doing the backup operation. If you are working at an intelligent work station which has its own diskette drive, you might expect to have the backup output appear there. It will not. The diskette must be placed in the disk drive of the host, not in

the disk drive of your own machine.

If you are logged on to a System/370 host, ask your system administrator how to proceed.

Restoring Individual Files

1. Insert the backup diskette into diskette drive 0 (also referred to as drive A).

2. Enter the **restore** command in the following form:

   ```
   restore -x filename
   ```

   where **-x** indicates that only the individually named file should be restored and **filename** is the name of the individual file.

Making backup copies of individual files is important and should be a routine part of your work. However, your backup routine also should include regular backups of complete file systems.

For information about backing up and restoring file systems, see the descriptions of the **backup** and **restore** command in *AIX Operating System Commands Reference*. Refer also to the information about backing up and restoring a file system in *Managing the AIX Operating System*.

3.14 Protecting Files and Directories

The AIX Operating System has a number of commands that enable you to limit access to your files and directories. You can protect a file or directory by setting or changing its ***permissions***, which are simply codes that determine the way in which anyone working on your system can use the stored data.

Setting or changing permissions is also referred to setting or changing the ***protections*** on your files or directories. You generally protect your data for one or both of the following reasons:

Your files and directories contain sensitive information that shoul not be available to everyone who uses your system.

Not everyone who has access to your files and directories should hav the power to alter them.

Checking and Setting Permissions

1. To display the current file permissions, enter the **ls** command with the **-l** flag. To display the permissions for a single file, enter the following:

   ls -l **filename**

   The **filename** entry is the name of the file for which you want to display the permissions.

2. If necessary, use the **chmod** command to change the file permissions:

   chmod **group-operation-permission filename**

   You can also use the following form of the **chmod** command:

   chmod **octalnumber filename**

3. If necessary, use the **chgrp** command to change the ***group*** ID or group name:

   chgrp **group filename**

Warning: Two or more users can make changes to the same file at the same time without realizing it. Several users, for example, might be editing the same file concurrently. In this case, the system **saves** the changes made by the last user to close the file; changes made by the other users are **lost**. It is therefore a good idea to set file permissions to allow only authorized users to modify files. The specified users should then

communicate about when and how they are using the files.

Each file and directory has nine permissions associated with it:

Each file/directory has the following three ***types of permissions***:

r (read)
w (write)
x (execute)

These permissions are associated with each of the following thre ***classes of users***:

u (user/owner)
g (group)
o (all others)

The **r** permission allows users to view or print the file. The **w** permission allows users to write to (modify) the file. The **x** permission allows users to execute (run) the file or to search directories.

The **owner** of a file or directory is generally the person who created it. If you are the owner of a file, you can change the file permissions with the **chmod** command, which is described in "Changing Permissions--The chmod (Change Mode) Command" in topic 3.14.2.

The **group** specifies the group to which the owner belongs. If you are the owner of a file, you can change the ***group ID*** of the file with the **chgrp** command, described in "Changing Owners and Groups" in topic 3.14.3.

Note: If you do not own a file, you cannot change its permissions or group ID unless you have superuser authority.

The meanings of the three types of permissions differ slightly between ordinary files and directories, as Table 3-3 shows.

Table 3-3. Differences Between File and Directory Permissions

Permission	**For a File**	**For a Directory**
r (read)	Contents can be viewed or printed.	Contents can be read, but not searched. Normally **r** and **x** are used together.
w (write)	Contents can be changed or deleted.	Entries can be added or removed.
x (eXecute)	File can be used as a command (program).	Directory can be searched.

Subtopics
3.14.1 Displaying File Permissions
3.14.2 Changing Permissions--The chmod (Change Mode) Command
3.14.3 Changing Owners and Groups

3.14.1 Displaying File Permissions

To display the permissions for all of the files in your current working directory, enter the **ls -l** command:

```
$ ls -l
total 3
drwxr-xr-x  2 user1  group1    96 Jun 5 11:08 change
-rw-r--r--  1 user1  group1   130 Jun 5 10:06 file3
-rw-r--r--  1 user1  group1   101 Jun 5 10:44 newfile
$ _
```

The first string of each entry in the directory shows the permissions for that file or directory. For example, the first entry, **drwxr-xr-x**, shows the following:

that this is a directory (the **d** notation)

that the owner has permission to view it, write in it, and search i (the **rwx** sequence)

that the group can view it and search it, but not write in it (th first **r-x** sequence)

that all others can view it and search it, but not write in it (th second **r-x** sequence).

The third field shows the file's **owner**, represented by **username**, and the fourth field shows the **group** to which the owner belongs.

You also can use the **ls -l** command to list the permissions for a single file, or the **ls -ld** command to list the permissions for a single directory:

```
$ ls -l file3
-rw-r--r--  1 username  system  130 Jun 5 10:06 file3
$ ls -ld change
drwxr-xr-x  2 username  system   96 Jun 5 11:08 change
$ _
```

Taken together, all the permissions for a file or directory are called its ***permission code***. As Table 3-4 shows, a permission code consists of four parts:

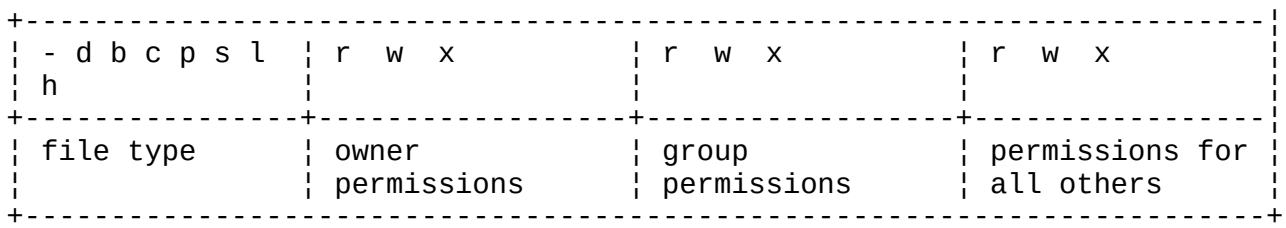
A character that shows the file type: **-** (hyphen) for an ordinary file; **d** for a directory; **b** for a block special file; **c** for a character special file; **p** for a pipe (first in, first out) special file; **s** for a socket; **l** for a symbolic link; and **h** for a hidden directory.

A three-character ***permission field*** that shows **user** (owner) permissions.

A three-character permission field that shows **group** permissions.

A three-character permission field that shows permissions for al **others**.

+--+
¦ Table 3-4. File and Directory Permission Fields ¦



When you create a file or directory, the system automatically supplies a predetermined permission code. Following are typical permission codes:

Permission codes for files

-rw-r--r--

Permission codes for directories

drwxr-xr-x

The hyphens (**-**) in some positions following the file-type notation indicate that the specified class of user does not have permission for that operation.

To change the predetermined permission code, you must change your ***file-creation mode mask*** with the **umask** (set file-creation mode mask) command. For an explanation of **umask**, see the description of the command in *AIX Operating System Commands Reference*.

3.14.2 Changing Permissions--The chmod (Change Mode) Command

Your ability to change permissions gives you a great deal of control over the way your data can be used. Use the **chmod** (change mode) command to set or change the permissions for your files and directories.

For example, you obviously permit yourself to read, modify, and execute a file. You generally permit members of your group to read a file; depending upon the nature of your work and the composition of your group, you often allow them or modify or execute it. You generally prohibit all other system users from having any access to a file.

Note: You must be the owner of the file or directory before you can change its permissions. This means that your user name must be in the third field in an **ls -l** listing of that file, or you must have superuser authority.

There are two ways to specify the permissions set by the **chmod** command:

You can specify permissions with letters and operation symbols

You can specify permissions with octal numbers

+--- **Changing File and Directory Permissions** ----------+
¦ ¦
¦ You can use either of the following formats of the **chmod** command to ¦
¦ change file and directory permissions. ¦
¦ ¦
¦ You can use letters and operation symbols to change file and ¦
¦ directory permissions. Enter the **chmod** command in the following ¦
¦ form: ¦
¦ ¦
¦ chmod **group-operation-permission filename** ¦
¦ ¦
¦ You can also use octal numbers to change file and directory ¦
¦ permissions. Enter the **chmod** command in the following form: ¦
¦ ¦
¦ chmod **octalnumber filename** ¦
¦ ¦
¦ In both examples, the **filename** entry is the name of the file whose ¦
¦ permissions you want to change. ¦
¦ ¦
+---+

Subtopics
3.14.2.1 Specifying Permissions with Letters and Operation Symbols
3.14.2.2 Specifying Permissions with Octal Numbers

3.14.2.1 Specifying Permissions with Letters and Operation Symbols

Following is the basic format of the **chmod** command:

chmod group operation permission filename

Groups, operations, and permissions are defined as follows:

Use one of these letters to represent the **group**:

u for user (owner)
g for group
o for all others (besides owner and group)
a for all (user, group, and all others).

Use one of these symbols to represent the **operation**:

+ add permission
- remove permission
= assign permission regardless of previous setting.

Use one or more of these letters to represent the type of **permission**:

r for **read**
w for **write**
x for **execute**.

For a detailed discussion of file formats, see *AIX Operating System Technical Reference*.

Changing File Permissions: In the following example, first enter the **ls -l** command to display the permissions for the file **newfile**:

```
$ ls -l newfile
-rw-r--r--  1 user1 group1  101 Jun 5 10:44 newfile
```

Now enter the **chmod** command with the flags **go+w**. This format gives both the group (**g**) and all other system users (**o**) write permission (**+w**) for the file:

```
$ chmod go+w newfile
```

Now list the new permissions for the file:

```
$ ls -l newfile
-rw-rw-rw-  1 user1 group1  101 Jun 5 10:44 newfile
$ _
```

Changing Directory Permissions: The procedure for changing directory permissions is the same as that for changing file permissions. However, to list the information about a directory, you use the **ls -ld** command. The **-d** flag provides information about the directory itself rather than the files in it.

```
$ ls -ld change
drwxr-xr-x  2 user1  group1  96 Jun 5 11:08 change
```

Now change the permissions with the **chmod g+w** command so that the group (**g**) has write permission (**+w**) for the directory **change**.

```
$ chmod g+w change
$ ls -ld change
drwxrwxr-x  2 user1  group1  96 Jun 5 11:08 change
$ _
```

Using Pattern-Matching Characters: If you want to make the same change to the permissions of all entries in a directory, you can use the pattern-matching character ***** (asterisk) with the **chmod** command.

Note: For information about pattern-matching characters, see "Shell Reserved Characters and Words" in topic A.10.

In the following example, the command **chmod g+x *** gives execute (**x**) permission to the group (**g**) for which that file is associated (*****) in the current working directory. Each file has a group associated with it. This command grants execute permission to all members of whatever group is associated with each file.

```
$ chmod g+x *
```

Now enter the **ls -l** command to show that the group now has execute (**x**) permission for all files in the current working directory.

```
$ ls -l
total 3
drwxrwxr-x  2 user1  group1   96 Jun 5 11:08 change
-rw-r-xr--  1 user1  group1  130 Jun 5 10:06 file3
-rw-rwxrw-  1 user1  group1  101 Jun 5 10:44 newfile
$ _
```

Setting Absolute Permissions: An ***absolute*** permission assignment resets all permissions for a file or files, regardless of how the permissions were set previously.

In the following example, the **ls -l** command lists the permissions for the file **file3**. Then the command **chmod a=rwx** gives all three permissions (**rwx**) to all users (**a**):

```
$ ls -l file3
-rw-r-xr--  1 username  system  130 Jun 5 10:06 file3
$ chmod a=rwx file3
$ ls -l file3
-rwxrwxrwx  1 username  system  130 Jun 5 10:06 file3
$ _
```

You can also use an absolute assignment (**=**) to remove permissions.

In the following example, the command **chmod a=rw- newfile** removes the execute permission (**x**) for all groups (**a**) from the file **file3**:

```
$ chmod a=rw- file3
$ ls -l file3
-rw-rw-rw-  1 username  system  130 Jun 5 10:06 file3
$ _
```

3.14.2.2 Specifying Permissions with Octal Numbers

In addition to specifying permissions with letters and operation symbols, you can also use the **chmod** command with ***octal numbers***.

An octal number corresponds to each type of permission:

4 = **r**ead
2 = **w**rite
1 = e**x**ecute.

To specify a group of permissions (a permissions field), add together the appropriate octal numbers:

3 = -wx (2 + 1)
6 = rw- (4 + 2)
7 = rwx (4 + 2 + 1)
0 = --- (no permissions).

The entire permission code for a file or directory is specified with a three-digit octal number, one digit each for **owner**, **group**, and **others**. The following table shows how octal numbers relate to permission fields.

Table 3-5. How Octal Numbers Relate to Permission Fields

Octal Number	**Owner Field**	**Group Field**	**Others Field**	**Complete Code**
777	rwx	rwx	rwx	rwxrwxrwx
755	rwx	r-x	r-x	rwxr-xr-x
700	rwx	---	---	rwx------
666	rw-	rw-	rw-	rw-rw-rw-

To use octal number permission codes with the **chmod** command, enter the command in the following form:

chmod octalnumber filename

Enter the following example to change the permission of **file3** using octal numbers:

```
$ ls -l file3
-rw-rw-rw-  1 username  system  130 Jun 5 10:06 file3
$ chmod 754 file3
$ ls -l file3
-rwxr-xr--  1 username  system  130 Jun 5 10:06 file3
$ _
```

It is more difficult to learn to specify permissions with octal numbers than it is to specify them with letters. However, once you are familiar with the octal number system, you may find using it more efficient than setting permissions with letters and operation symbols.

3.14.3 Changing Owners and Groups

In addition to setting permissions, you can control how a file or directory is used by changing its owner or group. Use the **chown** command to change the owner and the **chgrp** command to change the group.

Changing the Owner of a File or Directory

Enter the **chown** command in the following form:

```
chown owner file
```

The **owner** entry is the user name of the new owner of the file.

The **file** entry is a list of one or more files whose ownership you want to change.

Note: Superuser authority is necessary to use the **chown** command.

Changing the Group of a File or Directory

Enter the **chgrp** command in the following form:

```
chgrp group file
```

The **group** entry is the group ID or group name of the new group.

The **file** entry is a list of one or more files whose ownership you want to change.

Note: If you don't have superuser authority, there are restrictions upon which groups you can change.

For more information about the **chown** and **chgrp** commands, see the descriptions of the command in *AIX Operating System Commands Reference*.

4.0 Chapter 4. Using Processes and the Shell

Subtopics
4.1 CONTENTS
4.2 About This Chapter
4.3 Understanding Programs and Processes
4.4 Using the Shell with Processes

4.1 CONTENTS

4.2 About This Chapter

The first part of this chapter explains the concept of a process and the ways in which you can run a process, check the status of a running process, and stop a process.

Once you understand this introductory material, you should be able to use the techniques described in the second part of this chapter, "Using the Shell with Processes" in topic 4.4, to create and control more complex processes.

A good way to learn about processes is to work through the examples in this chapter on your system. In the examples, the entries you should type are printed in **special characters like this**. When the text instructs you to enter a command name or a string of characters, type the characters and then press the **Enter** key.

4.3 Understanding Programs and Processes

A ***program*** is a set of instructions that a computer can interpret and run. You may think of most programs as belonging to one of two categories:

application programs such text editors, accounting packages, o electronic spreadsheets

programs that are components of the AIX Operating System such a commands, the shell, and your login procedure.

While a program is running, it is called a ***process***.

The AIX Operating System can run a number of different processes at the same time. When more than one process is running, a scheduler built into the operating system gives each process its fair share of the computer's time, based on established priorities.

Note: Only your system administrator can raise these priorities. However, any user can lower priorities with the **nice** command, which is explained in *AIX Operating System Commands Reference*.

This section, which is an introduction to using processes, explains how to perform the following types of operations:

Check the status of processes

Cancel processes

Run processes in the background

- Check the status of background processes.

- Cancel background processes.

For more detailed information about processes and how to control them, see "Using the Shell with Processes" in topic 4.4. For reference-oriented material about features of the shell, consult Appendix A, "Using Advanced Bourne Shell Features."

Subtopics

4.3.1 Checking Process Status--The ps (Process Status) Command

A program that is actually running on the computer is referred to as a ***process***.

For example, a file on the system contains the program for the **cp** command, which copies files. When you enter this command, you start a **cp** process that runs until the system displays the prompt. This display signals that the copy is complete and that the **cp** process is finished.

Any time the system is running, several processes are also running. You can use the **ps** (process status) command to find out which processes are running, and to display information about those processes.

Checking Process Status

Enter the **ps** command in the following form:

```
ps
```

In the following example, the **ps** command displays the status of all processes you have started:

```
$ ps
   PID       TT    STAT     TIME     COMMAND
   1640956   p0    S>       0:10     sh
   1640997   p0    R>       0:00     ps
   1640856   p0    R>       0:11     qdaemon
$ _
```

You interpret these entries as follows:

PID Process identification. The system assigns a ***process identification number*** (PID number) to each process when that process starts. There is no relationship between a process and a particular PID number; that is, if you start the same process several times, it will have a different PID number each time.

TTY or TT Terminal designation. On a system with more than one display station, this field tells you which display station started the process. On a system with only one display station, this field can contain the designation **console** or the designation for one or more virtual terminals.

STAT The state of the process. This field tells you what the process is currently doing. It can be **R**unning (**R>**), **S**leeping (**S>**), or any of a number of other possibilities.

TIME Time devoted to this process by the computer; displayed in minutes and seconds as of when you enter **ps**.

COMMAND The name of the command (or program) that started the process. In this example, **sh** is the shell program, **ps** is the process status command that displayed this information, and **qdaemon** is a program that lets you send data to the printer.

Generally, the simple **ps** command described here tells you all you need to know about processes. However, you can control the type of information that the **ps** command displays by using its flags. One of the most useful **ps** flags is **-e**, which causes **ps** to return information about all processes, not just those associated with your display station.

For an explanation of the **ps** command flags, see **ps** in *AIX Operating System Commands Reference*.

4.3.2 Canceling a Process

If you start a process and then decide you do not want to let it finish, you can cancel it by pressing the **INTERRUPT** keystroke sequence.

Note: **INTERRUPT** does not cancel ***background*** processes. To cancel a background process, you must use the procedure described under "Ending a Background Process--The kill Command" in topic 4.3.6.6.

Canceling a Running Process

Press:

INTERRUPT

Note: Most simple commands are not good examples for demonstrating how to cancel a process--they run so quickly that they finish before you have time to cancel them.

The examples in the rest of this chapter therefore use a command that takes more than a few seconds to run: **find / -type f -print**. This command displays the path names for all files on your system. You do not need to study the **find** command in order to complete this chapter--it is used here simply to demonstrate how to work with processes.

If however, you want to learn more about the **find** command, see **find** in *AIX Operating System Commands Reference*.

In the following example, the **find** command starts a process. After the process runs for a few seconds, you can cancel it by pressing **INTERRUPT**:

```
$ find / -type f -print
/usr/lib/acct/acctcms
/usr/lib/acct/acctcon1
/usr/lib/acct/acctcon2
/usr/lib/acct/acctdisk
/usr/lib/acct/acctmerg
/usr/lib/acct/accton
/usr/lib/acct/acctprc1
/usr/lib/acct/acctprc2
/usr/lib/acct/acctwtmp
/usr/lib/acct/chargefee
/usr/lib/acct/ckpacct
/usr/lib/acct/dodisk
/usr/lib/acct/fwtmp
/usr/lib/acct/lastlogin
/usr/lib/acct/monacct
/usr/lib/acct/nulladm
/usr/lib/acct/prctmp
/usr/lib/acct/prdaily
/usr/lib/acct/prtacct
/usr/lib/acct/runacct
/usr/lib/acct/sdisk
/usr/lib/acct/shutacct     <INTERRUPT key sequence>
$ _
```

The system returns the **$** (shell) prompt to the screen. Now you can enter another command.

Note: There are actually more files in **/usr/lib/acct** than are shown here. This output was stopped by use of the **INTERRUPT** key sequence.

4.3.3 Redirecting Input and Output

A command usually reads its input from the keyboard (***standard input***) and writes its output to the display (***standard output***). Often, though, you may want a command to read its input from a file, write its output to a file, or both. You can select input and output files for a command with the shell notation shown in Table 4-1.

Table 4-1. Shell Notation for Reading Input and Redirecting Output

Notation	Action	Example
<	Reads standard input from a file.	**wc <file3**
>	Writes standard output to a file.	**ls >file3**
>>	Adds standard output to the end of a file.	**ls >>file3**

This section explains how to read input from a file and how to write output to a file.

Subtopics
4.3.3.1 Reading Input from a File--The < Symbol
4.3.3.2 Redirecting Output--The > and >> Symbols

4.3.3.1 Reading Input from a File--The < Symbol

Use the **<** (less than) symbol to take input from a file, as the following example shows:

```
$ wc <file3
      3      27     129
$ _
```

The **wc** (word count) command counts the number of lines, words, and characters in the named file. If you do not supply an argument, the **wc** command reads its input from the keyboard. In this example, however, input for **wc** comes from the file named **file3**.

Note: The **wc** command has three flags, **-l** (line count only), **-w** (word count only), and **-c** (character count only), which you can use separately or in combination with each other.

For more information about the **wc** command, see *AIX Operating System Commands Reference*.

4.3.3.2 Redirecting Output--The > and >> Symbols

To send output to a file, use either the **>** (greater than) or the **>>** symbol. The **>** symbol causes the shell to replace the contents of the file with the output of the command; the shell deletes the contents of the original file. The **>>** symbol adds (appends) the output of the command to the end of a file. If you use **>** or **>>** to write output to a file that does not exist, the shell creates the file.

In the next example, the output of **wc** goes to the file named **file**:

```
$ wc >file
$ _
```

If the file already exists, the shell replaces its contents with the output of **wc**. If **file** does not exist, the shell creates it.

In the following example, the shell adds the output of **wc** to the end of the file named **file**:

```
$ wc >>file
$ _
```

If **file** does not exist, the shell creates it. Page to the file to see the results of what you have done.

```
$ pg file
```

In addition to their standard output, processes often produce error or status messages known as ***diagnostic output***. For information about redirecting diagnostic output, see "Standard Error and Other Output" in topic A.8.

4.3.4 Job Control

Besides allowing the processes of many users at one time to run simultaneously, the AIX Operating System allows each single user to run many of his own processes at once. The user's control over this multitasking feature is called ***job control***. You are free to stop any process you have running. That program is removed from action temporarily, but can be started again at the same point of operation. You may also use the **kill** command to stop a process permanently.

4.3.5 Foreground vs. Background

Foreground vs. **background** is an operating system prioritizing method. Programs running in the foreground are given the highest priority, while those running in the background are considered less important and given attention whenever the operating system has computing cycles to spare.

A **foreground** process is interactive, that is it normally sends its output to the screen and takes the user's responses from the keyboard. A **background** process cannot interact with the user and normally sends its output to somewhere other than the screen (into a file or to the printer, for example).

The advantage of the **foreground/background** system is that the user can run several processes at once. Any process that can run on its own (without input from the user) can run in the background while the user continues working with another process that requires input in the foreground.

4.3.6 Running Background Processes

Run concurrent multiple processes by entering the appropriate commands with a symbol that instructs the operating system to run those commands as background processes.

The material in this section explains the following tasks:

Entering a command that runs as a background proces

Starting a background proces

Checking the status of a background proces

Killing a background process

Subtopics

4.3.6.1 The & Operator

The **&** (ampersand) operator at the end of a command tells the system to run that command in the background. Once a process is running in the background, you can perform additional tasks by entering other commands at your display station.

Running a Background Process

Enter the name of the command followed by an ampersand:

```
command name&
```

4.3.6.2 Starting a Background Process

Generally, background processes are most useful with commands that take a long time to run. Background processes represent work for the processor; therefore other elements of work may wait for the completion of processes and increase response time for display station users. This may or may not be a problem, depending upon how much the system slows and the nature of the other work you do while background processes run.

Note: You can use the **nice** command to lower the priority of a process, even a background process. For information about the **nice** command, see **nice** in *AIX Operating System Commands Reference*.

Most processes direct their output to standard output, even when they run in the background. Unless redirected, standard output goes to the display station. Because the output from a background process can interfere with your other work on the system, it is usually good practice to redirect the output of a background process to a file or a printer. Then you can look at the output whenever you are ready.

In the following example, the **find** command runs in the background (**&**) and directs its output to a file named **dir.paths** (with the **>** operator):

```
$ find / -type f -print  >dir.paths &
24
$ _
```

When the background process starts, the system assigns it a PID number (24 in this example), displays the number, and then prompts you for another command.

Note: Your process numbers probably will be different from the ones shown in these examples.

After you begin a process, you can type **Ctrl-Z** to suspend the process from running. When you stop a process, the process is suspended until you restart it.

For more information about redirecting output, see "Redirecting Input and Output" in topic 4.3.3.

4.3.6.3 Placing a Stopped Process in the Background--The bg Command

After you stop a process using **Ctrl-Z**, you can use the **bg** command to restart the process and run it in the background. Type **Ctrl-Z** to stop the process, then type **bg** at the prompt to start it running in the background.

4.3.6.4 Moving a Background Process to the Foreground--The fg Command

After you have stopped a process or placed a process in the background, you can use the **fg** command to bring it to the foreground. After you have typed **Ctrl-Z** or **bg**, type **fg** to run the command in the foreground.

4.3.6.5 Checking Background Process Status

As long as a background process is running, you can check its status with the process status (**ps**) command explained under "Checking Process Status--The ps (Process Status) Command" in topic 4.3.1. You can also check the status of a particular process by using the **-p** flag and the PID number with the **ps** command (for example, **ps -p PID number**).

The following example shows how to start another **find** process and then check its status:

```
$ find / -type f -print >dir.paths &
100137
$ ps -p 100137
   PID     TT    TIME    COMMAND
100137     p0    0:11    find
$ _
```

For an explanation of the data that the **ps** command displays, see "Checking Process Status--The ps (Process Status) Command" in topic 4.3.1. The **STAT** field, indicating the state of the process, is omitted from the listing when the **-p** option is used. This is because **-p** specifically requests information on any background process which is in the running state.

You can check background process status as often as you like while the process runs. In the following example, the **ps** command displays the status of the **find** process five times:

```
$ find / -type f -print >dir.paths &
100180
$ ps -p 100180
   PID     TT     TIME    COMMAND
100180     p0     0:18    find
$ ps -p 100180
   PID     TT     TIME    COMMAND
100180     p0     0:29    find
$ ps -p 100180
   PID     TT     TIME    COMMAND
100180     p0     0:49    find
$ ps -p 100180
   PID     TT     TIME    COMMAND
100180     p0     0:58    find
$ ps -p 100180
   PID     TT     TIME    COMMAND
100180     p0     1:02    find
$ ps -p 100180
   PID     TT     TIME    COMMAND
$ _
```

Notice that the sixth **ps** command returns no status information (because the **find** process ended before the last **ps** command was entered). You can also use the **jobs** command to check the status of your processes. To use the **jobs** command, type **jobs** at the prompt. The system responds with a message similar to the following:

```
[1] + stopped enscript
```

The first column lists the job number, the second column lists the status and the third column is the program name.

4.3.6.6 Ending a Background Process--The kill Command

If you decide, after starting a background process, that you do not want the process to finish, you can cancel the process with the **kill** command. Before you can cancel a background process, however, you must know its PID number.

If you have forgotten the PID number of that process, use the **ps** command, described in "Checking Process Status--The ps (Process Status) Command" in topic 4.3.1, to list the PID numbers of all processes.

After the **find** command begins to execute in the following example, the **ps** command, without the **-p** flag, displays the status of all processes:

```
$ find / -type f -print >dir.paths &
139801
$ ps
   PID      TT     STAT     TIME    COMMAND
140956     p0     S>       0:11    sh
139801     p0     R>       0:10    find
139881     p0     R>       0:01    qdaemon
140997     p0     R>       0:03    ps
kill 139801
$ ps
139801 Terminated
   PID      TT     STAT     TIME    COMMAND
140956     p0     S>       0:11    sh
139881     p0     R>       0:01    qdaemon
140997     p0     R>       0:03    ps
$ _
```

The command **kill 139801** stops the background **find** process, and the second **ps** command returns no status information about PID number 139801. The system does not display the termination message until you enter your next command or press the **Enter** key.

4.4 Using the Shell with Processes

The shell is a program that interprets commands for the AIX Operating System and provides a command programming language. Although the commands described in the previous chapters of this book are interpreted by the shell, you do not have to know anything about the shell itself in order to use them.

In contrast, this section of this chapter describes some tasks you can perform using specific features of the shell to:

- Connect commands to each other
- Use multiple commands and groups of commands
- Use special characters to match file names
- Write shell procedures (programs)

For detailed information about using these and other shell features, see Appendix A, "Using Advanced Bourne Shell Features."

Subtopics

4.4.1 Using Pipes and Filters

A ***pipe*** is a one-way connection between two related commands. One command writes its output to the pipe, and the other process reads its input from the pipe. When two or more commands are connected by the ¦ (pipe) operator, they form a ***pipeline***.

Each command in a pipeline runs as a separate process. Figure 4-1 represents the flow of input and output through a pipeline: The output of the first command (Cmd 1) is the input for the second command (Cmd 2); the output of the second command is the input for the third command (Cmd 3).

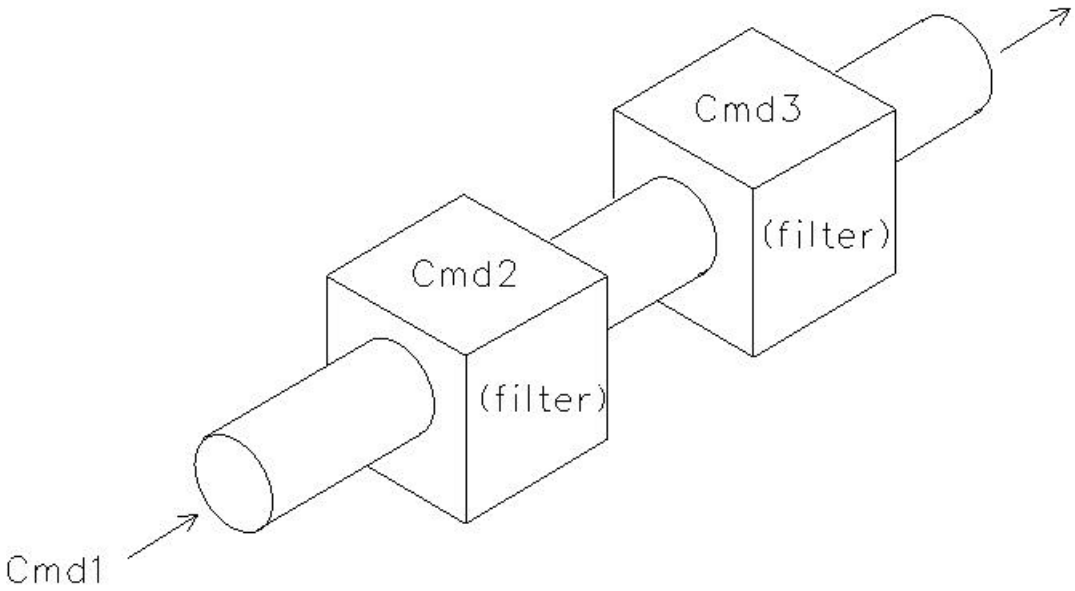


Figure 4-1. Flow Through a Pipeline

A ***filter*** is a command that reads its input from standard input, transforms that input, and then writes the transformed input to standard output. Filters are typically used as intermediate commands in pipelines--that is, they are connected by a ¦ (pipe) operator, for example, **ls -R¦pg** (the **-R** flag causes **ls** to list ***recursively*** the contents of all directories from the current, or named, directory to the bottom of the hierarchy).

Certain commands that are not filters have a flag that causes them to act like filters. For example, the **diff** (compare files) command ordinarily compares two files and writes their differences to standard output. The usual format for **diff** follows:

diff file1 file2

However, if you use the **-** (hyphen) flag in place of one of the file names, **diff** reads standard input and compares it to the named file.

In the following pipeline, **ls** writes the contents of the current directory to standard output; **diff** compares the output of **ls** with the contents of a

file named **dirfile**, and writes the differences to standard output one page at a time (with the **pg** command):

```
ls ¦ diff - dirfile ¦ pg
```

In the next pipeline example, the standard output of **ls -l /** (a long listing of files in the root directory) is piped through another program called **grep**. In this case, **grep** acts like a filter. The **grep** command is useful for searching a text file for any specified string. In the example, **grep r-x** searches the output of the **ls** command for the string **r-x**. Recall what the output of the **ls -l** command looks like:

```
$ ls -l

total 136
-rw-rw-rw-  1 user1  group1  89408  Feb 28  10:31  .login
drwxr-xr-x  2 user1  group1     32  Dec  9  14:52  News
-rwxrw-rw-  1 user1  group1   5104  Mar  8  15:29  act
-rwxr-xr-x  2 user1  group1     64  Feb 15  15:33  flow
-rw-r-----  1 user1  group1   1905  Mar  6  11:16  bnu.tcp.ate.di
-rw-r--r--  1 user1  group1    661  Mar  6  14:45  conference.pla
-rwxr-xr--  1 user1  group1  65536  Mar  1  10:21  core
-rw-------  1 user1  group1    401  Mar  6  14:17  header.info
-rw-rw-rw-  1 user1  group1    510  Mar  4  13:12  joe.msg
-rw-r--r--  1 user1  group1      0  Mar 10  17:24  ls.fyl
-rwxr-xr-   1 user1  group1   3913  Jan 24  14:01  makestuff
-rw-rw-rw-  1 user1  group1    657  Mar  8  15:57  manage.mes
-rw-r--r--  1 user1  group1    483  Mar  6  14:29  marianne.reply
```

Any file with **r-x** in its listing has **read** and **execute** permissions, but not **write** permissions, set for either owner, group or others. Therefore, it must be an executable file (a program). All such lines found will then be piped into the **wc** program, which, without any arguments specified, will give a count of items, words and characters.

The purpose of this pipeline is to count the files in the root which have permission settings of **read**, and **execute**, but not **write**. In the example, it will find programs called **act, flow, core,** and **makestuff**. It will also find a directory called **News**.

The complete pipeline and its output look like this:

```
$ ls -l / ¦ grep r-x ¦ wc
      4      36       159
$ _
```

To get the same results without using a pipeline, you would first have to direct the output of **ls -l/** to a file (for example, **ls -l / >file**). Next, you would have to use that file as input for **grep r-x** and redirect the output of **grep** to another file (for example, **grep r-x <file >file.0**). Finally, you would have to use the output file of **grep** as input for **wc** (for example, **wc <file.0**). Using a pipeline is a much more efficient way to accomplish this operation.

Pipelines operate in one direction only (left to right), and all processes in a pipeline can run at the same time. A process pauses when it has no input to read or when the pipe to the next process is full. The process continues until it receives an END OF FILE keystroke sequence from the pipe.

4.4.2 Using Multiple Commands and Command Lists

The shell usually takes the first word on a command line as the name of a command, and then takes any other words as arguments to that command. However, the operators shown in Table 4-2 give you five different ways to use more than one command on a single command line:

Table 4-2. Multiple Command Operators

Operator	Action	Example
;(semicolon	Causes commands to run in sequence.	**cmd1;cmd2**
&	Causes command to run in the background. (Described under "Running Background Processes" in topic 4.3.6.)	**cmd1>file &**
¦	Creates a pipeline. (Described under "Using Pipes and Filters" in topic 4.4.1.)	ls¦wc
&&	Runs the second command if first command succeeds.	**cmd1&&cmd2** (See Appendix A.)
¦¦	Runs the second command if the first command fails.	**cmd1¦¦cmd2** (See Appendix A.)

Subtopics
4.4.2.1 Separating Commands on the Same Line with a Semicolon (;)

4.4.2.1 Separating Commands on the Same Line with a Semicolon (;)

command separator

You can type more than one command on a line if you separate commands with the **;** (semicolon). In the following example, the shell runs **ls** and waits for it to finish:

```
$ ls ; who ; date ; pwd
change     file3     newfile
username      console/1      Jun 5 14:39
Wed Jun 5  14:42:51  CDT  1985
/u/username
$ _
```

When **ls** is finished, the shell runs **who**, and so on through the last command.

To make the command line easier to read, you can separate commands from the **;** (semicolon) with blanks or tabs. The shell ignores blanks and tabs used in this way.

4.4.3 Grouping Commands

The shell provides two ways to group commands, as shown in Table 4-3.

Table 4-3. Command Grouping Symbols

Command Grouping Symbol	**Action**
( ) (parentheses)	The shell creates a subshell to run the grouped commands as a separate process.
{ } (braces)	The shell runs the grouped commands as a unit.

Subtopics
4.4.3.1 Using ( ) (Parentheses)
4.4.3.2 Using { } (Braces)

4.4.3.1 Using ( ) (Parentheses)

When parentheses are used as a command grouping symbol, the shell creates a subshell to run the grouped commands as a separate process.

In the following example, the shell runs the commands enclosed in **( )** (parentheses) as a separate process:

```
$ (cd x;ls);ls
$ _
```

The shell creates a ***subshell*** (a separate shell process) that moves to directory **x** (**cd x**) and lists the files in that directory (**ls**). The first shell does not change directories. After the subshell process is complete, the shell lists the files in the current directory (**ls**).

If this command were written without the **( )**, the original shell would move to directory **x**, list the files in that directory, and then list the files in that directory again. There would be no subshell and no separate process for the **cd x;ls** command.

The shell recognizes the **( )** wherever they occur in the command line. To use parentheses literally (that is, without their command-grouping action), quote them by placing a **** (backslash) immediately before either the **(** or the **)**, for example, **\(**.

For more information on quoting in the shell, see "Quoting" in topic 4.4.4.

4.4.3.2 Using { } (Braces)

When commands are grouped in **{ }**, the shell executes them without creating a subshell. In the following example, the shell runs **date**, writing its output to the file **today.grp**, and then runs **who**, writing its output to **today.grp**:

```
$ { date; who; }>today.grp
$ _
```

If the commands were not grouped together with braces, the shell would write the output of **date** to the display and the output of **who** to the file.

The shell recognizes **{ }** (braces) in pipelines and command lists, but only if the left brace is the first character on a command line.

For other meanings of braces in the shell, see "Using Braces as Delimiters" in topic A.3.1.3.

4.4.4 Quoting

Reserved characters are characters such as **< > ¦ & ?** and ***** that have a special meaning to the shell. "Shell Reserved Characters and Words" in topic A.10 lists all the shell reserved characters. To use a reserved character literally (that is, without its special meaning), quote it with one of the three shell quoting conventions, as shown in Table 4-4:

Table 4-4. Shell Quoting Conventions

Quoting Convention	**Action**
\	(backslash) Quotes a single character.
\\	(backslash backslash) Quotes a backslash.
' '	(single quotes) Quote a string of characters (except the ' itself).
" "	(double quotes) Quotes a string of characters (except **$**, `, and \).

Subtopics
4.4.4.1 Using the Backslash
4.4.4.2 Using Single Quotes (' ')
4.4.4.3 Using Double Quotes (" ")

4.4.4.1 Using the Backslash

To quote a single character, place a **** (backslash) immediately before that character:

```
$ echo \?
?
$ _
```

This command returns a single **?** character.

4.4.4.2 Using Single Quotes (' ')

When you enclose a string of characters in single quotes, the shell takes every character in the string (except the ' itself) literally.

The following example shows how single quotes can be used in the arguments for a command:

```
$ echo 'x > y+0'
x>y+0
$ _
```

The **echo** command returns the string **x>y+0** because the single quotes remove the special meaning of the **>** reserved character.

You also can use single quotes when you assign values to variables. For information about using single quotes with variables, see "Single Quotes in Variable Assignments" in topic A.3.1.4.

4.4.4.3 Using Double Quotes (" ")

Double quotes provide a special form of quoting. Within double quotes, the reserved characters **$**, ` (grave accent), and **** keep their special meanings. The shell takes literally all other characters within the double quotes. Double quotes are most frequently used in variable assignments. For more information about using double quotes with variables, see "Double Quotes in Variable Assignments" in topic A.3.1.4.

4.4.5 Matching Patterns

The shell gives you five different ways to match character patterns, as shown in Table 4-5:

Table 4-5. Shell Pattern-Matching Characters

Pattern-Matching Character	Action	Example
*	Matches any string, including the null string.	**th*** matches **th**, **theodore**, and **thermohaline**.
?	Matches any single character.	**304?b** matches **304Tb**, **3045b**, **304Bb**, or any other file name that begins with **304**, ends with **b**, and has one character in between.
[...]	Matches any one of the enclosed characters.	**[AGX]*** matches all file names in the current directory that begin with **A**, **G**, or **X**.
[.-.]	Matches any character between the enclosed pair, including the pair.	**[T-W]*** matches all file names in the current directory that begin with **T**, **U**, **V**, or **W**.
[!...]	Matches any single character except one of those enclosed.	**[!abyz]*** matches all file names in the current directory that begin with any character except **a**, **b**, **y**, or **z**.

Subtopics
4.4.5.1 Naming Files with Pattern-Matching
4.4.5.2 Example of Using Pattern-Matching Characters

4.4.5.1 Naming Files with Pattern-Matching

Commands often take file names as arguments. To use several different file names as arguments to a command, you can type out the full name of each file, as the next example shows:

```
$ wc first.t second.t third.t fourth.t fifth.t
$ _
```

However, if the file names have a common pattern (in this example, the **.t** suffix), the shell can match that pattern, generate a list of those names, and automatically pass them to the command as arguments.

The ***** matches any string of characters. In the following example, the name of every text file in this directory includes the suffix **.t**. (This directory contains the same five files shown in the previous example: **first.t . . . fifth.t**.)

```
$ wc *.t
```

The ***.t** matches any file name that begins with a string and ends with **.t**. The shell passes every file name that matches this pattern as an argument for **wc**.

Thus, you do not have to type (or even remember) the full name of each file in order to use it as an argument. Both commands (**wc** with all file names typed out, and **wc *.t**) do the same thing--they pass all files with the **.t** suffix in the directory as arguments to **wc**.

Note: There is one exception to the general rules for pattern-matching. When the first character of a file name is a period, you must match the period explicitly. For example, **echo *** displays the names of all files in the current directory that do not begin with a period. The command **echo .*** prints all file names that begin with a period.

This restriction prevents the shell from automatically matching the relative directory names **.** (dot, which stands for the current directory) and **..** (dot dot, which stands for the parent directory). For an explanation of relative directory names, see "Using Relative Directory Names (. and .. Notation)" in topic 3.6.3.

In the Bourne shell, if a pattern does not match any file names, the shell returns the pattern itself as the result of the matching operation. For example, if the current directory does not contain any file names that end with **.c**, the command **echo *.c** returns ***.c**. In the C shell, you receive a message that no match was found.

Warning: Pattern matching can be very dangerous when used with the **rm** command. The pattern you enter can easily match more files than you realize and may erase valuable data unintentionally. Before using the **rm** command with pattern matching, use **echo** against the pattern. This shows you exactly which files are going to be destroyed. You may realize that the list includes files you need to save, and you may modify your pattern accordingly.

4.4.5.2 Example of Using Pattern-Matching Characters

You can use the **echo** command to learn how the shell interprets pattern-matching characters. The following example is based on a directory that contains the following files:

part1
part2
part3
pre.txt
post.txt

```
$ echo *
part1  part2  part3  pre.txt  post.txt
$ echo *.*
pre.txt  post.txt
$ echo part?
part1  part2  part3
$ echo ????.???
post.txt
$ echo *[1357]
part1  part3
$ echo [a-o]*
[a-o]*
$ echo [!abc]*
part1  part2  part3  pre.txt  post.txt
$ _
```

Each **echo** command in the example uses one or more pattern-matching characters in its argument and returns different information about the files in the directory. Notice that **echo [a-o]*** does not return any file names because none of the file names in this directory begins with one of the lowercase letters **a** through **o**.

4.4.6 Writing and Running Shell Procedures

Besides running commands from the command line, the shell can read and run commands contained in a file. Such a file, called a ***shell procedure*** or ***shell script***, is a program that you can use alone or as a part of a program written in another programming language, such as C or FORTRAN.

Even if you do not usually write programs in other languages, you may find that shell procedures are easy to develop and that using them can make your work with the operating system more efficient. If you develop programs, shell procedures provide a way to test logic without the time and effort of coding and compiling. Shell procedures are also useful for building program development tools.

In either case, because a shell procedure is an ordinary text file that does not have to be compiled, it is relatively easy to create and maintain.

Note: Some commands or programs are shell procedures. As you become more familiar with writing shell procedures, you may want to study the ones supplied with the system for ideas.

Look first at **/etc/rc** (the procedure that runs automatically when you start the system) and at any of the files containing shell commands that are located in **/bin**, **/usr/bin**, and **/usr/lib/acct**.

Writing and Running a Shell Procedure

1. Use a text editor to create a file of shell and AIX Operating System commands.
2. Use the **chmod** command to give the file **x** (execute) status.

Subtopics
4.4.6.1 Writing a Shell Procedure

4.4.6.1 Writing a Shell Procedure

The first step in writing a shell procedure is to create a file of the commands you need to accomplish a task. Create this file as you would any text file--with **ed** or another editing program.

Shell procedures can contain any system command (described in *AIX Operating System Commands Reference*) or shell command (described under "Shell Control Commands" in topic A.6 and **sh** in *AIX Operating System Commands Reference*).

For detailed information about using the features of the shell to write procedures, refer to Appendix A, "Using Advanced Bourne Shell Features."

4.4.7 Running a Shell Procedure

If you intend to use a shell procedure regularly, you should incorporate the **chmod** command to give it **x** (execute) status. For example, the command **chmod g+x reserve** gives execute status to the file named **reserve** for any user in the group (**g**).

After you give the file **x** status, run the procedure by simply entering its name. Enter the path name if the procedure file is not in your current directory.

For details about the **chmod** command, see "Changing Permissions--The chmod (Change Mode) Command" in topic 3.14.2 or **chmod** in *AIX Operating System Commands Reference*.

If you intend to use the procedure only a few times and then discard it, you do not have to give it execute status. However, to run a procedure that does not have execute status, you first must run the shell command (**sh**).

For example, if the procedure **reserve** does not have execute status, use the command **sh reserve** to run it. If the procedure file is not in your current directory, use its path name rather than its simple file name.

Subtopics
4.4.7.1 Example of Creating a Shell Procedure

4.4.7.1 Example of Creating a Shell Procedure

The following example shows every step required to create the simple shell procedure named **lss**:

Note: A line beginning with a # is a comment line.

```
$ ed
a
# lss: list, sorting by size
ls -s ¦ sort -n
.
w lss
q
$ chmod +x lss
$ _
```

Following is an explanation of each step in the creation of **lss**:

ed

Starts the **ed** line editor.

a

Causes **ed** to add text to the buffer.

 # lss: list, sorting by size

Comment line describing the purpose of the procedure.

ls -s ¦ sort

Enters the text (commands) of the procedure itself.

.

(period) Stops the editor from adding text to the buffer. The period must be entered in the first position on a line by itself.

w lss

Writes (copies) the text from the buffer into the file **lss**.

q

Quits (ends) the editing session.

chmod +x lss

Gives execute status (**+x**) to the file **lss** for all classes of users.

The **lss** procedure first finds the size, in blocks, for each entry in a directory (**ls -s**). Output from the **ls** command is then piped to the **sort -n** command (**¦ sort -n**). The **sort** command then arranges its standard input according to size, and writes the size and name of each file to standard output. To run the **lss** procedure, simply enter **lss**.

Shell procedures are especially useful for routine tasks because they enable you to execute multiple commands by entering only the name of the procedure.

5.0 Chapter 5. TCF Overview

Subtopics

5.1 CONTENTS

5.2 About This Chapter

This chapter describes the Transparent Computing Facility (TCF). Computers linked together with TCF are called ***cluster sites***. A ***cluster*** is a collection of computers that communicate via LAN (local area network). Each cluster site can be identified by its ***sitename*** or by its ***site number***.

Different types of computers may be included in a cluster, including IBM System/370 computers running AIX/370 and other computers capable of running the AIX Operating System. There can be up to 31 such computers linked together in a cluster.

Through the use of TCF, AIX becomes a ***transparent distributed operating system***, treating a collection of computers as a single resource where internal boundaries have no effect on users. A user may invoke the same commands to access data stored on another member of the cluster as he uses to access data stored on his local cluster. This is called ***data transparency***.

In addition to **data transparency**, TCF also provides for ***process transparency***. This is the ability to execute and control tasks on any cluster site, without regard to where the user is logged in or where other processes are running. The user may begin a process on another machine or move a process to the data location and still take advantage of the specialized facilities or services unique to a particular machine.

Many of the examples in this chapter refer to an imaginary cluster comprised of eight sites: **admin1, admin2, accts, adminb, chem, supply, lab, and safety**, with users named **joe, art and ruth**.

5.3 Communicating with Other Systems: An Overview

There are three primary configurations for the AIX Operating System. The first is a version of the AIX PS/2 Operating System, which is a standalone UNIX-based operating system designed to run on a single IBM Personal System/2. This configuration includes only one machine.

Another possibility with the AIX PS/2 Operating System is to link a group of Personal System/2 computers together into a single functional ***cluster*** with shared resources. This configuration can include up to 31 PS/2 computers. Other computers, functioning as ASCII terminals, can be attached as accessory work stations.

This most common configuration includes both PS/2 computers and System/370 machines linked together to form a single network that responds as though it were a single computer. This configuration includes at least one System/370 mainframe with accessory terminals attached to PS/2's functioning as work stations. This configuration requires both the AIX/370 and AIX PS/2 operating systems designed to run in a ***distributed processing*** computer network environment.

If your work environment fits the second or third description, you are interacting with other computers in an arrangement where all the machines share resources. The nucleus of such a network is the ***host computer***(s). These central units may be System/370 mainframes or PS/2's tied together in a Local Area Network (LAN). A ***cluster*** refers to all the machines which are serving as hosts in a single network (as many as 31 machines can be included) and any satellite machines connected to these hosts. These computers include the **TCF** (Transparent Computing Facility) option. TCF is also available as an optional LPP (Licensed Program Product) for the standalone version of the AIX PS/2 Operating System.

Any of the host computers can have other dependent display stations attached. These ***terminals*** include a wide variety of machine types, but they are all either ASCII terminals or other machines emulating ASCII terminals.

The AIX Operating System contains several communication facilities that help you transfer information electronically from one location to another. Depending on the facility, this transfer can be within an AIX cluster or between clusters, and involve any of the following:

Two terminals or work stations within an AIX cluste

Two AIX cluster

A display station and an AIX cluste

An AIX cluster and another, remote network

Note: Modems and cables, discussed in *Managing the AIX Operating System*, are required for some facilities.

Ask your system administrator which of the facilities discussed in this section are available to you, then select the facility best suited to each of your communication tasks. If you need basic information about electronic communication, refer to the IBM publication titled *Data Communications Concepts* (GC21-5169).

Subtopics

5.3.1 Cluster Communication

Sending messages to others on your local system is called ***cluster communication***. You may send and receive messages to and from other users and find out who is currently working on your cluster with the **talk**, **write**, and **mesg** and **who** commands.

To use these commands, and to use this simple communication facility on your cluster system, refer to Chapter 7, "Using Cluster Communications Facilities."

For more information about cluster communication, and for information about the Message Handler, see *Managing the AIX Operating System*.

5.3.2 Cluster File Access with the Transparent Computing Facility (TCF)

TCF ties the cluster hosts together in such a way that the entire resources of the cluster appear to the user as a single computer. The user does not need to know where his files are stored or which programs are kept on his own network host. The user is free to run any program for which there is proper permission against any data file for which there is proper permission.

Under TCF, a user may also use a display station to log on to one host and then begin a new shell process on another host in the same cluster. The user may physically move to another terminal which is connected to the cluster. The file system and the computing environment will appear the same as it did when the original terminal was used. All programs have the same names and all the data files appear to the user to be positioned in the same places on the root file system.

The programs may actually be different. They may be special versions compiled to run on different hardware. The data files, too, may not be original, but copies automatically created by the system. The user, however, always receives the same view of the system.

5.3.3 Mail

The **mail** program handles both cluster communications and messages to other sites outside the cluster. To send mail to someone within the cluster, use the following command:

mail **username**

To send mail to a user outside the cluster, you must specify an address with the **username**. A command of the following format is required:

mail **username@machinename**

The **mail** program uses **sendmail** as its routing and delivery agent. The **sendmail** program is smart enough to use whichever communications protocol it needs to send the message where it needs to go. In most cases this will be the BNU protocol. The **mail** program handles the details transparently. All the user needs to know is how to address the mail.

5.3.4 Remote File Access via NFS Mounts

AIX supports the ***IBM AIX Network File System (NFS)***, that lets you create and access the files located on other machines in the network from your local AIX cluster. Unlike remote copy or transfer facilities, NFS allows you to access entire file systems, and read and write to the files directly instead of reading and writing to copies. This means you can share files with other users as well as access additional disk space. You can use the same commands to work with the remote files that you use when working with the files that exist on your local system.

NFS allows users working on AIX display stations to create and access the files located on other computers in the network, even if they are not AIX computers. Included with NFS is a software component called the Network Information Service (NIS) that your system administrator uses to administer system information, such as passwords and machine names. For detailed information about NFS and NIS, see *Managing the AIX PS/2 Operating System*.

When NFS is installed, your system administrator ***mounts***, or makes available, on your computer sets of commonly used remote file systems. This means that you automatically have access to the remote files when the system starts just as you do the local files on your work station. If you have the appropriate authority and permissions, you can access other remote files directly using the **mount** command. See the **mount** command in *AIX Operating System Commands Reference* for detailed information and conditions.

The IBM Network File System licensed program is on individual diskettes that are purchased in addition to the software for the AIX Operating System.

> For detailed information about installing NFS, refer to *AIX/370 Installation and Customization Guide*.
>
> For detailed information about configuring NFS, refer to *Managing the Operating System*.

5.3.5 TCP/IP (Transmission Control Protocol/Internet Protocol)

TCP/IP is a communications protocol designed to interconnect a wide variety of computer equipment. It is available as a licensed program purchased separately.

TCP/IP was developed by the Department of Defense and adapted for use in the Ethernet LAN (Local Area Network). An important feature of TCP/IP is TELNET (**tn, tn3270**) which allows a user to log on to and interact with a wide variety of host computers. TCP/IP is most appropriate for short to medium distance communication. The AIX Operating System uses TCP/IP to bind host computers together into a cluster. It may also be used to send and receive files from other "near-remote" systems (systems in the same building, for example).

TCP/IP provides terminal emulation and file transfer between machines which may or may not be UNIX machines. You can send files to and receive files from a VM system, a DOS machine, or any system that offers TCP/IP. It runs over local or wide area networks, but it requires a dedicated line and it will not run over serial lines.

To transfer files from a near-remote system to your own directory, you can use the **ftp** (file transfer program) from System V or the **rcp** command from BSD4.3. To log on to a remote computer, use the System V TELNET (**tn, tn3270**) command or the BSD4.3 equivalent, **rlogin**.

This set of communications protocols is used extensively by AIX. The computer serving as host sites in any single cluster are tied together by LAN lines and communicate among themselves using TCP/IP packets sent over these lines. Most of the commands that users employ to manipulate files within the cluster move those files via TCP/IP.

These functions can run over phone lines, but typically they run over Local Area Network coaxial lines. They are usually used for communications between "near remote" systems; that is, separate networks which are physically close together. Two AIX clusters within the same building would be one example.

TCP/IP functions are fast, but they require a high-throughput hookup, like a LAN connection or a dedicated hardware line. Such lines are relatively expensive to maintain over long distances, so TCP/IP functions are most often used for connection between systems that are fairly close together physically.

These functions allow the user to establish a remote session on other machines outside the cluster and manipulate files from that position. Meanwhile, the user remains logged into his original host. If the new machine is a host on some other AIX cluster, he must also go through a login process on the new host. During the time he is logged in to the second site, he can manipulate files there, copy them to other machines, send them back to his original site and bring files over from his original site to the second site. When the user exits the remote session, he automatically returns to his original host where he is still logged on.

The AIX Operating System includes functionality from both of the major traditions of UNIX usage; AT&T's System V and Berkeley's BSD4.3 UNIX. Programs from both do similar jobs, differing primarily in program names and options offered. The System V versions of the TCP/IP programs offered by AIX are:

telnet (tn, tn3270) Allows users to create a remote session on a machine outside the cluster.

ftp File Transfer Protocol. This program moves files between sites which are not part of the same cluster. FTP gives the user a remote session on a remote site, after which he has reduced functionality available, restricted to file manipulations. This set of functions includes many of the capabilities that are performed by individually named commands in the Berkeley system.

The following programs are the BSD equivalents of the above. They are known as ***r commands***. The ***r*** stands for remote.

rlogin Equivalent to the **telnet** command. Opens a session on a remote site outside the cluster.

rcp Remote Copy Program. Copies files directly to and from a remote site.

rsh Remote Shell. Starts a new shell process for the user on a remote site. He may use that shell to run any other standard AIX commands. This can be done either interactively or non-interactively.

If the **rsh** command is given with a host name as one of its arguments and another AIX command as a second argument, the indicated command is run on the indicated host and the user finds himself back on the original host. This is a non-interactive usage with the following format:

```
rsh hostname command
```

If only a host name is given, the user gets an interactive session on the remote host, equivalent to the **rlogin** command. The format is:

```
rsh hostname
```

For further information about TCP/IP functions, see Chapter 9, "Using TCP/IP."

5.3.6 Remote Communications

An AIX cluster also behaves like a single computer system when it communicates with anyone outside the cluster. All users within the cluster are considered local users. All users outside the cluster are considered remote users.

When you communicate with remote systems you use procedures which closely parallel the methods used within the cluster, but are not entirely the same. In general, remote file transfers require more effort from the user. To access and transfer files from a remote system to your own directory, you have to name the file, the system on which it is stored, and the directory in which it resides. In some cases you may also need to type in a rather complex pathway by which the contact with this remote system is made.

You must also consider the operating system being run by the system you are communicating with, and the nature of the lines along which the communication travels. When copying blocks of files to and from remote systems, you need different utilities to communicate with different systems.

The ***mail program***, however, takes care of these details via software. You tell the program the name and system of the person you want to send a message to, and **mail** handles your communication. The mail program operates similarly both inside and outside the cluster. In most other cluster-to-remote system communications, the programs you use are different from those you use within the cluster, and the procedures, though similar, will be a bit more involved.

The AIX Operating System supports a wide range of ***communications protocols***, or formats to exchange information with other computer systems. Following is a brief overview of some of the communications protocols you may be using.

5.3.7 Basic Networking Utilities (BNU) Program

The Basic Networking Utilities (BNU) program is IBM's bundled package of utilities used to communicate with remote UNIX systems. In AT&T's form of UNIX, the same package is called **uucp**. BNU provides terminal emulation and file transfer between UNIX machines. The AIX Operating System uses BNU for long distance transfers between your cluster and any remote UNIX system. BNU uses whatever communications protocol is necessary to contact the machine you request. It can run over serial lines (like telephone lines) or it can use TCP/IP to run over local or wide area networks. BNU does not require a dedicated line.

For information about using BNU, refer to Chapter 8, "Using the Basic Networking Utilities." For information about customizing BNU, refer to *Managing the AIX Operating System*.

5.3.8 The uucp Protocol

UNIX to UNIX Copy (**uucp**) is another networking protocol used to send files from one UNIX system to another. This is a straightforward copy operation, so any sort of file can be sent, including program files, text files, and condensed data in binary format. One significant use of this function in the AIX Operating System is sending text files as mail messages from one AIX cluster to another when the systems are separated by considerable distance. It is equally useful for mail to other, non-AIX UNIX systems.

The **uucp** protocol is a modem-driven, serial transmission function. It is relatively slow compared to LAN transmissions, however it is reliable and usable over ordinary phone lines. Consequently, it is used almost exclusively for long-distance file transfers.

In the AIX Operating System, the **uucp** facility is called **bnu**, which stands for Basic Networking Utilities. The **uucp** protocol is actually a family of programs, most beginning with the letters "uu", which transmit, query and manipulate files between UNIX systems.

These programs include the following:

uucp Copies files to and from other UNIX systems. Used in a system where security of files is not a problem. In such a situation, the **uucp** protocol could copy files from a remote host directly into any directory in the AIX cluster.

uuto Copies files from a remote UNIX system to a special "public directory". From here, each user must move them to his own directory with the next command. Used in a security-conscious environment where the system administrator wishes to restrict system access. Users on the remote system have access only to the "public directory"; the rest of the cluster is beyond their reach.

uupick Used to "pick" files out of the "public directory" and move them to the user's own directory.

uux Executes a command on a remote UNIX machine. Similar in function to the **rsh** command.

uuname Gives the user a list of remote sites he may access.

uustat Shows the status of jobs given to the "uu" system. Since the request for a file transfer is usually made before a connection is actually established with the remote system, the user often wants to know whether a particular request has yet been serviced.

cu Stands for Call UNIX. Established a connection with a remote system via a serial port, and then provides many simple functions for manipulation of files on the remote system and on the users login site.

Full information on using any of these commands may be found in Chapter 8, "Using the Basic Networking Utilities." For information about customizing BNU, refer to *Managing the AIX Operating System*.

5.3.9 Asynchronous Terminal Emulation (ATE)

The Asynchronous Terminal Emulation (ATE) facility is used for connecting with AIX via telephone lines. Like BNU, ATE provides terminal emulation and file transfer between UNIX-style machines. It can be run over serial lines, but not over local or wide area networks. The main feature of ATE is enhanced terminal emulation support, allowing one terminal type to emulate another.

You may use ATE to log into your AIX system via phone line from your home, or you may use it to communicate to another system which offers only ATE connection.

For information about customizing ATE, refer to *Managing the AIX Operating System*.

5.4 Base AIX File Systems

Each TCF or AIX site is called a cluster site. Each cluster site has at least two separate file systems:

/ (slash) which represents the replicated root file syste

/ **sitename** which represents the **<LOCAL>** file system.

Subtopics
5.4.1 The Replicated Root File System
5.4.2 The <LOCAL> File System
5.4.3 User File Systems

5.4.1 The Replicated Root File System

When any cluster site is brought up, the AIX Operating System automatically installs the root directory and its immediate branches. This entire structure, consisting of many directories and their dependent files, is replicated on every cluster site. These directories hold files which are critical to system operation, such as the file which holds the AIX kernel itself. One example of such a directory is **/bin**, which holds most of the binary files for the system utilities.

Each cluster site has one copy of the ***replicated root file system***. When multiple AIX sites are connected into a cluster, the operating system replicates this root file system on every site. Since these are the files from which all system functionality is derived, all sites function in an identical manner. This enables the cluster to emulate a single computer.

The operating system assumes the task of seeing that all copies of replicated files on all sites are kept up-to-date and identical. For example, the **/etc/passwd** file is part of the **/etc** directory, which is part of the replicated root file system. Therefore, **/etc/passwd** appears on every site in the cluster. The system reads this file every time a user logs on to verify that the password he has given matches that assigned to him. When the system administrator updates this **/etc/passwd** file to issue a password to a new user, the operating system takes care of replicating the file so that all copies on all sites are identical.

There is one ***primary file system*** stored on the primary site of the cluster from which all other copies are made. Only the primary file system can be modified. All secondary copies are read-only. There will be occasions when the primary site becomes unavailable. During its absence from the cluster, certain activities cannot take place. If the system administrator cannot access the only modifiable copy of the **/etc/passwd** file, he cannot issue any new passwords. Users may still log on and off the host machines that remain active because the copies of **/etc/passwd** can still be read.

Subtopics
5.4.1.1 Advantages of File System Replication

5.4.1.1 Advantages of File System Replication

Replication allows cluster sites to run independently while presenting a uniform view to all users. The AIX/370 Operating System uses replication as an integral and indispensable part of TCF operation. Replication allows the sites in a cluster to act together as a single unit. It also provides a duplicate environment on every site so that each site has the files it needs to run independently if necessary. If the site where certain files are normally stored becomes unavailable, the user can often continue to work by accessing copies of those files stored elsewhere.

Replication provides an environment that looks nearly identical to the user, whether the user's login site is operating as a standalone computer or as one of the cooperating hosts in an AIX cluster.

Each cluster site contains the system files it needs to run alone, even if all other sites should leave the net. Some of these system files are contained in the replicated root system, and others are found in the local file system. Thus, all cluster sites have the ability to run cooperatively or in a standalone environment. This replication feature allows every cluster site to go up and down without affecting the operation of the cluster.

Another advantage of replication is the ability to store copies of each user's files on various sites in the cluster (if the system administrator chooses to do so). These user replicated files are read-only; only the primary copy of the user's file system is modifiable.

For example, **art** has his home directory on **admin1** and a copy of his file system on **admin2**. He tries to log on to **admin1** and finds it unavailable. He can still log on to **admin2** and read his files, but he cannot write to them directly. If, however, **art** needs to modify some of those files and cannot wait for **admin1** to be brought back up, he can copy the particular files he needs to change and then modify the copies. When **admin1** is brought back up, he can copy the modified files from **admin2** to the originals on **admin1**. He does not need to alter the files on **admin2** from which he made the copies. Those are still in the old, unchanged form, but standard system replication takes care of updating his files on **admin2** so that they match.

Whenever files or directories are added to the replicated root file system, the system must be told on which sites those files are to be replicated. This is done by setting the **fstore** value with the **chfstore** command. For more information about the **chfstore** command, see *AIX Commands Reference*.

5.4.2 The <LOCAL> File System

The **<LOCAL>** file system is named for a particular cluster site whose files it holds (see "The <LOCAL> Alias" in topic 3.10.6). It includes site specific information such as accounting and administrative files to track system usage and printer spool files for that site.

The **<LOCAL>** file system may also contain the binary files for the application programs specific to that site. One popular strategy of system administration is to assign all members of a particular work group to the same host as their regular logon site. In this case, the application programs needed by those workers will probably be stored on that host, and the data files with which they usually work may be stored there too. There may be copies of all these files elsewhere on the network so they can work when their usual host is down, but the primary copy of the file system they share will probably be the **<LOCAL>** file system of that machine.

For example, the **<LOCAL>** file system for a work group composed of writers might include word processors, spell checkers, a thesaurus program and text files.

5.4.3 User File Systems

In a TCF cluster, your system administrator may have set up the file systems that store users' home directories on a number of different sites. It is usual to spread these file systems evenly over the available processors rather than gather them all on a single site. These ***user file systems*** may have names like **/u1**, **/u2**, and **/u3**. To find out which file system your home directory belongs to, use the **pwd** command. To find out which machine it is stored on, use the **where** command.

5.5 Using TCF Commands

There are a number of useful TCF commands that allow you to access information about other sites in the cluster. These commands identify other types of computers, tell you where certain files are stored, how busy a certain cluster site is, and which cluster site is currently the least busy.

Subtopics

5.5.1 Identifying Sites in a Cluster--the site Command

Two common uses for the **site** command are identifying the site where you are currently logged on and identifying the names of all the sites in the cluster that are currently in operation. Flags to the **site** command include the following:

Option	***Function***
-t	Determines the availability of cluster sites.
-a	Names all sites that are members of the cluster.
-p	Identifies cluster sites of the indicated computer type.

To find which sites on a cluster are a particular type of computer, use the **site -p** command. You must specify the type of computer you are interested in after the **-p** For example, to find which sites on your cluster are PS/2 sites, issue the following command:

```
site -pi386
```

5.5.2 Locating a File or Directory--the where Command

The **where** command gives status information and a list of sites where the given file is stored, or where a file can be stored. To determine the location of a file or directory, give the **where** command followed by a **filename**.

The **where** command is especially useful when you know that a site in the cluster will be unavailable for a period of time and you want to continue working with a certain file. Using **where** allows you to find out if you should make a copy of that file on a site that will still be in operation.

For example, **ruth** uses the **where** command to tell her where her home directory, **/u1/ruth**, is stored. She can make a copy of the file or files she needs on another cluster site, to be used in the event that her normal cluster host becomes unavailable.

```
$ where /u1/ruth
ur m /u1/ruth
           safety
           lab
           admin1
```

The information **ruth** obtained from the **where** command tells her the following:

ur means that it is a user replicated file system.

m means it is modifiable.

[MP] means that **/u** is a file system.

The list of **safety, lab, and admin1** shows that copies of the file system are stored at these locations.

If **safety** becomes unavailable, **ruth** can access the files in **/u** from **lab** or **admin1**.

5.5.3 Checking Site Load Information--the loads Command

The **loads** command displays the load average of each currently participating site in the cluster. It provides the name and number of each site, the number of users logged in, the 1-minute, 5-minute and 15-minute load averages, and the length of time the cluster site has been available.

The load average is a way of telling how busy the cluster site is. It is the average number of programs competing for use of the computer simultaneously. **art** uses the **loads** command to display the load averages of each site in the **research** cluster:

```
admin1 [site 1]  Users:21   Loads:0.56 1.01 1.24   Up:12 days 14:33
admin2 [site 2]  Users: 9   Loads:2.60 3.27 3.33   Up:13 days 11:23
lab    [site 4]  Users: 2   Loads:0.41 0.36 0.35   Up:10 days 23:02
adminb [site 5]  Users: 1   Loads:0.32 0.27 0.28   Up:10 days  2:23
chem   [site 6]  Users: 3   Loads:1.41 1.95 1.84   Up: 7 days 10:01
accts  [site 7]  Users: 1   Loads:0.35 3.03 0.92   Up:17 days  9:15
safety [site 8]  Users:10   Loads:2.51 1.20 0.89   Up: 9 days  1:08
```

The **loads** command ignores cluster sites that are unavailable. In the example above, all of the sites in the **research** cluster are displayed except for **supply**, which is not available.

art needs to format and print a report, and he would like to run the job on the machine with the lightest load. He looks at the load averages provided by the **loads** command and determines that **adminb** has the lightest load. Although **art** is logged on to **safety**, he will run his job on **adminb**.

5.5.4 Finding the Fastest Site--the fastsite Command

The **fastsite** command determines the fastest site in the cluster. It provides the site number of the fastest cluster site by looking at load average values and the CPU performance factor, which is the speed of the computer. **art** uses the **fastsite** command to display the number of the fastest cluster site:

```
$ fastsite
5
```

The **fastsite** command has a -v flag that displays the name of the fastest cluster site:

```
$ fastsite -v
adminb
5
```

The output from the **fastsite** command can be used with non-local sites; that is, sites within the cluster but other than the one where the user is logged on. The commands that do this non-local program execution are the **on** command and **migrate** command. The **on** command starts a process on any site in the cluster other than the one where the user is logged on. The **migrate** command can be used to move a process from the machine on which it was started to any other site in the cluster.

For example, **art** is logged onto **safety** and wants to run a long, CPU-intensive job. He knows that **safety** is a slow machine, so he runs the **loads** command and sees that **adminb** has the lightest load. He confirms that **adminb** is the fastest site for his job by running the **fastsite** command.

5.6 Running a Job on a Non-local Site

The AIX Operating System allows you to run jobs on non-local cluster sites. Instead of starting a process on your cluster site, which may be heavily loaded at the moment, you can start or move a process to another site in the cluster that has a lighter load.

When too many CPU-intensive programs are running at the same time on a particular site, that host goes down. Rather than adding to the overload by starting another process on your cluster site, you can look for a site with a lighter load and start the process on that site.

The AIX Operating System provides three ways to run jobs on non-local sites: by using the **onsite** command, which starts a process on a particular cluster site, by using the **migrate** command, which moves an already started process from one cluster site to another cluster site, and by using the **fast** alias, which determines the fastest site and starts the job on that cluster site.

Subtopics

5.6.1 The onsite Command

The **onsite** command executes the given command line on the named cluster site. To run a command on a non-local site, give the **onsite** command followed by the site name and the command line you want to execute.

For example, **ruth** is working on **safety** and that host has become quite heavily loaded. She is writing a C program and wants to compile it, but doesn't want to add to the already heavy load on her local site, **safety**. **ruth** uses the **onsite** command to start her compile on another site in the cluster, **admin1**.

```
$ onsite admin1 cc -O price.c -o price &
```

Note: The local module obtained from **cc** can only be run on the same machine type on which the **cc** command was run. For example, if **cc** was run on a PS/2, the result is a PS/2 load module that runs **only** on a PS/2.

For command lines of the form:

```
$ onsite site_name command_line
```

everything will be evaluated and executed on the non-local site, the site_name named as the first argument to the **onsite** command.

For command lines of the form:

```
$ onsite site_name command_line > out_file
$ onsite site_name command_line >> out_file
$ onsite site_name command_line 2> error_file
$ onsite site_name command_line < in_file
```

the part of the command line before the I/O redirection symbols (>, >>, 2>, and <) is executed on the non-local site. Arguments after the redirection symbol, such as out_file, error_file, and in_file are evaluated on the local site.

For example, **art** is logged on to **safety**. He wants to format his file on **adminb** and put the formatted output into a file, so he types:

```
$ onsite adminb nroff -mm textfile > textfile.out
```

The **nroff** command will be evaluated and executed on **adminb**, the non-local site, while textfile.out will be put on the local site, **safety**.

For command lines of the form:

```
$ onsite site_name command_line | command_line
```

commands after the pipe will be evaluated on the local site, while the commands before the pipe will be evaluated on the named non-local site.

For example, **art** is logged into **safety**. He wants to format his document into a file and then print it using a pipeline, so he types:

```
$ onsite adminb pr -dh "Customer Report" cust.report | print &
```

The **pr** command is evaluated and executed on **adminb**, the non-local site, while the **print** command is evaluated and executed on the local site,

safety.

In the command line:

```
$ onsite accts cat /tmp/testfile > /tmp/testfile
```

the two files called **/tmp/testfile** are different. The first **testfile** is assumed to be in the **/tmp** of the named site, **accts**. The **testfile** after the redirection symbol is assumed to be in **/tmp** of the local site, which is **safety**. In order to avoid confusion, give absolute path names when dealing with non-local processing and files and directories in the **<LOCAL>** file system, such as **/tmp**.

5.6.2 The migrate Command

The **migrate** command causes a process that is running to be moved to another site in the cluster. You can only migrate a process to a computer of the same type as you are currently using. Give the **migrate** command, followed by a dash, the site name, and the process identification number (PID).

art currently has a compile job running on **adminb**. He issues the **fastsite** and determines that the fastest site is **admin2**. **art** needs his job completed as soon as possible, so he migrates the job from **adminb** to the faster site, **admin2**.

The **migrate** command takes two types of arguments. The first is the name of the target cluster site, and the second is the PID of the job to be moved. **art** doesn't remember what the PID of his compile job is, so he'll need to check his processes on **adminb** to find it. **art** uses the **ps** command with the -r option to report on non-local processes. He types the following:

```
$ ps -r
   PID TTY TIME COMMAND
   45 022 2:27 -sh
   758 022 0:13 ps -r
   children on site 5:
   479 ?   1:02 cc -o prog prog.c
```

The PID for his job is 479. **art** issues the following command to migrate his job to **admin2**:

```
$ migrate -admin2 479
```

art can check to see that his process is now running on **admin2** by typing:

```
$ onsite admin2 ps -r
   PID TTY TIME COMMAND
   479 ?   1:02 cc -o prog prog.c
   772 ?   0:13 ps -r
```

Note: Unlike the **onsite** command, the migrate command does not change the name of the **<LOCAL>** file system; the **<LOCAL>** remains the same before and after the process is migrated.

5.6.3 The fast Command

The **fast** command is an alias that determines which cluster site is the fastest and executes the given command on that site. To run a command on the fastest cluster site, give the **fast** command followed by the command line.

art could have used the **fast** command instead of determining the fastest site with the **fastsite** command and then starting the process with the **on** command. The command:

```
$ fast cc -o prog prog.c
```

would have sent the process to run on the fastest site, **admin2**.

Note: By default, both the **fast** and the **fastsite** commands will pick a host of the same machine type as the local site on which the command is run. If the user is logged on to a 386 machine, these commands will find the fastest PS/2 in the cluster. If the user is on a System/370, they will seek out the fastest 370. If the user specifically wants to consider both types of processor and find the fastest site overall, he can use the **-a** option, which is available with both commands.

In the example given above, it is unlikely that the user would want to consider both machine types. A program compiled on a 386 machine will only run on the PS/2. If the user is logged on to a PS/2, it is unlikely that he will want to compile it to a 370. This would produce a module that could only be run on a System/370.

5.7 Topology and its Impact

In data communications, ***topology*** refers to the pattern of interconnection between computers. In an AIX cluster, the term refers both to the way the hosts are connected to each other to form the cluster, and to the way the terminals are connected to the hosts.

The traditional topology for network operation is a number of terminals tied to a single host machine via fixed lines. They may form one of three patterns: a star, a ring, or a bus (straight line).

In these traditional, single-site environments, users have only one host to log on to. When that site becomes unavailable, all resources of the system become unavailable to all users until the host returns to operation.

In an AIX cluster, a variety of topologies may be used. In a traditional setup, a single PS/2 may serve as host for many terminals. More frequently, however, there will be a group of hosts tied together into a Local Area Network. These hosts may all be PS/2's or they may be some mixture of PS/2's and System/370's. These host machines are tied together by one of two means: Ethernet, which is a bus topology, or by Token Ring, which is a ring topology. Once joined, TCF allows this cluster of machines to function in many respects as if it was a single computer.

The display stations these hosts serve can be divided into two categories: ***work stations*** and ***dumb terminals***. They differ in their internal processing abilities and in the ways they are connected to the hosts.

A ***work station*** has some ability to analyze whether its messages are being properly received by the intended host and to retransmit those that have not been acknowledged. These smart terminals are usually tied into the cluster by joining them to the Ethernet or Token Ring lines. In many cases, users work at machines tied into the network communication lines, not directly to any individual host. In this situation, each user makes a choice as to which host to log on to at the beginning of each session. If the cluster site on which the user is working becomes unavailable, the user is free to log on to one of the other sites and continue his work. The user may lose access to certain files if they were stored on the host that went down and not replicated elsewhere, but he need not be completely blocked.

In the AIX cluster environment, sites may arrive and leave the cluster without significant impact on the cluster's operation. In most cases, users will experience minimal disruption of their work.

Dumb terminals have no such ability to analyze whether communications are being properly received. They simply send their data in the form of a serial transmission. These terminals are connected directly to one particular host. ASCII terminals (or other machines emulating ASCII terminals via software) must be tied to one of the PS/2 hosts of the cluster by dedicated RS-232 serial line. Direct serial connection to System/370 machines is not available.

Users on dumb terminals must log onto their own host at the start of each session. Once the session is started, they are free to log onto any of the other hosts for which they have permission through the use of the **telnet** (**tn**) or **rlogin** commands. These users are totally dependent upon their own host for the stability of their session. If that host machine becomes unavailable, their connection with the network is broken and work

cannot proceed, even if the user has logged onto another network host.

Cluster topology can be examined with the **site** command. When used with the **-t** flag, the **site** command lists the names of the cluster sites that are currently available. For example, there are eight sites in a particular cluster: **safety, accts, adminb, chem, lab, supply, admin2,** and **admin1**. Currently, three of the sites are unavailable. By using the **site -t** command, **ruth** can see the current cluster topology:

```
$ site -t
accts adminb chem supply admin1
```

Subtopics
5.7.1 Failure Conditions
5.7.2 When the Local Cluster Site Becomes Unavailable
5.7.3 When a Non-local Cluster Site Becomes Unavailable
5.7.4 Cluster vs. Standalone Operation

5.7.1 Failure Conditions

The topology of a cluster changes when sites become available or unavailable, or when cluster communication fails. A cluster site may become unavailable for reasons such as performing scheduled maintenance, adding new hardware to the system or system malfunction. When this happens, the cluster loses the resources provided by that cluster site.

When a cluster site becomes unavailable due to system malfunction of that host, the files that are stored there are also unavailable and the processes that are running on that site's CPU are terminated. When a cluster site becomes unavailable due to a communication failure such as a broken wire, that host remains operational but the files that are stored there are no longer available. The CPU that is running the processes continues to run, but if a process running on that site depends upon the resources on another cluster site, the process will be unable to complete successfully.

Note: While the topology of the cluster is changing, some activities are halted; when the topology of the cluster stabilizes, these activities resume.

5.7.2 When the Local Cluster Site Becomes Unavailable

There are two useful terms to describe the sites in an AIX cluster: the **local cluster site**, which is the site you're logged into, the **non-local cluster sites**, which are the other sites in the cluster.

When the local cluster site halts, resources vanish immediately. In order to continue working, users can log into other sites in the cluster (as long as they can connect to another cluster site and have the permissions to do so). When a user logs in, he may notice that some files and directories are no longer available because they are stored on a cluster site that is unavailable.

For example, **joe** is logged into his home site, **admin1**, and it becomes unavailable. He decides to log into **accts** and work there until **admin1** returns to the cluster. When he does so, he notices that he cannot access any files in his home directory, because they are stored only on **admin1**. When **joe** logged into **accts**, the AIX Operating System told him that he would be unable to access his login directory, **/u1/joe**. Instead, the system logged him into **/**. **joe** can now work on other files until **admin1** returns to operation, at which time he can access his home directory.

Not only can a topology change affect the files a user can access, but it can also affect running processes. For example, when **admin1** becomes unavailable, the sorting program **joe** is running is terminated. When he logs into **accts**, he can restart the sorting program if the files he was sorting are available.

5.7.3 When a Non-local Cluster Site Becomes Unavailable

When a non-local cluster site becomes unavailable, so do the resources from that site. All of the system and user processes running on that cluster site are unreachable, and any files that are stored only on that cluster site are unavailable. The user may notice that some files and directories are no longer available because those files and directories are stored only on the site that is unavailable.

For example, **ruth** is logged in to her home site, **admin1**, and **lab** becomes unavailable. **lab** contains the primary copy of the replicated root file system (the system files and programs that are common to all sites in the cluster). **ruth** was in the process of changing her password with the **passwd** command when **lab** became unavailable:

```
$ passwd
passwd:  Password file cannot be written (primary site is down)
```

Since **lab** is unavailable, the primary copy of **/etc/passwd** cannot be changed, and **ruth** receives the message from the AIX system. **ruth** must wait for **lab** to return before she can successfully change her password.

When a user is running a process on a non-local site and that site becomes unavailable, the process is terminated. A C-shell user is notified of this event by a message. For example, **art** is formatting a large document non-locally on **lab**. When **lab** becomes unavailable, the process is terminated. **art** receives a **site down** message and realizes that his formatting process has been terminated. He can now re-execute the process locally (on the site he is logged onto) or on another non-local site that remains in the cluster.

If **lab** is unavailable because of a communication failure, the document continues to format correctly as long as the files it is accessing are available to the processor on **lab**. This is true if those files are stored on **lab** or if **lab** manages to maintain its connection to the host where they are stored.

5.7.4 Cluster vs. Standalone Operation

The Transparent Computing Facility is probably the strongest single feature of the AIX Operating System. It is therefore expected that TCF will run in nearly every AIX installation, and that nearly every AIX user will be involved in some sort of cluster situation. Therefore, the examples throughout the rest of this manual are presented from the viewpoint of a user operating in a cluster environment.

It is possible, however, for a single AIX machine to operate as a single-user, standalone system. If you are operating your system this way, any text that refers to "your cluster" may be interpreted as "your computer". In this situation, "your cluster" consists of your own, single machine. Since there are no other users involved in "your cluster", you can skip those sections of the text concerned with in-cluster communications. You still need to read those sections concerned with communicating with users on remote systems.

6.0 Chapter 6. Sending and Receiving Mail

Subtopics

6.1 CONTENTS

6.2 About This Chapter

This chapter tells you how to use the electronic mail system. It provides an overview of the operation of the mail system and how to address messages for local or remote delivery. It also gives instructions to help you:

Compose and send messages to other users

Receive and read messages from other users

Organize the messages that you receive

Change the mail program to your preferences

Before you can use the mail system, it must be installed and running on your operating system. For local mail delivery only, you need only install the operating system (see *Installing and Customizing the AIX Operating System*). See *Managing the AIX Operating System* for information to configure the mail system to communicate across a network or a Basic Networking Utilities (BNU) link. Reference information about the mail system is in:

AIX Operating System Commands Reference:

- **mail** command
- **sendmail** command

AIX Operating System Technical Reference:

- **sendmail.cf** configuration file

6.3 Understanding the Mail System

The mail system is a series of programs that allows you to create, send and receive messages to and from other people on your cluster or on other computers connected to your cluster. It is similar in concept to delivery of letters through a national postal system. The following paragraphs use that similarity to help you become familiar with the parts of the mail system and how you can use them in your daily communications.

Subtopics
6.3.1 Parts of the Mail System

6.3.1 Parts of the Mail System

Note: The following paragraphs describe the operation of the mail system as it is initially installed. Both you and the person responsible for the operation of the mail system can change the operation of the mail system. See "Changing mail to Meet Your Needs" in topic 6.12 for changes that you can make to the operation of the mail system. *Managing the AIX Operating System* describes the changes that the person responsible for the mail system can make.

Figure 6-1 shows the major parts of the mail system. These parts work together to help you write, send, receive and organize your daily correspondence. Understanding the role of each of these parts will help you use the mail system to handle your correspondence most effectively.

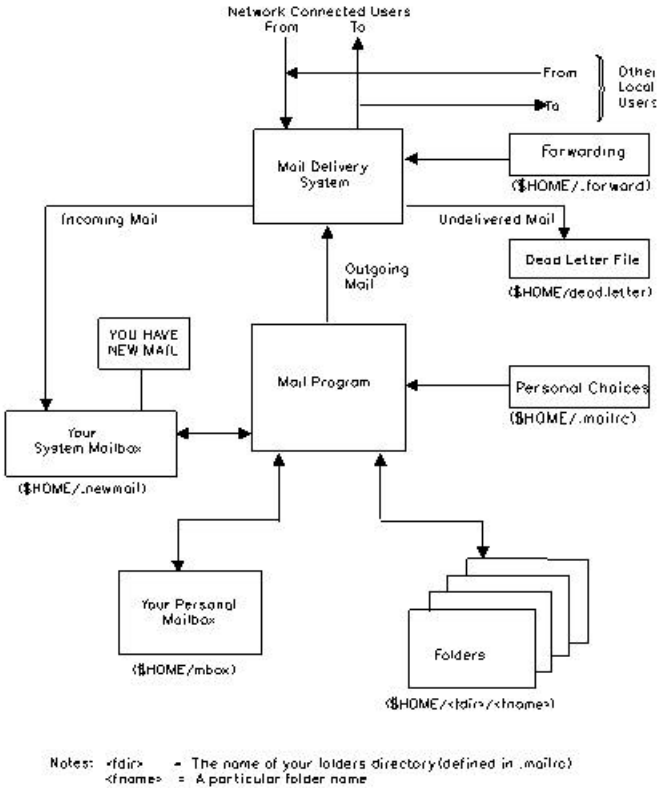


Figure 6-1. Parts of the Mail System

Subtopics
6.3.1.1 Delivery System
6.3.1.2 The dead.letter File
6.3.1.3 System Mailbox
6.3.1.4 Personal Mailbox
6.3.1.5 Folders
6.3.1.6 Personal Choices
6.3.1.7 Mail Program

6.3.1.1 Delivery System

The delivery system is a set of programs that routes mail to the correct system mailbox. You can send and receive system mail without knowing how to use the delivery system programs.

For example, imagine an ordinary letter being sent from one person to another through a post office. In this case, the sender and receiver of the letter don't worry about details like which trucks carry the mail, or which personnel sort the mail. All the sender has to do is address the letter correctly, and the letter is delivered. If the letter cannot be delivered, the sender will be notified of that fact.

Likewise, to use the AIX Operating System mail system, you don't need to know all the details about how the mail is delivered. Once the delivery system is set up (see *Managing the AIX Operating System* for information about managing the mail system), you need only be concerned about writing, sending, and receiving messages. If you provide the proper address, as described in "Addressing Mail" in topic 6.4, the delivery system either delivers your message or notifies you if the message can't be delivered.

6.3.1.2 The dead.letter File

The system uses **dead.letter** to save partially completed messages when you exit the **mail** editor with the **~q** command. In this case, the previous content of **dead.letter** is replaced with the partially completed message.

Note: Do not use **dead.letter** to store messages. The content of this file changes frequently. Use the **mail** editor **~d** command (see "Resending Undelivered Messages" in topic 6.11.13) to retrieve the contents of **dead.letter**. **~d** is a special command that the mail editor interprets as "read the contents of **~/dead.letter**". For more information on using the ~, see "Using the Escape Character" in topic 6.11.5.

6.3.1.3 System Mailbox

The ***system mailbox*** is similar in concept to the postal mailbox into which the post office delivers letters addressed to a particular person. In the mail system, the system mailbox is a file assigned to a particular user. The file is created when mail arrives for a user ID; it is deleted when all messages have been removed from the file. However, you can specify that the file not be deleted (see "Changing mail to Meet Your Needs" in topic 6.12). A separate system mailbox exists for each user ID defined in **/etc/passwd**. The mail system keeps all system mailboxes in the user's home directory. For example, if your user ID is **mark** and your home directory is **/u/mark**, then your system mailbox is **/u/mark/.newmail**.

When mail arrives for your user ID, the mail system puts the mail in your system mailbox. If you are logged on when the mail arrives, the mail system writes a message to your display station. If you are not logged on, the mail system writes the
message to your display station when you next log on. If you do not change it, the message is:

```
[YOU HAVE NEW MAIL]
```

Use the **mail** command (see "Receiving Mail" in topic 6.6) to read and remove messages from your system mailbox. Do not use the system mailbox to store messages; store messages in your personal mailbox and in folders.

6.3.1.4 Personal Mailbox

The ***personal mailbox*** is similar in concept to an in-basket in an office. You put mail in the in-basket after you have received it but before you have filed it. The personal mailbox is a working storage place for mail that still requires action.

In the mail system, the personal mailbox is a file assigned to a particular user. The mail system creates the file with the name **$HOME/mbox** (where **$HOME** is the user's login directory) when the user receives mail from his or her system mailbox. For example, if your home directory is **/u/george**, the mail system creates the file **/u/george/mbox** as your personal mailbox.

When you use the **mail** program to view mail in your system mailbox (see "Receiving Mail" in topic 6.6), **mail** automatically puts all messages that you have read but did not delete into your personal mailbox. The messages remain in your personal mailbox until you move them to a folder or delete them. See "Receiving Mail" in topic 6.6 for information about handling the contents of your personal mailbox.

6.3.1.5 Folders

Folders provide a way to save messages in an organized fashion. You can create as many folders as you need. Name each folder with a name that pertains to the subject matter of the messages that it contains, similar to file folders in an office. Using the **mail** program, you can put a message into a folder from:

Your system mailbo

Your personal mailbo

The **dead.letter** file

Another folder

Like the mailboxes, each folder is a text file. The mail system puts each folder in the directory that you specify in your **.mailrc** file (see "Creating and Using Folders" in topic 6.12.6 for information about creating and using folders). You must create this directory before using folders to store messages. Once the directory exists, **mail** creates the folders in that directory as needed.

6.3.1.6 Personal Choices

The mail system allows you to modify the way it operates to suit your needs. These choices include:

What information to include in message heading

Whether to forward incoming mail to another user I

How you want the messages handle

Other characteristics pertaining to your display station

Refer to "Changing mail to Meet Your Needs" in topic 6.12 and to "Forwarding Your Mail" in topic 6.7 for information about specifying these, and other, personal choices.

6.3.1.7 Mail Program

The **mail** program allows you to create, send, and receive messages to communicate with other users connected to your host (either directly or through a network). It includes a line-oriented editor (described in "Using the Mail Editor" in topic 6.11) for creating messages and provides a command-oriented interface for processing the contents of your system mailbox, your personal mailbox, any folders you may have, and **dead.letter**. "Sending Mail" in topic 6.5 describes how you use **mail** to create and send a message. "Receiving Mail" in topic 6.6 describes how to use **mail** to process the contents of any mailbox or folder.

6.4 Addressing Mail

Using **mail**, you can send messages and files to another user on your local cluster, on another system connected to your cluster in a network, or on another system connected to another network that has a connection to your network. The command always has the following form to start composing a message to another user:

```
$ mail address
```

However, you must supply a different form of the **address** parameter depending upon where the person receiving the message is. The concept is similar to how you might address a note to a fellow worker in an office.

For example, to send a note to someone in your department (a small department of 6 to 8 people), you might simply write his (or her) name on the envelope and put it in the mail system:

```
Hal
```

However, if Hal is in another department, you may have to provide more information on the envelope:

```
Hal
Payroll
```

If Hal is in another plant, you may need even more information to ensure that the message gets to him:

```
Hal
Payroll
Gaithersburg
```

Addressing of messages with **mail** operates in a similar fashion, as the next few sections show.

Subtopics

6.4.1 Addressing for Users on Your Local Cluster

To send a message to a user on your local cluster (that is, to someone whose login ID appears in **/etc/passwd** on your cluster), use the login ID for the address:

```
$ mail login_ID
```

For example, if user **hal** is on your cluster, use the following command to create and send a message to **hal**:

```
$ mail hal
```

This command activates **mail**, allows you to create a message to **hal**, and then tries to send the message to a local user ID of **hal**. If the message is delivered successfully, you receive no notification. If **hal** is not on your cluster, the mail system returns an error message, and puts the unsent message in your system mailbox.

6.4.2 Addressing for Users on Your Network

A cluster may be connected by network lines to other systems. These systems may be individual computers or other AIX clusters. To send a message to a user on another system connected to your cluster through a network, you must know the name of the other system in addition to the login ID (on the other system) of the person to whom you are sending the message. Refer to "Determining the Name of Another System" in topic 6.4.2.1 to find out the name of the other system and whether you can directly address the other system. If you can directly address the other system, use the login ID of the recipient, followed by @ (**at** sign), followed by the name of the remote system as the address for sending the message:

```
$ mail user_ID@system_name
```

For example, if user **hal** is on system **zeus**, use the following command to create and send a message to **hal**:

```
$ mail hal@zeus
```

This command activates **mail**, allows you to create a message to **hal**, and then tries to send the message to user ID **hal** on system **zeus**. If the message is delivered successfully, you receive no notification. If **hal** is not a user on **zeus**, you receive no error message but the mail system returns the undelivered message to your system mailbox, together with an explanation of why it could not be delivered.

Subtopics
6.4.2.1 Determining the Name of Another System

6.4.2.1 Determining the Name of Another System

The name of the system for mail routing is determined by a configuration file on that system. By convention, the name is often set to the node name of that system, but it may be defined differently in the configuration file.

If the other computer is also running AIX but it is not a part of your AIX cluster, you can find out its name directly by running the **uname -a** command from the other system. If you cannot do this for reasons such as distance or lack of a password, contact the person responsible for mail routing on the remote system and ask for the system name as it is defined in the mail configuration file. See *Managing the AIX Operating System* for more information about the configuration file. To determine other systems on the network, see your system administrator.

6.4.3 Addressing for Users on a Different Network

If the network to which your cluster is connected is also connected to other networks, you can send mail to users on those networks. If the networks use a central database of names, you do not need any additional information to send mail to users on the connected networks. Use the same addressing as for users on your local network:

```
$ mail user_ID@system_name
```

This type of addressing works well when the nature of the network allows a central database of names to be maintained. However, for networks that span large, unrelated local networks in widespread locations, a central database of names is not possible. To send mail to someone in such a network, more addressing information is needed. The address must be in the following format:

```
$ mail user_ID@system_name.domain_name
```

The additional information in this format is the ***domain name***. This information defines the remote network, relative to your local network, within the defined structure for the larger group of interconnected networks.

This information may be as simple as an added network name. For example, if your local network (named **olympus** for this example) is connected to a second network (named **valhalla**), you could use the following command to send a note to user **kelly** at system **odin** on the second network:

```
$ mail kelly@odin.valhalla
```

Similarly, user **kelly** could respond to user **hal** on **zeus**, with the following command:

```
$ mail hal@zeus.olympus
```

Frequently, however, the domain name is more than another network name. It becomes the path through the logical arrangement of domains in the network through which your message must travel. It does **not** represent the actual route that the message travels, only the position of the destination network in the interconnected network structure.

The largest and most common example of this type of interconnection is a network of business, government and educational institutions called ARPANET. At the highest structural level of this network it divides into several large domains, including:

COM for commercial entitie

EDU for educational institution

GOV for government agencie

ARPA for Advanced Research Projects Administratio

BITNET for connection to the BITNET networ

CSNET for connection to the CSNET network

Figure 6-2 shows a high-level view of some parts of the ARPANET network,

showing detail for some imaginary branches to illustrate the domain name concept.

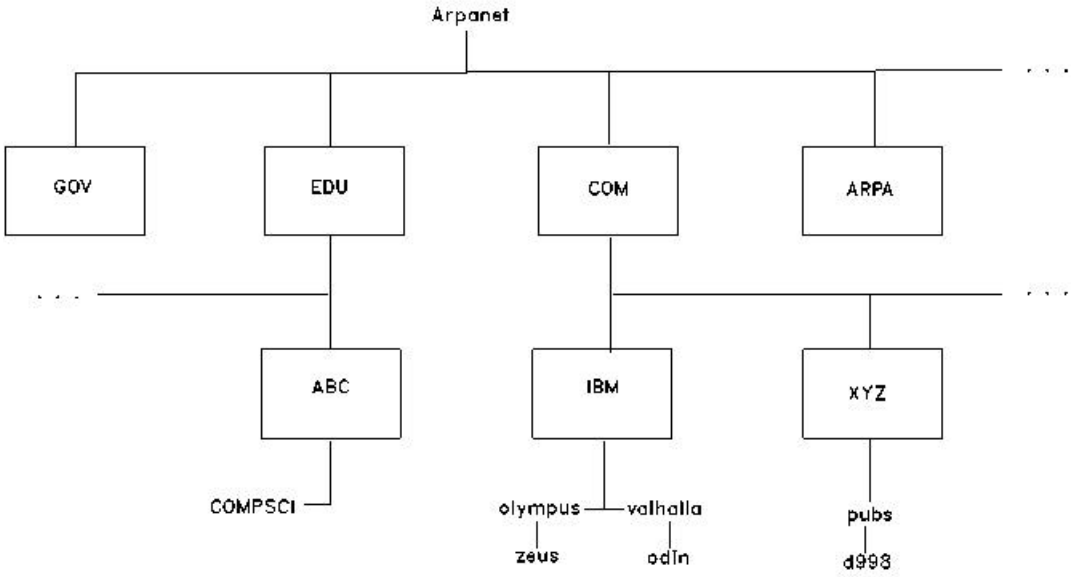


Figure 6-2. General ARPANET Structure with Example Connections

In this example, the domain **pubs** is connected to the larger domain **XYZ** and is not directly connected to **olympus** as was **valhalla** in the previous example. Therefore, use the following command to send a note to user **kelly** at system **odin** from system **d998**:

```
$ mail kelly@odin.valhalla.IBM
```

Similarly, user **kelly** responds to user **cath** on **d998**, with the following command:

```
$ mail cath@d998.pubs.XYZ
```

Each of these addresses specifies only that part of the address needed to reach the destination from the domain **COM**. The routing programs at that domain recognize the domains **IBM** and **XYZ**. However, someone at **COMPSCI** sending a message to **cath** must use the following command:

```
$ mail cath@d998.pubs.XYZ.COM
```

This example shows the complete address for user **cath** in the example network.

6.4.4 Addressing for Users Connected with a uucp Link

If your cluster is connected to some remote system via the BNU utility, this connection is referred to as a **uucp** link. Ask your system administrator if such a link has been established with any of the remote systems available to your cluster. If such links have been set up, the mail program uses that communication link automatically when you send mail to someone on that remote system. You may also send any file to this remote system by using the BNU facility directly. See Chapter 8, "Using the Basic Networking Utilities" for details.

To send a message to a user on another system connected to your cluster by the Basic Networking Utilities (BNU), referred to as **uucp** in this chapter, you must know the name of the other system and the physical route to that other system in addition to the login ID (on the other cluster) of the person to whom you are sending the message. The person responsible for connecting your system to the other system should be able to provide the proper routing information to address the other system.

Subtopics

6.4.4.1 Addressing When Your Computer Has a uucp Link

If your local computer has a **uucp** connection that can be used to reach the remote site, use the following format to address a message:

```
$ mail uucp_route!user_ID
```

The variable parameter **user_ID** is the user ID on the remote system of the person who is to receive the message. The variable parameter **uucp_route** describes the physical route that the message must follow along the **uucp** network to reach the remote system. If your system is connected to the remote system without any intermediate **uucp** systems between, then this parameter is just the name of the remote system. If your message must travel through one or more intermediate **uucp** systems before reaching the desired remote system, this parameter is a list of each of the intermediate systems, starting with the nearest system and proceeding to the farthest system, separated by **!** (exclamation marks).

For example, if your local system has a **uucp** link to a system called **merlin** and there are no other **uucp** systems between your system and **merlin**, use the following command to send a message to **ken** on that system:

```
$ mail merlin!ken
```

However, if the message must travel through systems **arthur** and **lancelot** (in that order) before reaching **merlin**, use the following command to send the message:

```
$ mail arthur!lancelot!merlin!ken
```

6.4.4.2 Addressing When the uucp Link Is on Another Computer

In a local area or wide area network environment, one of the systems on the network may have a **uucp** connection to a remote **uucp** system. You can use that **uucp** connection to send a message to a user on that remote **uucp** system. Use the following command format to send a message:

```
$ mail @systemA:@systemB:user_ID@systemC.UUCP
```

This format sends mail first to **systemA** and then to **systemB**, which routes it on a **uucp**link to **systemC**. The **.UUCP** addition to the address for **systemC** indicates that the **uucp** mailer at **systemB** handles the routing of the message to **systemC**. The system addresses in this format are in the addressing format described in "Addressing for Users on Your Network" in topic 6.4.2 and "Addressing for Users on a Different Network" in topic 6.4.3. Notice that in this format you are not sending mail to a user at any of the intermediate systems, so no user ID precedes the @ in the domain address.

Figure 6-3 shows an example network that uses domain addressing for much of the mail, but has a **uucp** link that routes mail to systems **depta** and **deptb**. The system **deptb** is connected to another system, **deptc** by a local area network. The following commands illustrate addressing using this example network.

For **kelly** at **odin** to send messages to **fred** at **depta**, **dick** at **deptb** and **bill** at **deptc**, she would use the following commands:

```
$ mail @odin.:fred@depta.UUCP
$ mail @odin.:@depta.UUCP:dick@deptb.UUCP
$ mail @odin.:@depta.UUCP:@deptb.UUCP:bill@deptc
```

These people respond with the following commands:

```
$ mail kelly@odin.UUCP
$ mail @depta.UUCP:kelly@odin.UUCP
$ mail @deptb.:@depta.UUCP:kelly@odin.UUCP
```

Similarly, **cath** at **d998** can send mail to the same people using the following commands:

```
$ mail @odin.valhalla.IBM:fred@depta.UUCP
$ mail @odin.valhalla.IBM:@depta.UUCP:dick@deptb.UUCP
$ mail @odin.valhalla.IBM:@depta.UUCP:@deptb.UUCP:bill@deptc
```

These people respond with the following commands:

```
$ mail @depta.UUCP:@odin:cath@d998.pubs.XYZ
$ mail @deptb.UUCP:@depta.UUCP:@odin:cath@d998.pubs.XYZ
$ mail @deptb.UUCP:@depta.UUCP:@odin:cath@d998.pubs.XYZ
```

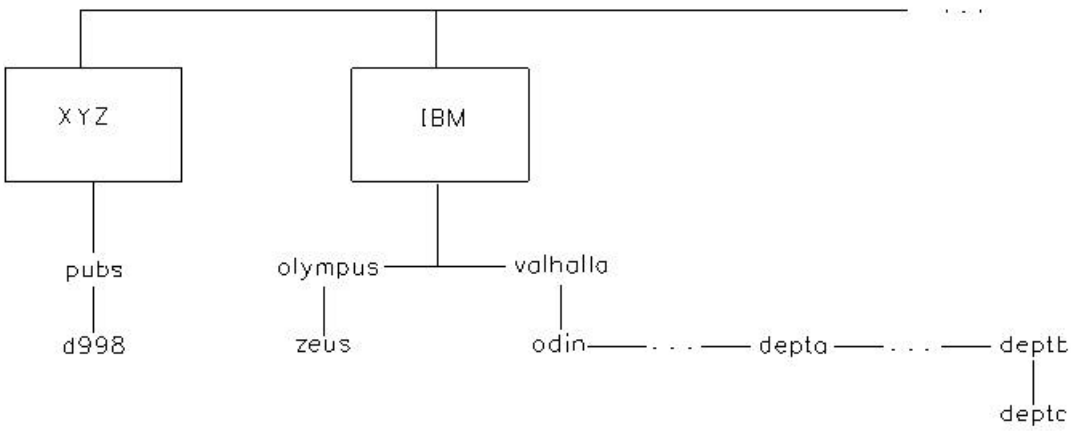


Figure 6-3. Example of uucp Connection on a Network

6.4.5 Creating Aliases and Distribution Lists

If you send mail on a large network or often send the same message to a large number of people, entering long addresses for each receiver can become tedious. To simplify this process you can create an alias or a distribution list:

alias A name that you define that can be used in place of a user address when addressing mail.

distribution list
A name that you define that can be used in place of a group of user addresses when addressing mail.

Aliases and distribution lists are used the same way and defined in similar ways; the only difference is the number of addresses defined for an alias (one address) and a distribution list (more than one address).

Subtopics
6.4.5.1 Defining an Alias or Distribution List

6.4.5.1 Defining an Alias or Distribution List

To define an alias or a distribution list that you can use when sending mail, edit the file **.mailrc** in your home (login) directory. This file contains many commands that **mail** reads when you start it from the command line. These commands are discussed in "Changing mail to Meet Your Needs" in topic 6.12. To define an alias, add a line in the following format to **.mailrc**:

```
alias name user_addr
```

To define a distribution list, add a line in the following format to **.mailrc**:

```
alias name user_addr1 user_addr2 ... user_addrn
```

In this format the variable parameter **name** can be any alphanumeric string that you choose. It should be short and easy to remember, but it cannot be the same as any of the **user_addr** parameters.

Note: If you define a **name** that is the same as a user ID on your system (as listed in **/etc/passwd**), you will not be able to send mail to that user ID. The alias name takes precedence over any defined user IDs.

The variable parameters **user_addr**x can be any address that can be used with the mail command as defined in "Addressing Mail" in topic 6.4. For example, to define an alias for user **cath** using the alias name **catherine**, you might use the following entry in **.mailrc**:

```
alias catherine @deptb.UUCP:@depta.UUCP:@odin:cath@d998.pubs.XYZ
```

With this line in **.mailrc**, you can send mail to user **cath** with the command:

```
$ mail catherine
```

Similarly, to define a distribution list that sends a common message to a group of people, you might use the following entry in **.mailrc**:

```
alias dept george anne mel@gtwn mark@mark.austin
```

With this line in **.mailrc**, you can enter the following command:

```
$ mail dept
```

To send the same message to users **george** and **anne** on the local cluster, to **mel** on system **gtwn**, and to **mark** on system **mark** in sub-domain **austin**.

In addition, you can use a previously defined alias in a distribution list. Therefore, you could add the first alias above to the distribution list to include user **cath** in the distribution list:

```
alias dept george anne mel@gtwn mark@mark.austin catherine
```

You can also define aliases that are longer than one line by adding another line that defines the same alias. The second definition is added to the first; it does not replace the first definition. For example, the following entries define the same distribution list **dept** as in the previous example:

```
alias dept george anne mel@gtwn
alias dept mark@mark.austin catherine
```

6.5 Sending Mail

Use the mail system to send information to another user. The other user need not be logged into the system when you send the information. You can use the **mail** command in one of two ways to send information. For short messages or letters that do not require a lot of formatting and editing, use the **mail** command's built-in editor to both compose and send the message. For larger letters, use your favorite editor to create the letter and then send the resulting file using the **mail** command.

Subtopics
6.5.1 Composing and Sending a Message
6.5.2 Sending a File

6.5.1 Composing and Sending a Message

The **mail** command provides a simple, line-oriented editor for entering messages. See "Using the Mail Editor" in topic 6.11 for information about using this editor. Use the following procedure to use this editor to compose and send a message.

Composing and Sending a Message

1. Enter the **mail** command on the command line followed by the address of the person, or persons, to receive the message.

   Mail **address**

   The system places the cursor on a new line and waits for input from the keyboard.

2. Type the message (see "Using the Mail Editor" in topic 6.11 for information about using the built-in editor).

3. When you are finished with the message, enter an END OF TEXT character on a line by itself. The system adds appropriate header information and sends the message. The command line prompt appears again.

For example, use the following command to compose and send a message to user **amy** on system **zeus** on a local network:

```
Mail amy@zeus
```

6.5.2 Sending a File

Use the **mail** command to send any text file to another user. The file may be a letter you have written using your favorite editor, a source file for a program you have written or any other file in text format. Use the following procedure to send a text file to another user.

Sending a File

To send a text file to another user, enter the **mail** command on the command line followed by the address of the person, or persons, to receive the message and then redirect standard input to come from the text file that you want to send.

```
Mail address < filename
```

The system reads the input file, **filename**, adds appropriate header information and sends the message. The command line prompt appears again.

For example, use the following command to send the file **letter** to user **amy** on your local cluster:

```
Mail amy < letter
```

6.6 Receiving Mail

When mail arrives for you from another user, the mail system puts the mail in your system mailbox. If you are logged on, it also sends a message to your display station periodically to tell you that new mail has arrived. If you are not logged on, a message is sent to your display station the next time that you log on. If you do not change it, the message is:

```
[YOU HAVE NEW MAIL]
```

Receiving a Message

1. Enter the **mail** command without parameters:

   ```
   $ mail
   ```

   The system displays a listing of the messages in your system mailbox.

2. Use the **t** command to display the text of a particular message.

3. Use the **q** command to exit the mailbox and return to the command line. The **mail** program saves the messages that you read in your personal mailbox if you did not delete them.

Subtopics
6.6.1 More Detailed Information

6.6.1 More Detailed Information

Use the **mail** command without parameters to view the contents of your system mailbox. If no mail is in your system mailbox, the mail system responds with the message:

```
No mail for user_ID
```

For example, if your user ID is **carol**, the following sequence occurs if no mail is in your system mailbox.

```
$ mail
No mail for carol
$
```

If there is mail in your mailbox when you enter the **mail** command, the mail system displays a listing of the messages in your system mailbox. The listing shows information about who sent the message, when it was received, how large the message is, and what the subject is (if included in the message). For example, user **george**, enters the **mail** command and receives the following display:

```
Mail   Type ? for help.
"/u/george/.newmail": 2 messages 2 new
>N  1 amy      Thu Sep 17 14:36  13/359 "Dept Meeting"
 N  2 amy      Thu Sep 17 16:28  13/416 "Meeting Delayed"
&
```

The first line is the **mail** program banner. It indicates that you can enter a **?** (question mark) to get the help screen. The second line indicates the name of the mailbox file being used (**/u/george/.newmail** is the system mailbox for user **george**), the number of messages in the mailbox, and their status. The following lines list information for each message in the mailbox. One line describes one message. The information about each message is arranged in fields, as shown in Table 6-1.

From this listing you can look at, save, reply to, or delete any of the messages. Refer to "Processing Messages in a Mailbox" in topic 6.10 for a description of what you can do while in the mailbox.

Type **q** at the **&** prompt to exit the mailbox.

Table 6-1. Mailbox Information

Field	**Content**
Pointer	The **>** in this field for a particular message indicates that the message is the ***current message*** in the mailbox. The current message is the default message for mailbox commands if no other message number is specified (see "Processing Messages in a Mailbox" in topic 6.10).
Status	A one-letter indicator of the status of the message: **M** Indicates that the message will be stored in your personal mailbox. **N** Indicates that the message is a new message. **P** Indicates that the message will be held (preserved) in your system mailbox. **U** Indicates that the message is an unread message. The message has been listed in the mailbox before,

but you have not looked at the content of the message.

R Indicates that you have read the message.

***** Indicates that you have saved or written the message to a file or folder.

No indicator indicates that the message is unresolved.

Message Number	An integer that mailbox commands use to refer to the message (see "Processing Messages in a Mailbox" in topic 6.10).
Address	The address of the person who sent the message.
Date	The date the message was received, including day of the week, month, date and time.
Size	Size of the message in number of lines and number of characters, including heading information.
Subject	The contents of the **Subject:** field of the message (if the message has one).

6.7 Forwarding Your Mail

If you are going to be away from your normal network address for an extended period of time, you may want to have your network mail sent to another network address while you are away. Sending your incoming mail to a different address (or addresses) is called ***forwarding***. The new address may be the address of a co-worker who will handle your messages while you are away, or it may be the network address where you will be working while away from your normal address. When you choose to forward your network mail, you do not receive a copy of any incoming mail in your mailbox. All mail goes directly to the address or addresses that you indicate. Use the following procedure to forward your incoming network mail to another address.

Forwarding Incoming Mail

1. Ensure that you are in your home directory:

   ```
   $ cd
   ```

2. Create a file called **.forward** that contains the network address or addresses (one address per line) to which you want to forward your incoming mail.

Subtopics
6.7.1 More Detailed Information

6.7.1 More Detailed Information

Note: The file **.forward** does not appear in a simple listing of the files in your home directory. Use the **ls -a** command to see all files that begin with a dot (.).

You tell the mail system to forward your incoming mail by creating a file in your home directory called **.forward**. This file must contain the network address or addresses to which you want to forward your incoming mail. If the file contains more than one address, each address must be on a line by itself. The following procedure explains how to create that file:

1. Use the **cd** command with no parameters to ensure that you are in your home directory. The following command sequence illustrates that action for the user ID **george**:

   ```
   $ cd
   $ pwd
   /u/george
   $
   ```

2. While in your home directory, create a file called **.forward** that contains the network address or addresses that are to receive your forwarded network mail. This file must contain valid addresses. If it is a null file (zero length), your mail is not forwarded and is stored in your mailbox. If it contains addresses that are not valid, you do not receive the mail, but the sender receives an error message and the mail is put in **dead.letter** in the sender's home directory.

   As an example of creating a **.forward** file, the following command sequence uses the **cat** command to create that file. (Note that the entry **END OF FILE** indicates the End of File character, frequently **Ctrl-D**, entered on a line by itself.) In this case, incoming mail will be forwarded to user **mark** on the local cluster and to user **amy** on system **zeus**.

   ```
   $ cat > .forward
   mark
   amy@zeus
   END OF FILE
   $
   ```

Once this file exists, you will receive no more mail. All mail is sent to the address(es) in **.forward**. When you return to your normal network address, remove this file to resume receiving mail:

```
$ rm .forward
```

6.8 Looking at Your Personal Mailbox

Messages that you read but do not delete are saved in your personal mailbox. Use the **mail -f** command to view the contents of your personal mailbox.

Looking at Your Personal Mailbox

1. Enter the **mail** command with the **-f** flag:

   ```
   $ mail -f
   ```

   The system displays a listing of the messages in your personal mailbox.

2. Use the **t** command to display the text of a particular message.

3. Use the **q** command to exit the mailbox and return to the command line.

Subtopics
6.8.1 More Detailed Information

6.8.1 More Detailed Information

Use the **mail -f** command to view the contents of your personal mailbox. If the personal mailbox does not yet exist, the system responds with an error message:

```
/u/userid/mbox: No such file or directory
```

If the personal mailbox exists but is empty, the mailbox handler becomes active and displays a mailbox header similar to:

```
Mail Version 5.2  Type ? for help.
"/u/george/mbox": 0 messages
&
```

Enter the **q** command to return to the command line.

If there is mail in your personal mailbox when you enter the **mail -f** command, the mail system displays a listing of the messages in your personal mailbox. The listing shows information similar to that shown when you look at your system mailbox (see "Receiving Mail" in topic 6.6).

From this listing you can look at, save, reply to, or delete any of the messages. Refer to "Processing Messages in a Mailbox" in topic 6.10 for a description of what you can do while in the mailbox.

Type **q** at the **&** prompt to exit the mailbox.

6.9 Looking at a Mail Folder

Use the **mail -f** command to view the contents of a defined mail folder. See "Creating and Using Folders" in topic 6.12.6 for information to create a folder.

Looking at a Mail Folder

1. Enter the **mail** command with the **-f** flag and the name of the folder using a plus sign (+) to indicate the folder name:

   ```
   $ mail -f +folder
   ```

   The system displays a listing of the messages in the indicated folder.

2. Use the **t** command to display the text of a particular message.

3. Use the **q** command to exit the folder and return to the command line.

Subtopics
6.9.1 More Detailed Information

6.9.1 More Detailed Information

Use the **mail -f** command to view the content of a mail folder. For example, to view the contents of the defined folder **status** in your folder directory (defined in **.mailrc**), use the following command:

```
$ mail -f +status
```

If the folder does not yet exist, the system responds with an error message:

```
/u/userid/folder/status: No such file or directory
```

If the folder exists but is empty or contains information that is not in the correct format, the mailbox handler becomes active, and displays a mailbox header similar to:

```
Mail Version 5.2 Type ? for help.
"/u/george/letters/reports": 0 messages
&
```

Enter the **q** command to return to the command line.

If there is mail in the folder when you enter the **mail -f** command, the mail system displays a listing of the messages in the folder. The listing shows information similar to that shown when you look at your system mailbox (see "Receiving Mail" in topic 6.6).

From this listing you can look at, save, reply to, or delete any of the messages. Refer to "Processing Messages in a Mailbox" in topic 6.10 for a description of what you can do while in the mailbox.

Type **q** at the **&** prompt to exit the mailbox.

6.10 Processing Messages in a Mailbox

You can use the **mail** command to process the contents of:

Your system mailbo

Your personal mailbo

Any mail folder that you have created

Using this program you can read, delete, store and respond to messages you receive through the mail system.

Subtopics

6.10.1 Using Mailbox Commands

When the **mail** program is processing a mailbox it displays the ***mailbox prompt*** to indicate that it is waiting for input. The mailbox prompt is the character **&** that appears at the beginning of a new line:

```
&
```

When this prompt appears, you can enter any of the mailbox commands described in this chapter or in *AIX Operating System Commands Reference*.

Subtopics
6.10.1.1 Specifying Groups of Messages
6.10.1.2 Specifying File or Folder Names

6.10.1.1 Specifying Groups of Messages

Many mailbox commands operate on a message or group of messages. You can specify the message(s) using information displayed in the listing of the contents of the mailbox, such as message number or sender. Use the **h** command ("Displaying the Contents of a Mailbox" in topic 6.10.8) to display the listing. Commands that allow groups of messages use the parameter **message_list** in the command format in this chapter. For example, the format of the **f** command (display information about messages) appears as:

```
& f message_list
```

In this format **message_list** can be one of the following:

One or more message numbers separated by space

```
& f 1 2 4 7
```

A range of message numbers indicated by the first and last numbers i the range separated by a dash

```
& f 2-5
```

is the same as

```
& f 2 3 4 5
```

One or more addresses separated by spaces to apply the command t messages received from those addresses

```
& f amy george@zeus
```

The characters entered for an address do not need to exactly match the address. They must only be contained in the address field of the messages in either upper or lower case. Therefore, the request for address **amy** matches all of the following addresses (and many others):

- **amy**
- **AmY**
- **amy@zeus**
- **hamy**

A string, preceded by a slash, to match against the **Subject:** field of the messages,

```
& f /meet
```

applies the command to all messages whose **Subject:** field contains the letters **meet** in upper or lower case. The characters entered for a match pattern do not need to exactly match the **Subject:** field. They must only be contained in the **Subject:** field of the messages in either upper or lower case. Therefore, the request for subject **meet** matches all of the following subjects (and many others):

- **Meeting on Thursday**
- **Come to meeting tomorrow**
- **MEET ME IN ST.LOUIS**

6.10.1.2 Specifying File or Folder Names

Many mailbox commands allow you to specify a file or folder name to be used with the command. Commands that allow a file or folder name use the parameter **fname** in the command format in this chapter. For example, the format of the **file** command (change mailbox files) appears as:

```
& file fname
```

In this format, **fname** is one of the following:

The pathname of the new mailbox relative to the current directory For example, if the current directory is your home directory, use the following command to change to your personal mailbox:

```
file mbox
```

The program changes to that mailbox and displays a list of the contents of that mailbox.

The absolute pathname of the new mailbox. For example, if your use ID is **george** use the following command to change to your system mailbox:

```
file /u/george/mbox
```

The shorthand form of a folder name in your directory defined fo folders (see "Creating and Using Folders" in topic 6.12.6 for information about using folders). For example, if you define your folder directory as **letters**, use the following command to change to the folder **reports**:

```
file +reports
```

The **+** is a shorthand form for the full pathname of the folder directory. Therefore, this command performs the same function as if it had been entered:

```
file /u/george/letters/reports
```

6.10.2 Looking at a Mailbox

To start the **mail** program with one of the main types of mailboxes, see the following procedures:

Mailbox	**See Description**
System Mailbox	"Receiving Mail" in topic 6.6
Personal Mailbox	"Looking at Your Personal Mailbox" in topic 6.8
Folders	"Looking at a Mail Folder" in topic 6.9

6.10.3 Leaving the Mailbox

You can leave the mailbox and return to the operating system using one of two commands:

Use the **q** command to leave the mailbox and return to the operating system. When you leave the mailbox, all messages that you marked to be deleted are removed from the mailbox and cannot be recovered. For example, the following command processes the mailbox commands and returns you to the operating system:

```
& q
```

Use the **x** command to leave the mailbox and return to the operating system **without changing the original contents of the mailbox**. The program ignores any requests to delete messages. For example, the following command returns you to the operating system without changing the content of the mailbox:

```
& x
```

6.10.4 Getting Help

While using **mail** to look at a mailbox, display a summary of many mailbox commands by entering the **?** command:

```
& ?
```

You can also display a list of all mailbox commands (with no explanation of what they do) by entering the **l** command:

```
& l
```

6.10.5 Finding the Name of the Current Mailbox

Although the **mail** command displays the name of the current mailbox when it is started, you may lose track of what the current mailbox is. Use the **file** command without parameters to find out the name of the current mailbox. When you enter this command, it responds with the name of the current mailbox, the number of messages and whether any messages have been marked to be deleted. For example, if the current mailbox is **/u/george/mbox**, the following sequence occurs when you enter the **file** command:

```
& file
/u/george/mbox: 2 messages 1 deleted
&
```

This message indicates that **/u/george/mbox** contains two messages and that one of those messages will be deleted when you finish with this mailbox.

6.10.6 Changing Mailboxes

Note: When you change mailboxes, any messages that you marked to be deleted are deleted when you leave the first mailbox. If you return to that mailbox, the deleted messages cannot be recovered.

Once the program is started with one mailbox, use the **file** command to change to another mailbox. The format of this command is:

file **fname**

Refer to "Specifying File or Folder Names" in topic 6.10.1.2 for an explanation of the **fname** parameter.

6.10.7 Reading a Message from a Mailbox

To look at a message, enter the number of that message at the mailbox prompt (**&**). Pressing **Enter** only at the mailbox prompt displays the current message. If the mailbox listing is:

```
Mail  Type ? for help.
"/u/george/.newmail": 2 messages 2 new
>N  1 amy      Thu Sep 17 14:36  13/359 "Dept Meeting"
 N  2 amy      Thu Sep 17 16:28  13/416 "Meeting Delayed"
&
```

pressing **Enter** displays the message **Dept Meeting**, because message number 1 is the current message (indicated by the > in the first column). Entering the number **2** displays the message **Meeting Delayed**:

```
CHAPTER
Message  2:
From george Thu Sep 17 14:38 CDT 1987
Received: by zeus
        id AA00716; Thu, 17 Sep 87 14:38:53 CDT
Date: Thu, 17 Sep 87 14:38:53 CDT
From: amy
Message-Id: <8709171938.AA00716@zeus>
To: george
Subject: Meeting Delayed
Status: R


The department meeting scheduled for 1:30 PM tomorrow
has been postponed to 3:30 PM.  It will still be held
in the planning conference room.

EOF:_
```

The **EOF:** prompt indicates that **pg** is being used to display the message. See "Controlling the Display Scroll" in topic 6.12.5.1 to change this option. Press **Enter** to return to the mailbox prompt.

Subtopics
6.10.7.1 Looking at the Next Message
6.10.7.2 Looking at More Than One Message

6.10.7.1 Looking at the Next Message

Use the **n** command to look at the next message in the mailbox. The next message then becomes the current message. For example, if the current message is message number 6, then the following command displays message number 7 and makes message number 7 the current message:

```
& n
```

6.10.7.2 Looking at More Than One Message

Note: When displaying more than one message at a time, be sure to include the **set crt** command in your **.mailrc** file (see "Changing mail to Meet Your Needs" in topic 6.12). You can also enter this command at the mailbox prompt. If you do not use this command, the displayed messages scroll up and off the screen without pausing for you to read them.

To display more than one message in succession, use the **t** command with a list or range of message numbers. The format for this command is:

& t **message_list**

Refer to "Specifying Groups of Messages" in topic 6.10.1.1 for an explanation of the **message_list** parameter.

6.10.8 Displaying the Contents of a Mailbox

When the **mail** program starts, it lists what is currently in the mailbox that it is using (as described in "Receiving Mail" in topic 6.6). You can see this list again by using the **h** command. This command is useful to help you keep track of messages in the mailbox as you perform actions on them. Messages that you delete are not shown in the listing.

Only a certain number (about 20) of messages can be listed at a time. The actual number is determined by the display station type being used and by the **set screen** command (see "Controlling the Display Scroll" in topic 6.12.5.1). If the mailbox contains more than that number of messages, information about only the first group of messages will be displayed. To see information about the rest of the messages, use the **h** command with a number that is in the next range of message numbers (21 to 40 in this case). You can also use the **z** command to display the next screen of messages.

For example, suppose the mailbox contains 25 messages and the current list shows messages numbered 1 through 20. The following command displays information about messages numbered 21 through 25:

```
& h 21
```

To return to the first group of messages, enter the following command:

```
& h 1
```

You can also change the group of messages by displaying any of the messages in the desired group. For example, if you display message number 5, then the first group of messages becomes the current group of messages. Using the **h** command shows information about messages 1 through 20.

Subtopics
6.10.8.1 Displaying Information about Selected Messages

6.10.8.1 Displaying Information about Selected Messages

If you have a large number of messages in your mailbox, you may want to display the heading information only about groups of messages. Use the **f** command with a list or range of message numbers. The format for this command is:

```
& f message_list
```

Refer to "Specifying Groups of Messages" in topic 6.10.1.1 for an explanation of the **message_list** parameter.

6.10.9 Deleting and Recalling Messages

Use the **d** command to delete messages from a mailbox. The format of this command is:

```
& d message_list
```

Note: If you delete a message and either change to another mailbox or quit the mailbox (with the **q** command), the deleted message cannot be recalled.

Once a message is deleted, but **before leaving** the current mailbox, you can recall that message ("undelete" it) with the **u** command. The format of this command is:

```
& u message_list
```

Refer to "Specifying Groups of Messages" in topic 6.10.1.1 for an explanation of the **message_list** parameter.

Entering **d** without a message list deletes the current message. Entering **u** without a message list recalls the last deleted message. You can also use the **dt** command to delete the current message and automatically display the next message. For example, if the current message is message number 4, then the following command deletes message 4 and displays message 5:

```
& dt
```

6.10.10 Saving Messages in a File or Folder

You can add the contents of a message to a file or folder using one of two commands. One command includes the message headings in the file or folder; the other command adds only the text of the message to the file or folder. Both of these commands add information to the end of an existing file, or create a new file. They do not destroy information currently in the file.

Use the **s** command to save a message including heading information to a file or folder. The format of this command is:

```
& s message_list fname
```

Use the **w** command to save a message without heading information (text of the message only) to a file or folder. The format of this command is:

```
& w message_list fname
```

Refer to "Specifying Groups of Messages" in topic 6.10.1.1 for an explanation of the **message_list** parameter. Refer to "Specifying File or Folder Names" in topic 6.10.1.2 for an explanation of the **fname** parameter.

For example, the following command saves messages 1, 2, 3 and 4 with their heading information to a file called **notes** in the current directory:

```
& s 1-4 +notes
"notes" [Appended] 62/1610
```

As an additional example, if message number 6 contains the following information:

```
From root Fri Sep 11 12:55 CDT 1987
Received: by zeus
        id AA00549; Fri, 11 Sep 87 12:55:25 CDT
Received: by thor
        id AA00178; Fri, 11 Sep 87 12:57:15 CDT
Date: Fri, 11 Sep 87 12:57:15 CDT
From: su@thor.8d33
Message-Id: <8709111757.AA00178@thor>
To: george@zeus
Status: RO

Please change your password.
```

Use the following command to save the entire message to a folder called **admin** in your folder directory (defined as **/u/george/letters** in your **.mailrc** file):

```
& s 6 +admin
"/u/george/letters/admin" [New file] 14/321
```

Use the following command to save the text only to a file called **text** in the current directory:

```
& w 6 text
"text" [New file] 12/30
```

The file **text** contains the following:

```
Please change your password.
```

6.10.11 Editing a Message

You can use your favorite editor to add information to a note in your mailbox. When you leave the editor, you return to the mailbox prompt to continue processing the messages in the mailbox.

Two mailbox commands allow you to activate one of two editors to edit the text of a message:

Command	**Function**
e [**message_number**]	This command activates the **ed** editor, or other editor that you define (see "Changing mail to Meet Your Needs" in topic 6.12).
v [**message_number**]	This command activates the **vi** editor, or other editor that you define (see "Changing mail to Meet Your Needs" in topic 6.12).

For each of these commands, **message_number** is the number of the message that you want to edit. If you do not specify a message number, **mail** activates the editor using the current message.

6.10.12 Creating a Message

While using the **mail** command to process a mailbox, you can create a new message by activating the **mail** editor (see "Using the Mail Editor" in topic 6.11 for information about using the editor). Use one of three commands to activate the editor from the mailbox prompt depending upon the purpose of the message:

Command	**Function**
R	Respond to the sender of a message.
r	Respond to the sender and all others who received copies of a message.
m	Create a new message independent from any received messages.

Subtopics
6.10.12.1 Responding to the Sender Only
6.10.12.2 Responding to the Sender and Recipients
6.10.12.3 Creating a New Message

6.10.12.1 Responding to the Sender Only

Use the **R** command to send a response message to the originator of a message. This command creates a new message addressed to the sender of the selected message and with a **Subject:** field that refers to the selected message. Then it activates the **mail** editor to allow you to enter text into the new message. The format of this command is:

```
& R [message_number]
```

The parameter **message_number** is the message number of the message to which you want to reply. If you do not specify a message number, **mail** creates a reply to the current message.

For example, suppose message number 4 is as follows:

```
From root Thu Sep 17 14:45 CDT 1987
Received: by zeus
        id AA00731; Thu, 17 Sep 87 14:44:59 CDT
Received: by thor
        id AA00614; Thu, 17 Sep 87 14:47:53 CDT
Date: Thu, 17 Sep 87 14:47:53 CDT
From: amy@thor
Message-Id: <8709171947.AA00614@thor>
To: george@zeus
Subject: Department Meeting
Cc: mark@zeus, mel@gtwn
Status: RO

Please plan to attend a department meeting tomorrow
at 1:30 PM in the planning conference room.
```

In this case, you would type the following things to send a reply message to **amy@thor** (**bold** indicates text that you enter; other text is echoed by the program):

```
& R 4
To:  amy@thor
Subject: Re:  Department Meeting

I'll be there.
[EOT]
```

When you enter the **EOT** (**Ctrl-D** on many display stations), the program sends the message to **amy@thor** and returns you to the mailbox prompt.

6.10.12.2 Responding to the Sender and Recipients

Use the **r** command to respond to the originator of a message and send a copy of your response to everyone on the **Cc:** list. The **r** command creates a new message that is addressed to the sender of the selected message and copied to the people on the **Cc:** list. The **Subject:** field of the new message refers to the selected message. The **r** command also activates the **mail** editor so you can enter text into the new message. The format of this command is:

```
& r [message_number]
```

The parameter **message_number** is the number of the message to which you want to reply. If you do not specify a message number, **mail** creates a reply to the current message.

For example, using message number 4 in the previous example, the following sequence occurs to generate a reply message to **amy@thor** as well as **mark@zeus** and **mel@gtwn** (**bold** indicates entered text; other text is echoed by the program):

```
& r 4
To:  amy@thor
Cc:  mark@zeus mel@gtwn
Subject: Re:  Department Meeting

I'll be there.
[EOT]
```

When you enter the **EOT** (**Ctrl-D** on many terminals), the program sends a copy of the message to all addressees and returns you to the mailbox prompt.

6.10.12.3 Creating a New Message

Use the **m** command to create a new message while processing a mailbox. The format for this command is:

& m **address**

The **address** parameter is any proper user address as described in "Addressing Mail" in topic 6.4. This command starts the **mail** editor to create a new message as described in "Sending Mail" in topic 6.5.

6.10.13 Listing Defined Aliases

While processing a mailbox you can get a listing of the aliases that are defined for this **mail** session by entering the **a** command. The format for this command is:

```
& a
```

This command displays all aliases and their corresponding addresses, one alias per line. Refer to "Addressing Mail" in topic 6.4 for information about defining an alias to be used as an address.

6.11 Using the Mail Editor

The **mail** command provides a line-oriented editor for composing messages. This editor allows you to enter each line of the message and then press **Enter** to get a new line to enter more text. You cannot change a line once you have entered it. However, before pressing **Enter**, you can change information on that one line by using the **Backspace** and **Delete** keys to erase the information, and then type in the correct information.

Subtopics

6.11.1 Starting the Mail Editor

You can start the **mail** editor in one of two ways. From the command line you can start the editor to compose and send a message to another user as described in "Composing and Sending a Message" in topic 6.5.1:

```
mail address
```

You can also use the **mail** editor to compose a reply to mail that you receive using the **R**, **r** or **m** commands, as described in "Receiving Mail" in topic 6.6.

6.11.2 Sending the Message

When the editor is active and it contains some message text that you have entered, you can send that message and end the editor with the following procedure.

Sending the Message

1. Press **Enter** to position the cursor at the beginning of a new line.

2. Enter an END OF TEXT character (**Ctrl-D** on many terminals). The system sends the message and returns you either to the mailbox handling program or to the operating system command line, depending upon where you were when you started the editor.

You can change the editor (as described in "Changing mail to Meet Your Needs" in topic 6.12) to allow a . (period) to be used as an additional END OF TEXT character in the above procedure.

6.11.3 Quitting without Sending a Message

When the editor is active, you can use the following procedure to quit the editor without sending a message.

Quitting the Editor

1. Press **Enter** to position the cursor at the beginning of a new line.
2. Enter **~q**. The system saves the message in **dead.letter** and returns you to the operating system command line.

You can also quit the editor using the **INTERRUPT** key sequence. This method also saves the message in **dead.letter** unless you press **INTERRUPT** before pressing **Enter** on the first line of text.

Quitting the Editor without Saving

1. Press **INTERRUPT**. The system responds with the prompt:

   (Interrupt -- one more to kill letter)

2. Press **INTERRUPT** again. The system ends the **mail** program. If you have entered text, the system displays the message:

   (Last Interrupt -- letter saved in dead.letter)

   It then returns you to the command line.

6.11.4 Getting Help

While using the **mail** editor to create a message, you can display a summary of the editor commands by entering the key sequence

```
~?
```

on a line by itself.

6.11.5 Using the Escape Character

The editor includes many control commands that allow you to perform other operations on a message. Each of these commands must begin with a special escape character. This is usually the tilde (~), but with older versions of AIX, it is often the Control (**Ctrl**) key.

Note: You can change the escape character to any other character by including the **set escape** command in your **.mailrc** file. (See "Changing mail to Meet Your Needs" in topic 6.12).

These two escape keys are used differently. When the **Ctrl** key is used, you must first press the **Ctrl** key and then, while holding the **Ctrl** key down press a second key. **Ctrl** key escape sequences can be entered anywhere on a line. When the tilde is used, you must enter each of the escape characters on a new line. You move to a new line, press the tilde key once and release it. Then press a second key.

If you should ever need to start a new line in your message with the tilde character, a special technique is necessary. You must use two tildes together. For example, the following message entered as:

```
This is a tilde (~) and this is two tildes (~~).  However,
~~ results in sending only one tilde.
```

is sent as the following message:

```
This is a tilde (~) and this is two tildes (~~).  However,
~ results in sending only one tilde.
```

In this book all examples use the tilde escape character. Your system may require that you use the **Ctrl** key or some other character instead. If you cannot obtain the expected results by using the tilde escape sequences recommended here, ask your system administrator for help.

6.11.6 Displaying a Message

To look at lines of the message that you have entered (or that have been read from another file), use the **~p** command. When you enter this command on a line by itself in the **mail** editor, the editor displays the contents of the message including the header information for the message. The text scrolls up from the bottom of the display. If the message is larger than one screen and you have not set the page size for your terminal (see **stty** in *AIX Operating System Commands Reference*), the text scrolls off the top of the screen until it displays the last screen of the message followed by the **mail** editor's **(Continue)** prompt. To look at the content of large messages, use the **mail** editor commands to view the message with your favorite editor as described in "Changing a Message" in topic 6.11.7.

6.11.7 Changing a Message

You cannot change information on a line once you have pressed the **Enter** key and gone on to the next line. You can, however, change the content of your message before sending it by editing the message with another editor. The following paragraphs describe how to activate different editors from the **mail** editor.

Subtopics
6.11.7.1 Using a Different Editor
6.11.7.2 Defining a Different Editor

6.11.7.1 Using a Different Editor

To change information that you have already entered, you can activate a different editor without leaving the **mail** editor. Once you have activated a different editor, you can use it to change the message, or add new information to the message. When you leave the different editor, you return to the **mail** editor to continue composing, or to send, your message.

Enter the following key sequences on a new line in the **mail** editor to activate one of the different editors to edit the text of the current message:

~e This sequence activates the **ed** editor, or other editor that you define.

~v This sequence activates the **vi** editor, or other editor that you define.

When you save the message and quit the different editor, you return to the **mail** editor. You can then continue to compose the message, or use one of the other **mail** editor commands to process the message.

6.11.7.2 Defining a Different Editor

The **mail** editor allows you to define two different editors to use when changing a message from within the **mail** program. You define either or both editors with the **set** command in your **.mailrc** file as shown in the following table. If you do not define these editors in **.mailrc**, **mail** tries to use the editors shown as the default in the table.

set Command	**Usage**
set EDITOR=pathname	This command in your **.mailrc** file defines the editor that you activate with the **~e** key sequence. The value of **pathname** must be the full path name to the editor program that you want to use. Default: **/usr/bin/ed**
set VISUAL=pathname	This command in your **.mailrc** file defines the editor that you activate with the **~v** key sequence. The value of **pathname** must be the full path name to the editor program that you want to use. Default: **/usr/bin/vi**

For example, the following entry in your **.mailrc** file defines the **ed** editor for use with the **~e** key sequence:

```
set EDITOR=/bin/ed
```

6.11.8 Reformatting the Message

Note: Do not use the **fmt** command if the message contains imbedded messages or preformatted information from external files (see "Including Another Message" in topic 6.11.12). This command reformats the heading information in imbedded messages and may change the format of preformatted information.

After you have entered the message and before sending it, you may want to reformat the message to improve its appearance. Use the **~|** key sequence along with the **fmt** shell program to reformat the message. Enter the following command on a new line to reformat the message:

```
~| fmt
```

This command uses the **fmt** command to change the appearance of the message by reflowing the information for each paragraph within defined margins (a blank line must separate each paragraph).

6.11.9 Checking for Misspelling

If you have the text formatting set of programs installed on your system, you can use the **spell** program to check your message for misspelled words with the following procedure:

Checking for Misspelling

1. Write the message to a temporary file. For example, to write the message to the file **spellchk**, use the following command:

   ```
   ~w spellchk
   ```

2. Run the **spell** program using the temporary file as input. For example, the following command uses the temporary file **spellchk** as input to the **spell** program:

   ```
   ~! spell spellchk
   ```

   The **spell** program responds with a list of words that are not in its list of known words, followed by an **!** (exclamation point) to indicate that you have returned to the **mail** program.

3. Examine the list of words to determine if you need to use an editor to correct any of them (see "Changing a Message" in topic 6.11.7).

4. Erase the temporary file. The following command erases the temporary file in the preceding example:

   ```
   ~! rm spellchk
   ```

6.11.10 Changing the Heading

The heading of the message contains routing information and a short statement of the subject. You must specify at least one recipient of the message, but the other information is not required. The information in the heading may include the following:

To: This field contains the address or addresses to which the message is to be sent.

Subject: This field contains a short summary of the topic of the message.

Cc: This field contains the address or addresses of persons that are to receive copies of the message. This field is included as part of the message sent to all who receive the message.

Bcc: This field contains the address or addresses of persons that are to receive blind copies of the message. This field is **not** included as part of the message sent to all who receive the message.

You can set up **mail** to automatically ask you for the information for these fields by putting commands in your **.mailrc** file (see "Changing mail to Meet Your Needs" in topic 6.12). If you have not changed **.mailrc** or if you need to change the information that you entered in these fields, use the commands described in the following paragraphs to change the information in these fields.

Subtopics
6.11.10.1 Editing the Heading Information
6.11.10.2 Adding to the To: List
6.11.10.3 Setting the Subject: Field
6.11.10.4 Adding to the Cc: List
6.11.10.5 Adding to the Bcc: List

6.11.10.1 Editing the Heading Information

To add to or change information in more than one of the heading fields, use the **~h** command. When you enter this command on a new line, the system displays each of the four heading fields, one at a time. You can view the content of each field, delete information from that field (using the **Backspace** key) or add information to that field when the field and its contents are displayed. Pressing **Enter** saves any changes to that field and displays the next field and its contents. When you press **Enter** for the last field (**Bcc:**), you return to the editor.

For example, when composing a message, enter the **~h** command to change the **Subject:** and **Cc:** fields:

```
~h                  (Enter)
To: mark@austin_
```

The system responds with the contents of the **To:** field (**mark@austin**) and places the cursor at the end of that field. You could edit or add to this field at this time, but this information is correct. Press **Enter**. The system responds with the contents of the **Subject:** field:

```
Subject: Fishng Trip_
```

In this case, we want to correct the misspelling in the indicated subject. The cursor is at the end of the subject field. Press the **BACKSPACE** key seven times to position the cursor under the **n** in **Fishng**. Retype the rest of the subject to correct it to **Fishing Trip**. Press **Enter**. The system responds with the contents of the **Cc:** field:

```
Cc: mel@gtwn_
```

To add another person to the copy list, ensure that the cursor is at the end of the list, type a space, and then type the address of the new person. For example:

```
Cc: mel@gtwn george@austin
```

This entry expands the copy list to two persons. When you have completed the copy list, press **Enter**. The system responds with the contents of the **Bcc:** field. Press **Enter**. The system responds with the **(Continue)** prompt and returns you to the **mail** editor at the current end of the message.

6.11.10.2 Adding to the To: List

Use the **~t** command to add one or more addresses to the **To:** list. For example, the **To:** list for a message may contain the following address:

```
To: mark@austin
```

To add to this list, use the following command:

```
~t george@austin mel@gtwn
```

This command changes the **To:** list to:

```
To: mark@austin george@austin mel@gtwn
```

When you enter this command, the system does not show you the contents of the **To:** field (as it did with the **~h** command). It accepts your input as an addition to whatever is already in the field. You cannot use the **~t** command to change or delete the contents of the **To:** list. If you have forgotten what is in the field or you wish to change it, you must use the **~h** command.

6.11.10.3 Setting the Subject: Field

Use the **~s** command to set the **Subject:** field to a particular phrase or sentence. Using this command replaces the previous contents (if any) of the **Subject:** field. For example, the **Subject** field for a message may contain the following phrase:

```
Subject: Vacation
```

To change the **Subject:** field, use the following command:

```
~s Fishing Trip
```

This command changes the **Subject:** field to:

```
Subject: Fishing Trip
```

The system does not show you the previous contents of the **Subject:** field. It overwrites the contents with your new entry. You cannot append to the **Subject:** field with this command. To edit this field or append to it, use the **~h** command.

6.11.10.4 Adding to the Cc: List

Use the **~c** command to add one or more addresses to the **Cc:** list. For example, the **Cc:** list for a message may contain the following addresses:

```
Cc: mark@austin amy
```

To add to this list, use the following command:

```
~c george@austin mel@gtwn
```

This command changes the **Cc:** list to:

```
Cc: mark@austin amy george@austin mel@gtwn
```

This command does not show the contents of the **Cc:** field; it allows you to append new entries to the contents.

You cannot use the **~c** command to change or delete the contents of the **Cc:** list.

6.11.10.5 Adding to the Bcc: List

Use the **~b** command to add one or more addresses to the **Bcc:** list. For example, the **Bcc:** list for a message may contain the following address:

```
Bcc: mark@austin
```

To add to this list, use the following command:

```
~b george@austin mel@gtwn
```

This command changes the **Bcc:** list to:

```
Bcc: mark@austin george@austin mel@gtwn
```

This command does not show the contents of the **Bcc:** field; it allows you to append new entries to the contents.

You cannot use the **~b** command to change or delete the contents of the **Bcc:** list.

6.11.11 Including Information from Another File

You can include information from other files in the message you are currently writing. This allows you to include data, such as a schedule, from another file. Use the **~r** command to read the contents of a file into the current message. The format of this command is:

```
~r filename
```

For example, to read the contents of file **schedule** and append that information to the current end of the message, use the following command:

```
~r schedule
```

6.11.12 Including Another Message

Note: To use the commands **~m** and **~f** that include other messages within your message, you must enter the editor from the mailbox (using either the **r**, **R**, or **m** mailbox commands). See "Receiving Mail" in topic 6.6 for information about using the mailbox commands.

You can include another message within the current message for reference purposes or to forward the other message to another user.

Use the **~m** command to include another message in the current message for reference purposes. This command reads the indicated message and appends it to the current end of the message. The included message is indented one tab character from the normal left margin of the message. You can change the indent character by editing **~/.mailrc**. The format of this command is:

~m **numlist**

Use the **~f** command to include another message in the current message to forward the message to another user. This command reads the indicated message and appends it to the current end of the message, but does not indent the appended message. Also use this command to append messages for reference whose margins are too wide to imbed with the **~m** command. The format of this command is:

~f **numlist**

In the preceding formats, the parameter **numlist** is a list of integers that refer to valid message numbers in the mailbox or folder currently being handled by **mail**. You can enter simple ranges of numbers also. For example, the following commands imbed the indicated messages if those message numbers exist in the current mailbox or folder:

Command	**Result**
~m 1	Appends message number 1 to the current end of the message being written. Message number 1 is indented one tab/current indent character from the left margin.
~m 1 3	Appends message number 1 and then message number 3 to the current end of the message being written. Both messages are indented one tab/current indent character from the left margin.
~f 1-4	Appends message numbers 1, 2, 3 and 4 to the current end of the message being written. These messages are aligned with the left margin (not indented).

6.11.13 Resending Undelivered Messages

When **mail** cannot deliver a message that you send, it places that message in a file named **dead.letter** in your home (login) directory. This file can also contain a partial letter that was saved when you quit using the **~q** command from the **mail** editor. To read the contents of **dead.letter** into the current message, use the **~d** command.

```
~d
```

This command appends the contents of **dead.letter** to the current end of the message and responds with the **(Continue)** prompt. You can then continue to add to, or send, the message.

6.12 Changing mail to Meet Your Needs

The person responsible for managing your system defines the initial configuration of the **mail** program, You may alter the way the **mail** program operates to meet your personal requirements. The characteristics of a **mail** session that you can change include:

Whether **mail** prompts for the subject of a message

Whether **mail** prompts for users to get a copy of a message

If any aliases or distribution lists are define

How many lines are displayed when reading message

What information is listed in message

Whether a folder directory is selected for storing messages i

Whether you wish to be notified automatically when new mail arrives (See the **biff** command in *AIX Operating System Commands Reference*.)

Whether a log file is set up to record outgoing message

Whether different editors can be used for entering message

How to exit the **mail** editor

How **mail** stores messages.

The system administrator uses the file **/usr/lib/Mail.rc** to define the initial configuration. This file contains **mail** commands that perform the tasks mentioned above. Although the initial configuration can meet the needs of most users, you can easily
alter it by creating a file in your home (login) directory with the name **.mailrc**. Commands in this file override similar commands in **/usr/lib/Mail.rc** when you run **mail**.

Another way of executing **mail** commands that are stored in a file is by using the **source** command. When reading mail, you can issue this command from the **mail** command line as follows:

```
& source pathname
```

where **pathname** is the path and file containing the **mail** commands. Commands in this file override the previous settings of any similar commands for the duration of the current session. You may also alter the characteristics of the current **mail** session by entering commands at the mailbox prompt (**&**).

Subtopics

6.12.1 Commands for Customizing Mail

There are four **mail** commands that alter the characteristics of the **mail** session. These are **set**, **unset**, **alias**, and **ignore**.

Subtopics
6.12.1.1 The set and unset Commands
6.12.1.2 The alias Command
6.12.1.3 The ignore Command

6.12.1.1 The set and unset Commands

The **set** command and its inverse, the **unset** command, are used in conjunction with ***options***. Use the **set** command to enable options and the **unset** command to disable options. You can also use the **set** command to assign a value to an option.

The format for using the **set** command to enable options is:

```
set [option_list]
```

The **option_list** may be one or more options that you want to enable. Entering **set** without the **option_list** shows what options are already enabled. Refer to the section "Checking Mail Characteristics" in topic 6.12.2 to see when to use **set** with no **option_list**.

The format for the **unset** command is:

```
unset option_list
```

You must include the **option_list** with the **unset** command.

For example, to cause **mail** to prompt for a subject field, use the following entry:

```
set ask
```

To also cause **mail** to prompt for a copy-to-field, use the following entry:

```
set ask askcc
```

To suppress both of these prompts, use this entry:

```
unset ask askcc
```

The format for using the **set** command to assign a value to an option is:

```
set option=value
```

An example of a ***valued option*** is shown in the following entry:

```
set toplines=10
```

With this entry in your **.mailrc** file, the **top** command displays only the first ten lines of a message. (See "Controlling the Display Scroll" in topic 6.12.5.1.)

6.12.1.2 The alias Command

Use the **alias** command in your **.mailrc** file to define alias names and distribution lists. The **alias** command allows you to send messages without entering long addresses or long lists of addresses. The section "Creating Aliases and Distribution Lists" in topic 6.4.5 describes how to use the **alias** command.

You can define a distribution list with the **alias** command that includes your own address. If you send a message using the distribution list, however, the mail system does not normally send a copy to your mailbox. Use the following **set** command to enable sending a copy to yourself also:

```
set metoo
```

With this entry in **.mailrc**, anytime you send a message using an alias name that includes you, a copy of the message will be put in your mailbox.

6.12.1.3 The ignore Command

Use the **ignore** command to define what information is listed in message headers. The message headers are the fields like **To:** and **From:** at the tops of messages. Refer to "Controlling What Information Is Displayed" in topic 6.12.5.2 to see how to use the **ignore** command.

6.12.2 Checking Mail Characteristics

The characteristics of a **mail** session are determined by many commands and options. Commands in **.mailrc** and **/usr/lib/Mail.rc** affect each **mail** session. So do any commands you entered during the current session. You can avoid confusion by reviewing the characteristics of a mail session as described in this section.

Before running **mail**, you can use the **pg** command to view **/usr/lib/Mail.rc** and see what **mail** commands are in it. You can also look at your **.mailrc** file.

When reading your mail, use the **set** command without any arguments to list all of the options that are currently enabled. From this list you can also see if a folder directory is selected and if a log file is set up to record outgoing messages. For example, using the **set** command from the mailbox prompt (**&**) could produce a display as follows:

```
& set
ask
metoo
toplines        10
&
```

You can see from this list that two options are enabled: **ask** and **metoo**. Notice that there is no **askcc** entry in the list. This indicates that the **askcc** option is not enabled. You can also see that the **toplines** option has been assigned the value 10.

Two other commands from the **mail** command line provide current command settings. The **ignore** command with no arguments lists all header fields that are not included when you use a **t** or **p** command to display a message. The **alias** command without any arguments lists all alias names that are currently defined.

The information listed by the **set**, **ignore**, and **alias** commands includes system default settings, settings from the file **/usr/lib/Mail.rc**, settings from your **.mailrc** file, and any settings you made during the current **mail** session.

6.12.3 Prompting for a Subject Field

When you start **mail** to begin writing a message to another user, the program may or may not ask you for a **Subject:** field with a prompt similar to:

```
Subject:
```

If this prompt appears you can fill in a summary of the subject matter of the message, and that summary is included at the start of the message that you send. Whether this prompt appears or not is determined by the presence of the **ask** option. To enable the subject prompt, enter the following line in your **.mailrc** file:

```
set ask
```

To prevent **mail** from displaying this prompt, either delete the **set ask** statement from **.mailrc**, or enter the following line in **.mailrc**:

```
unset ask
```

6.12.4 Prompting for a Copy-To Field

You can set up **mail** so that when you send a message, **mail** prompts you for the names of other users whom you want to receive copies of the message. This prompt is similar to the following:

```
Cc:
```

This prompt appears if the **askcc** option is set in the system file **/usr/lib/Mail.rc** or in your **.mailrc** file as shown below:

```
set askcc
```

To suppress this prompt, either delete the **set askcc** entry from your **.mailrc** file, or include the following entry in **.mailrc**:

```
unset askcc
```

6.12.5 Changing How Mail Displays a Message

You can set several options from your **.mailrc** file to control how much information **mail** displays at different times. You can also put the **ignore** command in your **.mailrc** file to keep header fields from being displayed.

The following sections show you how to use **mail** commands to control display functions.

Subtopics
6.12.5.1 Controlling the Display Scroll
6.12.5.2 Controlling What Information Is Displayed
6.12.5.3 Combining the delete and print Commands

6.12.5.1 Controlling the Display Scroll

Each message in your mailbox has a one-line heading in the message list. If you have more than 24 messages, the first headings from the message list scroll past the top of your screen whenever you display the message list. The **set screen** command controls how many lines of the list are displayed at a time. For example, with the following entry in **.mailrc**:

```
set screen=20
```

the **h** command displays 20 message headers, then waits for you to press the **Enter** key before displaying the next 20 headers.

A similar situation occurs when you display a long message. If you display a message with more than 24 lines, then the first lines of the message scroll past the top of the screen. You can use the **pg** program from within **mail** to browse through long messages if you include the **set crt** command in **.mailrc**.

The **set crt** command has the following form:

```
set crt=n
```

The value for **n** determines how many lines a message must be before the **pg** program is started. The **pg** program is invoked whenever you read messages with more than this many lines.

For example, if you use the **t** command to read a long message, only one screen (or ***page***) is displayed. The page is followed by a colon prompt to let you know that there are more pages. Press the **Enter** key to display the next page of the message. After the last page of the message is displayed, there is a prompt similar to the following:

```
EOF:
```

At this prompt, or the colon prompt, you can enter any valid **pg** command. You can display previous pages, search the message for character strings, or quit reading the message and return to the mailbox prompt (**&**). Refer to *AIX Operating System Commands Reference* for more information about the **pg** program.

The **top** command lets you scan through messages to get more information without reading entire messages. You control how many lines of a message are displayed with the **top** command by setting the **toplines** option as follows:

```
set toplines=n
```

In this command, **n** is number of lines, starting from the top and including all header fields, that are displayed with the **top** command.

For example, if user **amy** has the following line in her **.mailrc** file:

```
set toplines=10
```

When Amy runs **mail** to read her new messages, she receives the following display:

```
Mail  Type ? for help.
"/u/amy/.newmail": 2 messages 2 new
```

```
>N  1 george   Wed Jan  6  9:47  11/257 "Dept Meeting"
 N  2 mark     Wed Jan  6 12:59  17/445 "Project Planner"
& _
```

Now Amy uses the **top** command to browse through her messages as shown in the following dialogue (what Amy enters is in bold type):

```
& top 1
Message 1:
From george Wed Jan 6 9:47 CST 1988
Received: by zeus
        id AA00549; Wed, 6 Jan 88 9:47:46 CST
Date: Wed, 6 Jan 88 9:47:46 CST
From: george@zeus
Message-Id: <8709111757.AA00178>
To: amy@zeus
Subject: Dept Meeting

Please plan to attend the department meeting on Friday
at 1:30 in the planning conference room.  We will be

& _
```

The message was not displayed completely because **toplines** was set to ten, so only lines 1 (the **Received:** field) through 10 (the second line of the message body) were displayed. The first line, **From george Wed Jan 6 9:47 CST 1988**, is always present and does not count in the **toplines** option.

6.12.5.2 Controlling What Information Is Displayed

Every message has several header fields at the top. These header fields are normally displayed when you read a message. However, you can use the **ignore** command to suppress the display of header fields when a message is read with a **t** or **p** (lowercase) command. The format for the **ignore** command is:

```
ignore [field_list]
```

The optional **field_list** can consist of one or more field names that you want to **ignore** when you display a message. For example, if Amy includes the following line in her **.mailrc** file:

```
ignore date from to
```

and the file **/usr/lib/Mail.rc** has the line:

```
ignore received message-id
```

the result of using the **t** command is shown below:

```
& t 1
Message 1:
From george Wed Jan 6 9:47 CST 1988
Subject: Dept Meeting

Please plan to attend the department meeting on Friday
at 1:30 in the planning conference room.  We will be
discussing the new procedures for using the project
planning program developed by our department.

& _
```

Many fields do not appear in the display. To display these fields, use a **T** or **P** (uppercase) command or the **top** command. You can enter the **ignore** command without any arguments at the **mail** command line to get a list of the currently ignored header fields.

You can set the **quiet** option so that when you display a message, the message number is not displayed first. To do this, put the following entry in **.mailrc**:

```
set quiet
```

With the **quiet** option in **.mailrc**, the **mail** banner is not displayed when you start **mail**. The banner is the line that shows the name of the **mail** program.

Another option that suppresses the **mail** banner is shown below:

```
set noheader
```

If you put this command in **.mailrc**, the list of messages in your mailbox is not displayed when you start **mail**. The only response you will get is the mailbox prompt (**&**). You can get a list of messages by using the **h** command.

6.12.5.3 Combining the delete and print Commands

After you read a message you can delete it with the **d** command and display the next message with the **p** command. You can also combine these commands by entering the following line in your **.mailrc** file:

```
set autoprint
```

This command causes the **d** command to delete the current message and display the next one.

6.12.6 Creating and Using Folders

As you read mail messages pertaining to different subjects, you can store them in appropriate folders (mail system files) and read them again during later **mail** sessions. You can create new folders during a **mail** session as necessary, but a directory for storing them must exist before defining any new folders. Since a folder directory is just a normal directory used for storing folders, you can create a new folder directory from within **mail** by using the **mkdir** shell command as follows:

```
& !mkdir pathname
```

You must select a folder directory before storing any messages in it. To select a folder directory, set the **folder** option from the **mail** command line as follows:

```
& set folder=pathname
```

You can also include the **set folder** command in **.mailrc** so that when you invoke **mail**, the folder directory is already selected.

As you read messages, you can append them to any folder or place them into new folders within the selected folder directory. In this manner, you can sort your new messages into folders like in a file cabinet. For example, upon logging in, user **george** sees that he has new mail. He enters the **mail** command and receives the following display:

```
Mail   Type ? for help.
"/u/george/.newmail": 2 messages 2 new
>N  1 amy      Tue Dec 14 13:24  32/947 "New Utilities"
 N  2 mark     Wed Dec 15 15:47  16/417 "Project Schedule"
& _
```

After reading the first message, George sees that it documents some fancy new shell procedures that Amy has written. George decides that it should go into a special folder he uses to collect such things. George has the following **set folder** command in his **.mailrc** file so that the folder directory where that folder is kept is already selected:

```
set folder=/u/george/doc
```

George uses the **save** command to append the new message to the special folder **procedures** by using a **+** symbol to indicate the folder name as follows:

```
& save 1 +procedures
```

He receives the message:

```
"/u/george/doc/procedures" [Appended] 32/947
```

George can access all messages saved in the **procedures** folder as described in "Looking at a Mail Folder" in topic 6.9.

The second message is a project schedule. There is no folder yet for keeping project schedules, so George decides to create one. He also wants to put the folder into a directory where he has other files concerning the project. George selects this directory by entering the following command:

```
& set folder=/u/george/projectX
```

and the new folder can be created with the **save** command as follows:

```
  & save 2 +schedules
```

The message:

```
  "/u/george/projectX/schedules" [New File] 16/417
```

indicates that a new folder has been created.

6.12.7 Keeping a Record of Messages Sent

The **mail** command can automatically make a copy of any messages you send and store them in a specified file that can be read later. Since the header information is also stored, recording outgoing messages is a useful way of logging when important information was sent to others. Normally **mail** does not keep any record of messages sent.

To enable this option, place a **set record** command in **.mailrc** as shown below:

```
set record=pathname
```

Here the **pathname** indicates the file relative to your home (login) directory. The **mail** commands do not create directories, so any directories included in the **pathname** must already exist before using this command. Putting this command in your **.mailrc** file guarantees that copies of all of your messages will go to the same place.

If Amy has the following lines in her **.mailrc** file:

```
set record=letters/mailout
set folder=letters
```

a copy of the messages she sends out is put into the file **/u/amy/letters/mailout**.

Amy can read the recorded messages with **mail** as follows:

```
$ mail -f +mailout
```

because the folder **mailout** is in the folder directory selected by the **set folder=letters** command in her **.mailrc** file.

If you set up a file to record outgoing messages, you should read the file periodically with **mail** and delete all of the unnecessary messages. Otherwise, the file will grow and eventually use up all of your storage space.

6.12.8 Selecting a Different Editor

The standard **mail** editor is good for entering short messages, but it does not allow you to alter text after you press the **Enter** key. An alternative is to use another editor to create a file and use **mail** to send the file. However, the file will still exist after it has been sent. You can set up **mail** so that you can use any editor on your system to enter a message from within **mail**, and the message will not be left in a file when you exit **mail**.

Use the **set** command with the following valued options to define two different editors:

```
set editor=e_pathname
set visual=v_pathname
```

In the first entry, **e_pathname** is the full pathname of the editor you want to activate with the **~e** escape sequence or the **e** command. In the second entry, **v_pathname** is the full pathname of the editor you want to activate with the **~v** escape sequence or the **v** command.

If user Amy includes the following line in her **.mailrc** file:

```
set editor=/bin/ed
```

she can call up her favorite editor (from **/bin/ed**) by using the **~e** escape sequence from within the standard **mail** editor.

When Amy is finished entering her message, she exits from her favorite editor and returns to the standard **mail** editor. She can then type the EOF character to exit **mail** and send the message.

As Amy reads her mail she can edit messages to add information to them. Using the **e** command from the mailbox prompt (**&**) also invokes the editor specified in the **set editor** command. After she exits the editor, she returns to the **mail** command line where she can save the message to a folder.

6.12.9 Defining How to Exit the Mail Editor

When you enter the **mail** command to send a message, you invoke the **mail** editor. From the **mail** editor you can compose your message. You can exit from the **mail** editor in one of two ways. One method is to type the **EOF** character.

Another method is to enter a period on a line by itself. This period does not appear in the message. To enable this method, place the following line in **.mailrc**:

```
set dot
```

After you exit the **mail** editor, the message is sent, and you return to the system prompt.

If you reply to a message when reading your mail, you also invoke the **mail** editor. The **set dot** command allows you to exit from the **mail** editor, but you return to the mailbox prompt (**&**). From there you can exit **mail** with a **quit** command, an **exit** command, or with the **EOF** character.

6.12.10 Defining How Mail Stores Messages

The **mail** program has several defaults for how messages are stored when you exit. You can set three options in **.mailrc** to change how **mail** stores messages. These options are the **append** option, the **hold** option, and the **keepsave** option. This section describes how these options change the way **mail** stores messages.

Normally, **mail** adds messages to a mailbox at the top of the mailbox. You can cause **mail** to append messages to the end of the mailbox by putting the following entry in **.mailrc**:

```
set append
```

Messages are stored in different places when you exit **mail**, depending on how you exit. You can exit **mail** in three ways. One way to exit **mail** is to enter the **exit** command. Use the **exit** command to return all messages to the mailbox you are reading. The mailbox will have the same messages the next time you read it. Another way to exit is to enter the **quit** command. Use the **quit** command to dispose of files as described in the following paragraphs. The third way to exit **mail** is to enter the **EOF** character. Using the **EOF** character is the same as using the **quit** command.

As you read messages from a mailbox, you can delete them, save them, or leave them unresolved. If you do not read a message, it remains in the mailbox, no matter how you exit **mail**.

Messages that are deleted are not saved anywhere if you exit **mail** with a **quit** command. However, if you exit with an **exit** command, deleted messages remain in the mailbox.

An ***unresolved message*** is one that you read, but did not delete or save. If you exit **mail** with a **quit** command, any unresolved messages are stored in a file called **mbox** in your home (login) directory. You can cause **mail** to leave unresolved messages in the mailbox you are reading, instead of storing them in **mbox**, by putting the following entry in **.mailrc**:

```
set hold
```

The **hold** option has no effect on deleted messages.

Instead of using the **set hold** option, you can use the **hold** command to specify that a message remain in the mailbox when you exit with the **quit** command. For example, if you read message 3, but you are not sure if you want to delete it, mark it with the **hold** command:

```
& hold 3
```

The message remains in the mailbox instead of going to **mbox**. You can wait until the next time you read the mailbox to decide how to dispose of it.

If you set the **hold** option in **.mailrc**, you can use the **mbox** command to mark messages so that they are stored in **mbox** when you exit with the **quit** command. For example, if you are reading new mail, you can mark messages that you read, but did not dispose of, by using the **mbox** command:

```
& mbox 1 3 5-7
```

This example marks messages 1, 3, 5, 6, and 7 so that they are stored in

mbox instead of remaining in the system mailbox with any unread messages or other unresolved messages.

If you save a message with a **save** or **write** command, **mail** deletes the message from the mailbox when you exit with the **quit** command. To keep the saved message in the mailbox, use the **set keepsave** command in **.mailrc**:

```
set keepsave
```

Now **mail** handles messages that you save just like unresolved messages. The **set hold** option causes them to be held in the mailbox. Without the **set hold** option, they are stored in **mbox**.

If you exit **mail** with the **exit** command, all messages remain in the mailbox, no matter what options are set. Also, if you save any messages, the messages remain in the files where you saved them, even if you use the **exit** command.

6.12.11 Viewing New Message Before Exiting Mail

If new mail messages arrive for you while you are viewing your mail, they are not automatically displayed. To view new messages automatically, set an option in your **.mailrc** file as follows:

```
set seenew
```

This option causes **mail** to re-read your system mailbox file before you exit. If new messages have arrived, **mail** displays the following message:

```
New Mail has arrived
```

followed by the new mail messages.

6.13 Communicating with CMS Users

AIX/370 is only one of the operating systems that will run on a System/370 mainframe. There are a number of others, and in many large organizations, AIX/370 will share the central processor(s) with one or more of these operating systems. Two of the most popular are called CMS and TSO. There may be CMS or TSO users in your own organization with whom you need to communicate, and one or more of these other operating systems will almost certainly be in use on remote systems available to your cluster. The AIX Operating System allows you to exchange mail and other files with these users just as you can with users on other AIX systems.

There are two AIX commands which send and receive files to users of CMS or MVS/TSO:

uvcp Used to send messages or other files to users of CMS or MVS/TSO.

vucp Used by AIX to receive incoming files from CMS or MVS/TSO.

Both of these utilities handle binary data, so they can be used to send and receive program files and condensed data as well as text files. They both take optional parameters that allow you to specify that the file being handled is a CMS **Note**.

The **uvcp** command has the following format:

uvcp -**options filename** -d **username** at **node**

By using the appropriate **options** with the **uvcp** command, the user may do the following:

Send a binary file

Request an acknowledgement to your message

Specify that the message is to be received as a CMS **Note**.

Specify that your message is to be received by CMS as either a fixe or variable format file.

Other variables to the **uvcp** command include:

filename The name of the file you wish to send.

-d username Specifies the user to wish to receive your file.

at node Specifies the name of the computer in the network where your intended receive is logged in.

The **vucp** command has the following format:

vucp -**options destination option filename**

By using the appropriate **options** with the **vucp** command, the user may do the following:

Receive a binary file

Have the received file overwrite existing files of the same name. I

you do not specify this option, the software searches the destination directory for a file that has the same name as the one being received. If it finds one, it cancels the operation. This prevents you from mistakenly overwriting valuable information that exists in a file of the same name.

Send data to your screen that informs you of the sender, the machin from which the file is being sent, and the time and date the file was sent.

Other variables to the **vucp** command include:

destination Where the file is to be stored. This can be a directory name or a full pathname, which may include a directory and filename.

filename The name of the file being received.

Your system administrator can give you addresses for the remote users you may need to contact. For further information on the **uvcp** and **vucp** commands, see *AIX Operating System Commands Reference*.

Subtopics
6.13.1 Addressing for Users Connected by an RSCS Link
6.13.2 Using Japanese Characters in Mail

6.13.1 Addressing for Users Connected by an RSCS Link

The AIX Operating System **mail** command can be used to exchange messages with CMS users. The **mail** command has been designed to use the **rscsmail** and **rscssrvr** commands automatically. All that is necessary is to address outgoing mail as follows:

```
mail user@node.RSCS
```

This command sends your file to the named **user** on the named **node** (machine) using the RSCS program to communicate with the system.

Incoming mail appears in your mailbox and you may read it like any other message.

6.13.2 Using Japanese Characters in Mail

Like any other standard AIX program, the **mail** command is invoked by typing its name in ASCII characters. After mail is running, however, the user is free to enter the body of the message text in Japanese characters. This means that a Japanese language user will need to switch back and forth between ASCII and Japanese text entry modes.

User names are always in ASCII. This means that you address your mail to a Japanese user by typing in his user name in ASCII characters. (The program always prompts you with the ASCII To: string.)

If your recipient is on a remote system, you may need a whole string of ASCII characters. This may include strings of the following type:

```
$ mail user_ID
$ mail user_ID@system_name
$ mail user_ID@system_name.domain_name
$ mail uucp_route!user_ID
```

Note: All of these entries must be composed entirely of ASCII characters.

The program also asks you for the names other users to whom copies of the message are to be sent. (The program prompts you with the ASCII CC: string.) Those user names also require ASCII characters.

Then **mail** prompts you for the subject. This entry may be in Japanese characters.

The body of the message may be Japanese text. The standard **mail** editor allows you to enter Japanese characters. If you call up another system editor (such as **vi**) to edit your message, that second editor will also allow you to edit Japanese characters. The **~v** sequence used to call this editor, however, must be ASCII characters, and all the editor commands you give within the editor program must also be ASCII.

It is important to consider, however, that a message in Japanese text may not be readable unless the receiver is also operating within a Japanese locale. The receiver may be a Japanese speaking person, but if that user is operating with a non-Japanese locale, the **mail** program displays a Japanese text message as nonsense on the screen.

7.0 Chapter 7. Using Cluster Communications Facilities

Subtopics

7.1 CONTENTS

7.2 About This Chapter

This chapter shows you how to use the communication facility for your cluster. Using this simple cluster facility, you can communicate with other users in your cluster using the **talk**, **write**, **who**, and **mesg** commands. Communications are in the form of **messages**; that is, the sender and the receiver are both logged in on the cluster.

The communications described in this chapter are of two types:

messages The sender and the receiver are both logged in on the cluster.

conferences Messages are sent back and forth by a pre-arranged protocol.

The rest of this chapter discusses the specific tasks you can perform with a cluster communication facility. You can:

Determine who can currently receive messages

Send messages to a user logged in on the cluster

Receive or reject messages when you are logged in on the cluster

Participate in an on-line conference

The following chapters contain information about cluster communication facilities:

To send mail to a user on a cluster system, refer t Chapter 6, "Sending and Receiving Mail."

To use the Basic Networking Utilities commands to communicate with remote system, refer to Chapter 8, "Using the Basic Networking Utilities."

To communicate with a remote system using TCP/IP, refer t Chapter 9, "Using TCP/IP."

To communicate with a remote system using ATE, refer to Chapter 10, "Using Asynchronous Terminal Emulation (ATE)."

Ask your system administrator which of these facilities is available for your use. For more detail about the commands mentioned here, see *AIX Operating System Commands Reference*.

7.3 Determining Who Can Receive Messages (who)

To receive messages, a recipient must be logged in on the cluster when the message is sent. You can determine who is logged in by using the **who** command.

who Displays the user name and display station of all users who are currently logged in on the cluster and shows the date and hour each current session began.

who -u Additionally, displays the number of hours and minutes since there was activity at a display station and provides the process ID number of each user. Activity for less than one minute is indicated by a blank space.

Determining Who Is Logged In

1. Type **who** following the prompt on the command line.

```
who
```

   If cluster **bjh** logged in at **7:43** from display station **tty0** and **jmc** logged in from display station **tty1** at **9:27**, the system displays the following:

```
bjh  tty0  Feb 8  07:43
jmc  tty1  Feb 8  09:27
```

2. Type **who -u** for more extensive information. If **bjh** has been inactive for an hour and two minutes and **jmc** has been inactive for less than a minute, the system displays the following:

```
bjh  tty00  Feb 8  07:43  01:02  17
jmc  tty01  Feb 8  09:27         28
```

   The numbers l7 and 28 are process IDs.

Note: If you are logged on to a PS/2 running the standalone version of the AIX PS/2 Operating System, your displays for the **who** command will look like those in the example. If you are operating in a cluster environment where more than one host is available, the **machinename** of the host you are logged on to also appears.

7.4 Sending Messages (write)

Note: This message facility allows text to be entered in Japanese characters. As with other system commands, you start the program with its ASCII name. Then, you can switch to any Japanese entry mode. You end the program with an ASCII **Ctrl-D** or whatever other ASCII key sequence your terminal uses for the **END-OF-FILE** symbol.

You can send a message to anyone currently logged on the cluster issuing the **write** command. If the person is not currently logged on, the system displays this message:

```
user is not logged on
```

If you receive this message, you can still communicate with the individual by sending a note to the recipient's mail box. The recipient will see the note at the next login. See "Sending Mail" in topic 6.5 for information on sending mail messages.

The box that follows shows you how to send a message, using **jmc** as the sender and **bjh** as the recipient.

Sending a Message

1. Type **write username** on the command line following the prompt.

   ```
   write bjh
   ```

   This sends an alerting sound (the ASCII BEL character) and the following notice to **bjh's** display:

   ```
   Message via 'write' from jmc at tty1
   Mon Feb 8 10:32:45
   ```

   When the connection is ready, you receive two alerting sounds (ASCII BEL characters).

2. Type your message and press the **Enter** key.

3. Now send an **END OF FILE** symbol on a separate line.

   This tells **bjh** that you have finished, and signals the system to terminate the connection.

If **bjh** answers your message, the system displays the following:

```
Message via 'write' from bjh at tt0
Mon Feb 8 10:35:00
```

The configuration of your keyboard determines whether **Ctrl-D** will produce

the **END OF FILE** symbol. Refer to the technical information for your specific display station if **Ctrl-D** does not produce an **END OF FILE** symbol.

Note: If you **write** to a user who is logged on to two different machines, your message goes to the first one the system finds, which may or may not be the one where he is currently working.

Subtopics
7.4.1 Having a Conversation
7.4.2 Sending a Long Message

7.4.1 Having a Conversation

If you expect to have a conversation (sending messages back and forth), select a symbol, such as the letter **o** for "over", to use to indicate that a message is complete. Select a different symbol to identify the end of the conversation (such as **oo** for "over and out.").

If you use an **END OF FILE** symbol before the conversation ends, the system terminates the connection. The conversation can be continued by one of the participants by issuing the **write** command.

7.4.2 Sending a Long Message

If you want to send a long message, you can create a file to contain the text and then send that file as the message.

Sending a Long Message

1. Create a file, using the text editor of your choice.

2. Write the text. Make the last entry in the file an **END OF FILE** symbol.

3. Use the **write** command to send the message, and enter the user name of the recipient, a < (less than symbol), and the name of the file.

   To send a message file named **letter.jmc** to **bjh**, enter the following:

```
write bjh < letter.jmc
```

When the message is sent, the connection between the display stations ends. You can, however, signal the system that you want to retain the connection.

7.5 Receiving Messages (mesg)

Note: This message facility allows text to be entered in Japanese characters. As with other system commands, you start the program with its ASCII name. Then, you can switch to any Japanese entry mode. You end the program with an ASCII **Ctrl-D** or whatever other ASCII key sequence your terminal uses for the **END-OF-FILE** symbol.

Unless you specify otherwise, the system automatically sends you all the messages directed to you by other users on the cluster. Sometimes, however, receiving a message on your display can interrupt your current work, so you may occasionally prefer not to receive messages. You can use the **mesg** command to turn off the message facility, to turn it back on again, and to check the current message status of your display station.

The procedure that starts up the message facility is included as part of the AIX Operating System; this ***start-up procedure*** contains a default value that lets you receive messages. If you prefer not to receive messages at any time, you can change this default so that you do not have to enter the **mesg** command each time you want to turn the message facility off or on.

Subtopics

7.5.1 Using the mesg Command

You use the **mesg** command in the following ways:

To check the current status of the message facility on your displa station

To turn off the message facility, thereby rejecting message

To turn the message facility back on, thereby again receivin messages.

To check the current status of the message facility, you enter the **mesg** command by itself, without any options. To turn off the message facility and reject incoming messages, you enter the command with the **n** option, called a flag in the command line. To turn the message facility back on so you can receive messages, enter the command with the **y** flag.

mesg Indicates the current message status of your display station. The system displays either **is y**, which means that you can receive messages, or **is n**, which means that you cannot receive a message.

mesg n Prevents incoming messages from appearing on your display, except under the following conditions:

A person with superuser authority sends the message.

The sender uses the **mail** command and sends the message to your mail box. See "Receiving Mail" in topic 6.6.

mesg y Permits incoming messages to appear on your display.

Checking and Changing Your Message Status

1. Check your message status by entering **mesg** following the prompt on the command line:

   ```
   mesg
   ```

   If you can receive messages, the system displays the following:

   ```
   is y
   ```

2. To change the setting, enter **mesg y** or **mesg n**. For example, to reject messages, enter the following:

   ```
   mesg n
   ```

You can type **mesg y** or **mesg n** without first checking your message status.

When you change the setting to **msg n**, someone attempting to contact you through this facility receives the message **Permission denied**.

7.5.2 Changing the mesg Start-Up Procedure

The shell start-up procedure included with the AIX Operating System contains a default value that lets you receive messages. If you prefer not to receive messages, you can change this default.

To change the default value so that you do not receive any messages, add a **mesg n** notation to the file named **.profile** in your **$HOME** or login directory.

Note: When you log in to the operating system, it automatically places you in a personalized directory called your **$HOME** or login directory. This directory was created for you when your account was set up. For detailed information about directories, refer to Chapter 3, "Using the File System."

Modifying Your Message Start-up Procedure

1. Enter the following command to move to your **$HOME** or login directory:

   ```
   cd
   ```

2. To edit the file named **.profile**, enter the following:

   ```
   ed .profile
   ```

   The system displays the contents of your **.profile** file on the screen.

3. Move to the end of the file and enter the following as the last line in the file:

   ```
   mesg n
   ```

4. When you are ready to exit from the file, press **END OF FILE**.

Adding **mesg n** to your personal **.profile** overrides the default value in the shell start-up procedure so that you no longer automatically receive messages.

You can turn the message facility on during a particular session b entering the **mesg y** command. When you log out, however, the message facility is turned off, and it remains off until you enter **mesg y** again.

7.6 Holding a Conversation (talk)

The AIX Operating System offers an alternative to the **write** command. The **talk** command allows you to have a split-screen display. Your messages appear in one half of the screen and the other user's messages appear in the other half. It also allows you to interrupt the other person.

The **talk** command provides a full report of the other user's whereabouts. If the user you want to **talk** to is logged on to more than one display station, **talk** gives you a listing of these display stations and the idle time for each. You may then choose the one that has the least idle time and be reasonably sure you have selected the display station where your party is currently active.

Conducting a talk Exchange

1. Request a **talk** exchange with another user by typing the command name followed by the user-id of the person you want to **talk** to.

   ```
   talk john
   ```

   Your screen clears and divides into two portions, an upper section and a lower section, divided by a horizontal line in the center. The upper section is for your messages. The other user's responses will appear in the lower section. You receive the following message

   ```
   No connection yet.
   ```

   while the system is trying to establish your connection.

2. The other user gets a beep from his terminal, accompanied with a message that tells him which user on which system is requesting the exchange.

   ```
   Message from Talk_Daemon@poker at 13:05 ...
   talk:  connection requested by steve@poker.
   talk:  respond with:  talk steve@poker
   ```

3. The other user responds by issuing the **talk** command with your user-id.

   ```
   talk steve@poker
   ```

4. Your terminal displays the following message:

   ```
   Connection established.
   ```

5. Now each user can type whatever they wish. You will see the other user's message as he types it. He will see yours the same way.

6. When both of you have finished, one user sends a message to the other indicating that all has been said. Any words will do.

   Bye.

7. Each user terminates the connection with **Ctrl-C**. The system responds with the message:

   Communication closed. Exiting.

8.0 Chapter 8. Using the Basic Networking Utilities

Subtopics

8.1 CONTENTS

8.2 About This Chapter

The BNU program consists of commands, processes, and a supporting database composed of files that contain information necessary to establish connections to remote computer systems.

The BNU commands enable you to perform the following types of operations:

Connect to remote systems over a hardwired line or over a telephon network, using a modem

Run commands on a remote syste

Send and receive files locally or transmit files between the loca cluster and a remote system

Get status information about the BNU's remote connection or the fil transfer.

BNU commands are also used to manage network-related tasks such as installing and maintaining the BNU software. For information about these operations, see *Managing the AIX Operating System*.

8.3 Introduction to the Basic Networking Utilities Program

The BNU program, which is actually a group of programs and files, functions as a ***background process***. This means that once a BNU task is running, you can use your display station for other jobs.

In addition to your ability to communicate with other users in your TCF cluster, you can also communicate with remote systems. BNU enables two UNIX-based computer systems to communicate in three different ways:

over a hardwired lin

over a telephone lin

over an IBM Token Ring Network, or an Ethernet network, runnin TCP/IP.

BNU has two communication features:

Call-out Sends files and commands to another system.

Call-in Receives files and commands sent from another system.

You can have both types of communication features on your cluster if you have a line dedicated to BNU.

The following figure illustrates, in a simplified way, the setup of a BNU network.

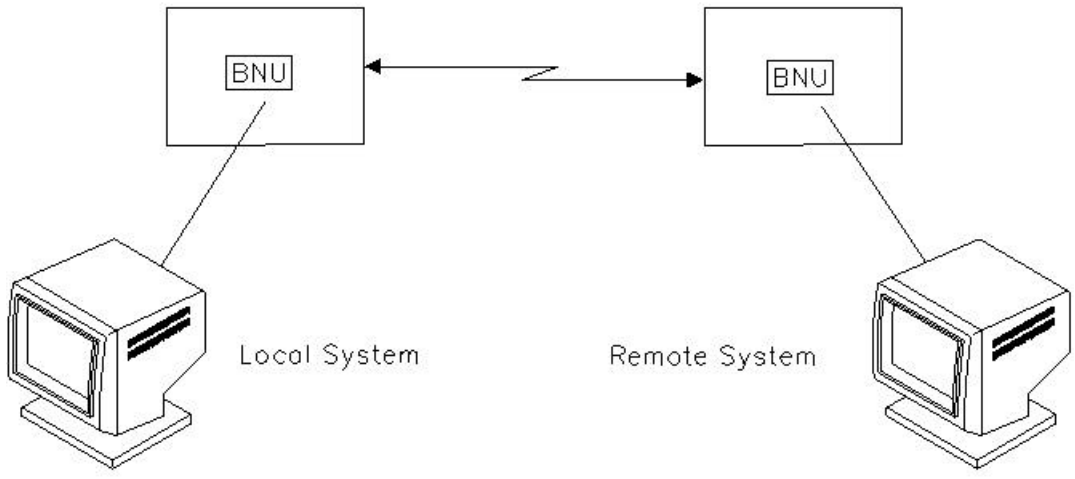


Figure 8-1. BNU Communications

8.4 Identifying Compatible Systems (uuname)

In order for a local cluster to communicate with a remote system using BNU, the remote system must meet the following criteria:

It must have a UNIX-based operating system, like AIX

It must be connected to your local cluster

Note: You can use BNU to communicate between an AIX system and a non-UNIX-based operating system, but such communications may require additional hardware or software. The remote systems that you can access with the BNU commands are identified when BNU is installed; these systems are listed in a file called **Systems**, which is stored in the directory **/usr/adm/uucp** which is maintained by the system administrator. For information about this file, see *Managing the AIX Operating System*.

Use the BNU command **uuname** to identify compatible remote systems with which your AIX system is set up to communicate using the BNU program.

Subtopics

8.4.1 Form of the uuname Command

To identify the systems with which you can communicate using the BNU program, issue the **uuname** command in the following form:

uuname [**option**]

When you invoke **uuname**, your local cluster (the host you are currently logged on to) displays the names of the remote systems that you can access with the BNU commands. These names (which were assigned to the computers when they were installed as BNU recipients) appear in a list like the following:

```
venus
merlin
hera
zeus
research
cad
archives
```

The entries your system displays in response to the **uuname** command are the names your system administrator has assigned to the remote systems linked by BNU.

8.4.2 Option Used with the uuname Command

You can use one option, called a ***flag*** in the command format, with the **uuname** command.

This is the **-l** flag, which displays the name of your local cluster.

Identifying Compatible Systems

Use the form **uuname** [**option**].

Identify Remote Systems

1. Type the **uuname** command on the command line following the prompt:

```
uuname
```

2. Press the **Enter** key.

   The names of the systems you can access with BNU commands appear on your display:

```
arthur
hera
server
engg
```

Identify Local Cluster

1. To find out the name of your local cluster, add the **-l** flag to the **uuname** command:

```
uuname -l
```

2. Press the **Enter** key.

   The name of your cluster appears on the display:

```
tempest
```

For additional information about the **uuname** command, refer to *AIX Operating System Commands Reference*.

8.5 Communicating with a Remote System

BNU has several commands that enable you to communicate with computers other than the AIX system at which you are currently working. Using these commands, you can:

Connect, over a hardwired asynchronous line, to another AIX system; t another computer running under a UNIX-based operating system like AIX; or possibly to a non-UNIX system, if you have the proper hardware and software.

Connect, over a telephone line, to a remote system or a remot terminal, using modems at both ends of the connection.

Communicate with a remote system over an IBM Token Ring Network, or a Ethernet network, running TCP/IP.

You can make remote connections over both a hardwired line and a telephone line using the BNU command **cu**. The command **ct**, on the other hand, is used only to connect to a remote terminal over a telephone line, using a modem.

Subtopics
8.5.1 Connecting to a Remote Computer (cu)
8.5.2 Connecting a Remote Terminal to an AIX System Using a Modem (ct)

8.5.1 Connecting to a Remote Computer (cu)

The **cu** command enables you to connect with a specified remote computer, login to it, and then perform tasks on it while you remain physically working at your local computer. You are thus logged in on both systems at the same time, and you can switch back and forth between the two computers, performing tasks on both concurrently.

If the remote system is running under the AIX Operating System, you can issue regular AIX Operating System commands on the remote computer to change directories, list directory contents, view files, send files to the print queue, and so on. You can also use special **cu** local commands both to issue AIX Operating System commands on your local system and to perform tasks such as transferring ASCII files between the two systems. You preface these commands, which are discussed in "Using the cu Local ~Commands" in topic 8.5.1.3, with a tilde (~).

For example, suppose you want to transfer a copy of a file from your local system to a remote system for printing. While the first file is printing, you want to edit a second file on the remote system and then send a copy of that file over to your local computer. Following is an overview of the steps you would perform in an operation of this kind:

1. While logged in to your own work station, connect to a specific remote system and then log in to that system.

2. Issue the appropriate local tilde (~) command to transfer the file from the local to the remote system for printing.

3. Issue the **pr** command on the remote system to display the file on the screen or **print** to print the file. You can also issue any other AIX Operating System command on the remote computer, such as **cd** to change to a different directory, **li** to list the contents of a directory, **pg** to examine the file, and so on.

4. Now you can edit another file on the remote computer while the first file is printing. Because the communication link remains open, you can also move easily between the local and the remote systems, checking the status of a job in progress on your local system, monitoring the printing job on the remote system, and so on.

5. When you have finished editing the second file on the remote system, use the appropriate local ~command to send a copy of the updated file back to your local cluster host. You can then continue with other tasks on both your local cluster and the remote system.

Subtopics
8.5.1.1 Form of the cu Command
8.5.1.2 Options Used with the cu Command
8.5.1.3 Using the cu Local ~Commands
8.5.1.4 Additional Information about the cu Command

8.5.1.1 Form of the cu Command

To connect to a remote computer, issue the **cu** command in the following form:

cu [**options**] **system_name**

This form of the command enables you to connect to a remote system over a hardwired line. If your system administrator has set up BNU so that you can communicate with remote systems over a telephone line, this version of the **cu** command also enables you to connect to a remote system using a modem.

Note: For BNU to connect two systems over a telephone line using the **cu** [**options**] **system_name** form of the command, both systems must be attached to modems, and both systems must be set up for this type of communication. For information about customizing the files in the BNU supporting database for remote communications, see *Managing the AIX Operating System*.

Occasionally, you may need to communicate with a remote computer that does not support BNU. You can use a version of the **cu** command to establish such a connection under the following conditions:

Both the local and the remote system must be connected to workin modems.

You must know the telephone number of the remote modem and have valid login on that system.

Under these circumstances, you can connect to the remote computer using the following form of the **cu** command:

cu [**options**] **telno**

where **telno** is the telephone number of the remote modem.

Note: If the remote system is not running under a UNIX-based operating system such as AIX, some of the tilde (~) commands of **cu** will not function.

In general, however, you will probably find that the form of the **cu** command that connects you to a specified system is sufficient for your work.

8.5.1.2 Options Used with the cu Command

You may issue the **cu** command with a number of options, called flags in the command format. Many of these flags perform specialized communications functions, and their default values are set when BNU is installed and customized.

Under certain conditions, however, you may need to specify the following two flags:

-**sspeed**
-**lline**

Specify Transmission Speed (-**sspeed**)

The -**sspeed** flag sets the rate at which data is transmitted to the remote system. The default transmission speed is generally "Any," which instructs BNU to use the rate appropriate for the default (or specified) transmission line. Most modems operate at 300, 1200, or 2400 baud, while most hardwired lines are set to 1200 baud or higher. When transferring data (such as a file) between a local and a remote computer, you may occasionally need specify a 300-baud transmission speed (the lower baud rate results in less interference on the line).

Note: You should **not** have to set the transmission rate as an ordinary practice. The default rate, set when BNU is installed and customized for your site, should be sufficient for most of your work with the **cu** command. Refer to *Managing the AIX Operating System* for additional information about this feature of BNU.

Specify Transmission Line or Device (-**lline**)

The -**lline** flag specifies the name of a device to be used as the line of communication between the local and the remote system. The default device is generally a hardwired asynchronous line, or a telephone line associated with an automatic dialer such as a modem. If your site has a number of lines of communication between local and remote computers, you may occasionally want to specify a particular device, or line, for your **cu** link.

Note: Under ordinary circumstances you should **not** have to specify a line or device. The default device established when BNU is installed should be sufficient. However, if you want to connect to a remote computer and are not certain of the system name, you can issue the **cu** command with the -**l** flag and a variation of the standard device name **tty** (for example, **tty1** or **ttyab**). Check with your system administrator for the device names used at your installation.

You must also specify the remote system with which you want to connect, using one of the following:

system_name (hardwired or modem connection to recognized system)

telno (modem connection to unrecognized system)

Remote System Name (**system_name**)

The **system_name** entry is the name of the remote system, recognized by BNU, with which you want to establish a connection. This is the assigned name of the system, such as **gumby**, **homer**, **phoebus**, and so on. BNU establishes

this connection either over a hardwired line or over a telephone line using a modem, depending on how your system administrator has set up communications between your local cluster and the specified remote system.

Number of Modem on an Unrecognized System (**telno**)

The **telno** entry is the telephone number you want to use to establish a remote connection, using a modem. In this case, the remote system is an AIX or other UNIX-based computer that has not been set up to communicate with your local cluster through BNU. The **telno** entry can be either a local or a long-distance telephone number.

For detailed information about the **cu** command, refer to *AIX Operating System Commands Reference* and to *Managing the AIX Operating System*.

Note: When you work through any of the examples in this book, such as the sample entries in the following quick reference box, remember that the word "enter" means to type the indicated entry and then press the **Enter** key.

+--- **Connecting to a Remote System** ----------+

Use the form **cu** [**options**] **system_name**.

1. To connect to a remote system named **hera** while sitting at your local display station, enter the following:

   ```
   cu hera
   ```

   The system displays the following message:

   ```
   Connected
   ```

   and the screen displays the login prompt for the remote system.

   Note: When connecting to some remote systems, you may need to press **Enter** one or more times before the remote system displays its login prompt.

2. Log in to the remote system.

   You are now logged in to and ready to work concurrently on both your local cluster and the remote system **hera**. You can issue any command on the remote system simply by entering that command following the prompt.

3. To display the contents of a directory called **/usr/pubs/msg** on system **hera**, which is running the AIX Operating System, enter the following:

   ```
   li /usr/pubs/msg
   ```

   You can issue commands on the local cluster by prefixing each command with a tilde. See "Using the cu Local ~Commands" in

topic 8.5.1.3.

Most of the time you will connect to a remote system using the system name, as in the preceding quick reference box. However, you may occasionally need to connect to a remote system whose name you do not know.

In that case, you can issue the **cu** command and connect with a remote system by specifying the name of the device (the hardwired line that actually connects your cluster with the specified remote computer).

The standard device name is prefixed by **tty**. Most hardwired communication lines have names that are variations of the **tty** device name, such as **tty0**, **tty1**, and so on.

When you issue **cu** with the name of a device, you must include the name of the device, or line, preceded by the -**l** flag.

Connecting to a Named Device

Use the form **cu** -**lline**.

1. To connect with a remote system using a hardwired device named **tty2**, enter the following:

   ```
   cu -l tty2
   ```

   The system displays the **Connected** message.

   Note: When connecting to some remote systems, you may need to press **Enter** one or more times before the remote system displays its login prompt.

2. When the remote system displays its login prompt, log in and begin your work. Remember that the connection to your local cluster is still open, so you can perform tasks on both systems concurrently.

You can also use the **cu** command to connect, via a modem, to a remote computer that is not set up to communicate with your cluster through BNU. The remote system must be attached to a modem that can answer the telephone, and you must have a valid login on the remote computer.

Connecting to a Remote System via a Modem

Use the form **cu** [**options**] **telno**.

Local Area Connection

1. To connect to a remote system whose telephone number is 461-1492, type the following:

   ```
   cu 4611492
   ```

2. Press **Enter**

3. After the system displays the **Connected** message, press the **Enter** key.

   Note: When connecting to some remote systems, you may need to press **Enter** one or more times before the remote system displays its login prompt.

4. When the remote system displays its login prompt, log in and begin your work.

Long-Distance Connection

1. To connect to a remote system whose telephone number is 1-612-223-1612, where dialing 9 is required to get an outside dial tone, and you want to transmit data at 300 baud, type the following:

```
cu -s300 9=16122231612
```

   Note: The = causes the dialer to pause momentarily to allow the outside line to be connected.

2. After the system displays the **Connected** message, press the **Enter** key until the login prompt appears, and log in on the remote system.

8.5.1.3 Using the cu Local ~Commands

Once you have issued **cu**, connected to the remote system, and logged in to it, you can issue regular AIX Operating System commands on either the remote system or the local cluster. You can also issue special **cu** commands on the local cluster to transmit ASCII files between the two computers.

When you are logged in to a remote system using a **cu** link, you issue AIX Operating System commands on the remote computer simply by typing the command at the prompt. For example, to list the contents of a directory on the remote system, you would use the command **li**.

However, suppose you want to display the contents of a directory on your local cluster while linked to a remote system through the **cu** command. In that case, you must type a tilde (~) and an exclamation point (!) preceding **li** to indicate that BNU should execute the command on your local system. You would therefore issue the **li** command like this:

```
~!li
```

Note: As soon as you type the ~ and ! (or %, which is used with three local commands), the system displays the name of your local cluster in this form: ~[**system_name**]!. You then type the appropriate command. The complete entry requesting a list of the contents of your current working directory on local cluster **hera** would therefore look like this:

```
~[hera]!ls
```

Following is a list of some of the local ~commands you may use with the **cu** command:

~.
~!
~!**cmd**
~%**cd directory_name**
~%**take from** [**to**]
~%**put from** [**to**]

Terminate Remote Connection (~.)

The ~. (tilde period) characters instruct BNU to log you off the remote system and then terminate the remote connection. In general, however, you should always use the standard procedures to log off the remote system. Then type **~.** at the prompt and press **Enter** to terminate the remote connection.

Note: Entering the **~.** characters always terminates the **cu** process. In some cases where you are connected to the remote system over a telephone line using a modem, however, **~.** does not always successfully log you off the remote system. For this reason, it is generally a good idea to use the appropriate command for the actual logoff sequence.

Escape to Local System (~!)

The ~! (tilde exclamation point) sequence returns you to your local host after you have been working on the remote system. Type **~!** at the prompt and press **Enter**. Then, when you want to return to the remote system, use

the **END OF FILE** sequence to leave the local cluster and work on the remote system. Once you have established the **cu** connection, toggle back and forth between the two systems by entering ~! (to go from remote to local) and **END OF FILE** (to go from local to remote).

Execute cmd Locally (~!**cmd**)

The ~!**cmd** sequence tells BNU to execute the command on the local cluster. Once you have established the **cu** link, you can run commands on your local host only by typing a tilde and an exclamation point before the name of the command.

Change Local Directory (~%**cd directory_name**)

The ~%**cd directory_name** command changes your local working directory from the current directory to the directory specified with the **directory_name** entry.

Copy from Remote to Local (~%**take from** [**to**])

The ~%**take from** [**to**] command "takes" a specified file, copying it from the **remote** system to a specified file on the local cluster. If you do not type a name for the file on the local cluster (the **to** entry), the command copies the specified file from the remote to the local cluster under the same file name. Remember that with the ~%**take** command, the source file is on the remote system.

Copy from Local to Remote (~%**put from** [**to**])

The ~%**put from** [**to**] command "puts" a specified file, copying it from the local cluster to the remote system. Again, if you do not enter a target file name, the command copies the file to the remote system under the same file name. Note that in the case of the ~%**put** command, the source file is on the local cluster.

Note: You can transfer only ASCII files with the ~%**take** and ~%**put** commands. Use the **uucp** command, discussed in "Sending and Receiving Files (uucp)" in topic 8.7, to transfer other types of files. In addition, neither ~%**take** nor ~%**put** checks to ensure that the system transfers the file(s) without errors. For the most reliable file transfers, use the **uucp** command.

+--- **Running Commands on Your Local Computer** ----------+
¦ ¦
¦ Use the form ~!**cmd** to run an AIX Operating System command on your ¦
¦ local cluster. ¦
¦ ¦
¦ View a File ¦
¦ ¦
¦ 1. Issue **cu** and log in to the remote system. ¦
¦ ¦
¦ 2. To display the contents of the file **status10** in the ¦
¦ **/usr/msg/memos** directory on your local host **venus**, enter the ¦
¦ following: ¦
¦ ¦
¦ ~! ¦
¦ ¦
¦ The system responds with the following prompt: ¦

```
~[venus]!
```

Now enter the command name and the name of the file (in this case, the complete path name of the file):

```
~[venus]!pg /usr/msg/memos/status10
```

Open a Virtual Terminal

Note: This applies only if your local work station is a PS/2.

1. After issuing **cu** and logging in on the remote system, open a virtual terminal on your local host **hera**. Type the following:

```
~!
```

When the system responds with the name of your local host, enter the following to open a shell:

```
~[hera]!open sh
```

2. Press **Enter**.

Changing Directories on Your Local System

Use the form ~%**cd** to change from one directory to another directory on your local cluster.

1. Enter **cu** and log in to the remote system.

2. Now change from your current local working directory **/usr/msg** to directory **/adm/msg**, also on your local host **zeus**. Type the following:

```
~%
```

The system responds with the name of your local system, prompting you to enter the command:

```
~[zeus]%
```

3. Enter the **cd** command and the name of the directory following the **~[zeus]%** prompt:

```
cd /adm/msg
```

You can also transfer files between the local and the remote systems during a **cu** connection.

Note: When you use the ~%**put** and ~%**take** commands to transfer files, make sure that the target directory (the one to which you are copying the source files) already exists on the specified system. Unlike the **uucp** command, these **cu** local ~ commands do not create intermediate directories during file transfers.

Making a Remote-Local Transfer

Use the form ~%**take from** [**to**] to copy a file from a remote system to your local cluster.

Copy to Same File Name

1. Enter the **cu** command and then log in to the remote system.
2. To transfer a copy of the file **/u/amy/test1** from the remote system to your local cluster, enter the following:

```
~%take /u/amy/test1
```

where **/u/amy** is also the name of an existing directory on your local cluster. This command copies the file to the local system under the same file name, **test1**.

Copy to Different File Name

1. Enter **cu** and log in to the remote system.
2. To copy the file **u/amy/test1** from the remote to the local cluster under a different file name, enter the following:

```
~%take /u/amy/test1 /usr/dev/amy/tmptest
```

Making a Local-Remote Transfer

Use the form ~%**put from** [**to**] to copy a file from your local cluster to a remote system.

Copy to Same File Name

1. Enter the **cu** command and then log in to the remote system.
2. To copy the file **/usr/pubs/geo/ch2a** from the local cluster to the remote system to which you are connected, enter the following:

```
~%put /usr/pubs/geo/ch2a
```

where **/usr/pubs/geo** is also the name of an existing directory on the remote system. This command copies the file to the remote system under the same file name, **ch2a**.

Copy to Different File Name

1. After executing **cu** and logging in to the remote system, copy the file **/usr/pubs/geo/ch2a** from the local to the remote system under a different file name:

```
~%put /usr/pubs/geo/ch2a /u/geo/part2
```

2. Press **Enter**.

8.5.1.4 Additional Information about the cu Command

Issue **cu** and press the **Enter** key. The system displays the message **Connected**. Then press **Enter** again. When the remote system displays its login prompt, log in and begin your work.

Do not use the **system_name** flag in conjunction with the -**lline** flag. You can use the -**sspeed** flag with either -**l** or **system_name**, but you cannot use all three flags in the same command line. If you do, **cu** connects to the first available line for the requested system name, ignoring the specified line and speed.

You can issue **cu** to connect cluster **X** to system **Y**, log in to system **Y**, and then issue **cu** again on system **Y** to connect to system **Z**. You then have one local computer, cluster **X**, and two remote computers, systems **Y** and **Z**.

You can run AIX Operating System commands on system **Z** simply by logging in and issuing the command. You can run commands on cluster **X** by prefixing the command with a single tilde (~**cmd**). You can also run commands on system **Y** by prefixing the command with two tildes (~~**cmd**).

In general, a single tilde causes the specified command to be executed on the original local cluster, and two tildes cause the command to be executed on the next system on which you executed **cu**.

Remember that the ~! sequence takes you from the remote system to th local cluster. To return to the remote system from the local cluster, use the **END OF FILE** sequence.

For more information about the **cu** command, refer to *AIX Operating System Commands Reference*.

8.5.2 Connecting a Remote Terminal to an AIX System Using a Modem (ct)

The **ct** command enables a user on a remote ASCII terminal, such as an IBM 3161 or a DEC VT100, to communicate with an AIX system over a telephone line attached to a modem at each end of the connection. The user on the remote terminal can then log in and work on the AIX system.

A user on the local cluster issues **ct** with the appropriate telephone number to call the modem attached to the remote terminal. When the connection is established, **ct** issues a login prompt that is displayed on the remote terminal screen. The user on the remote terminal enters his or her AIX login name at the prompt, and AIX opens a new shell. The user at the remote terminal then proceeds to work on the AIX system just like a local user.

Note: In order to establish a **ct** connection, the user on the remote terminal generally contacts a user on the AIX system (with a regular phone call) and asks that user to issue the command. If such connections occur regularly at your site, however, your system administrator may prefer to set up BNU in such a way that a specified local cluster automatically issues **ct** to one or more specified terminals at certain designated times. Refer to *Managing the AIX Operating System* for information about customizing BNU for use at your site.

The **ct** command is useful in the following situations:

When a user working offsite needs to communicate with an AIX Syste under strictly supervised conditions. Because the local host contacts the remote terminal, the user on that terminal does not need to know the telephone number of the local cluster. Additionally, the AIX System user issuing **ct** can monitor the work of the remote user.

When the cost for the telephone connection should be charged either t the local site or to a specific account on the calling AIX System. Assume that the user on the remote terminal has the appropriate access permissions and can make outgoing calls on the attached modem. That user can call the specified AIX System, log in, and issue the **ct** command with the phone number of the remote terminal but **without** the **-h** flag. The local host hangs up the initial link so that the remote terminal is free for an incoming call and then calls back to the terminal. This process is similar to making a collect call.

When you issue **ct** to connect to a remote terminal, you will find the following features of the command useful under certain circumstances:

You can instruct **ct** to continue dialing the number until the connection is established or a set amount of time has elapsed.

You can specify more than one telephone number at a time to instruc **ct** to continue dialing each modem until a connection is established over one of the lines.

Normally, **ct** dials the number specified in the command line, reaches the modem attached to the remote terminal, and displays the AIX login prompt. If there are no free lines, however, **ct** displays a message to that effect and asks if you want to wait for one.

If you reply **no**, **ct** hangs up. If you reply that you do want to wait for a free line, **ct** prompts for the number of minutes to wait. The command

continues to dial the remote system at one-minute intervals until the connection is established or the specified amount of time has elapsed.

Subtopics

8.5.2.1 Form of the ct Command

To connect to a remote terminal, issue the **ct** command in the following form:

ct [options] telno

8.5.2.2 Options Used with the ct Command

You may issue the **ct** command with a number of options, called flags in the command format. Following are several of these options:

-**wn**
-**sspeed**
-**h**

You must also enter the telephone number of the remote terminal:

telno

Specify Wait Time (-**wn**)

The -**wn** flag enables you to specify the maximum amount of time that **ct** waits for a line. You type the command and then the -**w** flag, followed immediately by the amount of time, which you enter as minutes (**-w5**). The **ct** command then dials the remote modem at one-minute intervals until either the connection is established, or the specified number of minutes has passed.

Entering this flag on the command line suppresses the messages that **ct** normally displays if it cannot make the connection. Instead of asking whether to wait for a free line and then prompting for the wait time, **ct** continues to dial for the specified amount of time when you issue the command with the -**wn** flag.

No Hangup (-**h**)

Normally, **ct** hangs up on the current call in order to respond to a call coming in to your modem from another modem. The -**h** flag instructs **ct** not to break the current connection in order to answer an incoming call.

Specify Transmission Rate (-**sspeed**)

The -**sspeed** flag enables you to specify the rate at which **ct** transmits data. The default speed is 1200 baud. Enter this flag when you want to connect to a remote terminal using a modem set to another baud rate, such as 300 baud (often used to transfer files) or 2400 baud (for high-speed transmissions).

For detailed information about the **ct** flags, refer to *AIX Operating System Commands Reference*

Specify Telephone Number (**telno**)

You must specify the phone number of the remote modem. You can enter a local or a long-distance number, and you can specify secondary dial tones such as 9 for an outside line, or an access code.

Use an equal sign (=) following a secondary dial tone (**9=**), and an appropriately placed minus sign (-) for delays (**687-5092**). Telephone numbers may contain up to 31 characters and may include digits from 0 to 9, minus and equal signs, asterisks (*), and pound signs (#).

For detailed information about the **ct** flags, refer to *AIX Operating System Commands Reference*.

+--- **Connecting to a Remote Terminal** ----------+

Use the form **ct** [**options**] **telno**.

Dial Internal Number

1. To dial a modem attached to a remote terminal with the internal telephone number 7-6092, type the following:

```
ct 76092
```

2. Press **Enter**.

   The system responds:

```
Allocated dialer at 1200 baud
Confirm hang_up? (y to hang_up)
```

   Press **y** to hang up any other phone lines currently in use and establish your **ct** connection. Press **n** to cancel the command.

Dial External Local Number

1. To dial a modem attached to a remote terminal with a local telephone number, specifying 9 for an outside line and a two-minute wait for the modem line:

```
ct -w2 9=6340043
```

2. Press **Enter**

Dial Long-Distance Number

1. To dial a modem on a remote terminal with a long-distance number, specifying an outside line and a five-minute wait:

```
ct -w5 9=15023597824
```

2. Press **Enter**.

8.6 Running Remote Commands (uux)

The **uux** command enables you to run a command on a designated remote system while continuing with other work on your local cluster.

The command first gathers various files from the designated systems, if necessary. It then runs a specified AIX Operating System command on a designated system, if possible. If the command you issue on the designated system produces some type of output, such as the **cat** or **diff** command, you can instruct **uux** to place that output in a specified file on any specified AIX site.

Note: You can use the **uux** command on any AIX system configured to run the specified command. For security reasons, however, certain sites may restrict the use of particular commands. Some sites, for example, may permit access only to the **mail** command.

Subtopics

8.6.1 Form of the uux Command

You can enter the **uux** command in either of the following forms:

uux [**options**] "**commandstring** > **destination_name**"

uux [**options**] **commandstring** \{**destination_name**\}

8.6.2 Options Used with the uux Command

You may issue the **uux** command with a number of options, called flags in the command format. Following are three of these flags:

-**n**
-**z**
-**j**

No Notification Message (-**n**)

Normally, the **uux** command notifies you through the mail system about whether the command executed successfully on the designated system. The -**n** flag instructs **uux** not to send you this notification.

Failure Message Only (-**z**)

The -**z** flag instructs **uux** to notify you only if the command fails to execute successfully on the designated system. In that case, **uux** sends you notification about the failure through the mail.

Note: The -**n** flag and the -**z** flag are mutually exclusive; you may use one or the other with **uux**, but not both.

Display Job ID (-**j**)

The -**j** flag tells **uux** to display the job identification number of the process that is running the remote command. You can use this **jobid** with the BNU command **uustat** to check the status of the remote command, or use it with the **uustat** -**k** (kill) flag to terminate the remote command before it finishes executing.

Note: Refer to "Getting Status Information about BNU Jobs (uustat)" in topic 8.9.1 for information about the **uustat** command.

For a complete list of the flags available with the **uux** command, refer to *AIX Operating System Commands Reference*.

You must also enter the name of the command you want to run on the remote system. In addition, you must also specify the site and file in which you want to store the output of the remote command.

Name of Remote Command (**commandstring**)

This name, represented by the **commandstring** entry, may be any AIX Operating System command accepted by the designated system.

To specify the system on which you want to run the command, type the name of the system, an exclamation point (!), and the command name:

system_name!commandstring

The section "Additional Information about the uux Command" in topic 8.6.4 contains examples of this usage.

Name of Destination System and File (**destination_name**)

The **destination_name** entry indicates the system and file in which you want to store the output of the remote command.

Suppose, for example, that you want a listing of all the files in a certain directory on a remote system. Rather than having the AIX **ls** simply display the file names on the remote system, you can specify that you want the **uux** command to place the directory listing in a file on your own cluster by entering the appropriate destination name.

Note: The **destination_name** is the path name to the location in which you want to store the output of the executed command. For information about path names, refer to "Path Names Used with BNU Commands" in topic 8.6.3.

You can type the destination name in either one of two ways:

commandstring > **destination_name**"

commandstring \{**destination_name**\}

Notice the set of double quotation marks (" ") in the first form. If you use the > (greater than) symbol to direct the output of the remote command to the destination name, then you must type one double quotation mark (") before the name of the command, and another double quotation mark (") following the destination name: ". . .>. . .".

If you use the second form, you must type a backslash, a left brace (also called a curly bracket), the destination name followed by a second backslash, and a right brace: \{. . .\}. You need to include the backslashes because the left and right braces are special characters to the shell command interpreter.

See "Additional Information about the uux Command" in topic 8.6.4 for examples of these forms.

Running a Remote Command

1. To concatenate two files and direct the output to a third file, enter the **uux** command in either of the following forms:

   ```
   uux "zeus!cat zeus!/u/amy/f1 hera!/usr/amy/f2
   > zeus!/u/amy/catout"
   ```

   or

   ```
   uux zeus!cat zeus!/u/amy/f1 hera!/usr/amy/f2
   \{zeus!/u/amy/catout\}
   ```

2. Press **Enter**.

   Either form of **uux** executes the **cat** command, which is stored on system **zeus**. The **cat** command combines the file **f1**, located in the directory **/u/amy** on **zeus**, with the file **f2**, located in **/usr/amy** on system **hera**. The command places the new file on system **zeus** under the file name **catout** in the directory **/u/amy**.

8.6.3 Path Names Used with BNU Commands

Path names used with BNU commands are essentially the same as path names used with AIX Operating System commands, but BNU path names often include the name of the remote system.

Full pathname. A full path name lists all the directories along the route from the root directory to a specific directory or file, ending with the name of the final directory or file. By convention, the elements in a path name are separated by slashes (/).

Relative pathname. A relative (or partial) path name lists the route to a specific directory or file relative to the current directory. If you are working in a directory named **/usr/bin/lgh/reports**, then typing **month8** without a leading slash identifies the file called **/usr/bin/lgh/reports/month8**. You may always use relative path names with the **cu**, **uucp**, and **uux** commands, and with the name of the source file in the **uuto** command.

Note: Relative path names may not always work with all commands. If you are having trouble accessing a file with a relative path name, re-issue the command using the full path name to the file.

user pathname. The tilde (~) is a short-hand way of identifying part of a path name. In this case, ~**user pathname** refers to the login directory of the person identified as **user**.

uucp/filename. In this case, the entries preceding the file name refer to the public directory on the designated system. BNU uses this directory, **/usr/spool/uucppublic**, for sending and receiving information. The ~**uucp** entry is a short-hand way of specifying the public directory.

System_name!pathname. This is the syntax BNU uses to identify the path to a file on another system. The following example identifies the file **new** in the directory **research** on a system named **merlin**:

```
merlin!/research/new
```

System_name!system_name!pathname. This is the path name to a file on another system that goes through one or more other intermediate systems. In the following example, the path name specifies the file **cells** in directory **research** on system **merlin**, which is reached first through system **zeus** and then through system **venus**.

```
zeus!venus!merlin!/research/cells
```

8.6.4 Additional Information about the uux Command

If you request a command that the remote system cannot run, you wil receive a mail message to that effect from the remote system.

To run commands on more than one system, type the information o separate command lines:

```
uux merlin!print /reports/memos/charles
uux zeus!print /test/examples/examp1
```

In addition to the two forms of the destination name that you can us with the **uux** command, you can also represent your local cluster in several different ways.

For example, enter the following to run the **diff** command, which is on your local site **hera**, to compare the file **/u/f1** on system **venus** with the file **/u/f2** on system **merlin**. Specify that the output of the **diff** command should be placed in the file **/u/f3** on your local host:

```
uux "hera!diff venus!/u/f1 merlin!/u/f2 > hera!/u/f3"
```

You can also enter the destination name in the following form:

```
uux hera!diff venus!/u/f1 merlin!/u/f2\{hera!/u/f3\}
```

These examples used the name of the local host, **hera**, followed by an exclamation point. You can also represent the local host using just an exclamation point, as in the following example:

```
uux "!diff venus!/u/f1 merlin!/u/f2 > !/u/f3"
```

Both the system name and the exclamation point are optional. Following is the easiest way to represent the local host:

```
uux "diff venus!/u/f1 merlin!/u/f2 > /u/f3"
```

Of course, in any of these examples, you can enter the destination path name using the \{. . .\} sequence.

Note: The output file must be writeable, which means that the permission for the file allows you to place data in it. If you are uncertain about the permission status of a specific target output file, direct the results of the command to the public directory, **/usr/spool/uucppublic**.

When specifying the destination for the output of a command, you ma use a full name or a path name preceded by ~**user**. In this case, replace the **user** entry with a login name that refers to the user's login directory.

When specifying the path name for a file you want to use as the sourc in running commands such as **diff** or **cat**, you may include the following shell pattern-matching characters, which the remote system can interpret:

```
?
*
[ (left bracket)
```

] (right bracket)

Precede these characters with a backslash (\) or place them between a pair of quotation marks (" . ."), so the local shell cannot interpret the characters before **uux** sends the command to the remote system.

Do not use pattern-matching characters in destination names.

If you use the following shell characters, precede them with backslash (\) or place " . ." around the individual character or the entire command string:

< (less than)
> (greater than)
;
|

Do not use the shell redirection characters (<< and >>) because they do not work in the BNU program.

The exclamation point representing a remote system in BNU syntax ha another meaning in c shells (**csh**). When using a BNU command such as **uux** running in a c shell, you must place a backslash (\) before the exclamation point.

The **csh** command will also interpret the ~user syntax on the local cluster. To allow **uux** to interpret the user syntax, place it in quotes or precede the ~ with a backslash.

For example, to get the first field from each line of the password file on remote system **hera** and place the output in the file **passw.cut** in the public directory on your local system, use either of the following forms:

```
uux "cut -f1 -d: hera\!/etc/passwd > ~uucp/passw.cut"
```

or

```
uux cut -f1 -d: hera\!/etc/passwd \{~uucp/passw.cut\}
```

Remember that ~**uucp**/ is a short-hand way of specifying the public directory, **/usr/spool/uucppublic**.

For additional information about the **uux** command, see *AIX Operating System Commands Reference*. For information about how the **uux** command works, see *Managing the AIX Operating System*.

8.7 Sending and Receiving Files (uucp)

In general, you will probably use BNU primarily to send and receive files. The BNU command **uucp** and its options enable you to copy one or more **source_files** from one UNIX system (such as a Personal System/2 running under the AIX Operating System) to one or more **destination_files** on another UNIX system that supports BNU.

You can use **uucp** to copy files between and among systems in the following ways:

within your local cluste

between a local cluster and a remote syste

between two remote systems

Subtopics

8.7.1 Form of the uucp Command

The **uucp** command has the following form:

uucp [**options**] **source_file(s) destination_name**

8.7.2 Options Used with the uucp Command

You may enter **uucp** with one or more options, which are called flags in the command format. Following are several of these available flags:

-d
-f
-j
-m
-nuser
-sfile

Create Intermediate Directories (-d)

The **-d** flag, which is on by default, creates any intermediate directories needed to copy a source file to a destination file on a remote system. For example, instead of first creating a directory and then copying a file to it, you can simply enter **uucp** with the destination path name, and the BNU Program will create the required directory.

No Intermediate Directories (-f)

The **-f** flag instructs **uucp** not to create any intermediate directories during the file transfer. Use this flag if the destination directory already exists and you do not want the BNU Program to write over it.

Display Job ID (-j)

The **-j** flag tells **uucp** to display the job identification number of the transfer operation. You can use this **jobid** with the **uustat** command to check the status of the transfer, or use it with **uustat -k** (kill) to terminate the transfer before it is completed.

Note: Refer to "Getting Status Information about BNU Jobs (uustat)" in topic 8.9.1 for information about the **uustat** command.

Mail Message to Sender (-m)

The **-m** flag sends you a mail message when the source file is successfully copied to the destination file on a remote system. The message goes to your mail box, **$HOME/.newmail**. The **mail** command does not send a message for a local transfer.

Notify Recipient (-nusername)

The **-nusername** flag notifies the recipient on the remote system identified by the **username** entry that a file has been sent. Again, the mail system does not send a message for a local transfer.

You must also specify the name of the source file that you're copying to another system, as well as the name of the destination file on the target system (that is, the file into which you're copying the source file).

Path Name of Source File (source_file)

The **source_file** entry is the path name of the file that you want to send or receive. For detailed information about path names used with the BNU Program, refer to "Path Names Used with BNU Commands" in topic 8.6.3.

Path Name of Destination File (destination_name)

The **destination_name** entry is the path name of the file (or directory) to which the copy is being sent. Again, see "Path Names Used with BNU Commands" in topic 8.6.3 for detailed information about path names.

For information about other **uucp** options, see *AIX Operating System Commands Reference*.

8.7.3 Additional Information about the uucp Command

A file path name may contain the following shell pattern-matchin characters:

?
*
[(left bracket)
] (right bracket)

See "Matching Patterns" in topic 4.4.5 for information about using these characters.

If you are working in a c shell **csh**) or a Bourne shell (**sh**) on your local system and want to copy multiple files from a local directory to a directory on a remote system, you can use shell pattern-matching characters in the path name of the source file.

Be sure to place a backslash immediately following the name of the remote system:

```
uucp out* hera\!~uucp/OF1
```

You can send files to the public directory on the remote system, as in the example, or to a specified directory. BNU creates the destination directory (**OF1** in the example) if it does not already exist.

While working in a c shell on your local system, you can also cop multiple files from a remote system to a local directory . Use one of the following forms:

```
uucp zeus\!/u/amy/out'*' ~uucp
```

or

```
uucp "zeus\!/u/amy/out*" ~uucp
```

In the first example, the pattern-matching character * in the path name of the source files is enclosed in single quotation marks. In the second example, the entire path name of the source files is enclosed in double quotation marks. In both examples, the multiple source files are copied to the public directory on the local system.

Sending files to arbitrary destination names on other systems o getting files from arbitrary source names on other systems often fails because of security restrictions. Such failures will occur unless the files give read or write permission to the user and/or group, or to others.

Note: When **uucp** transfers a file, it gives read and write permissions to that file to the owner, the group, and all others. To change these permissions, use the **chmod** command, discussed in *AIX Operating System Commands Reference*. Refer also to "Changing Permissions--The chmod (Change Mode) Command" in topic 3.14.2, and to the discussion on permissions in *Managing the AIX Operating System*.

You can use **uucp** to transfer your own protected files, as well as files in protected directories that you own.

The sections that follow discuss the procedures for local and remote file transfers.

8.7.4 Local Transfers

In a local transfer, the BNU command **uucp** works almost exactly like **cp**, the **copy** command. You can copy a file into another file in the same directory or from one directory into another directory. See *AIX Operating System Commands Reference* for information about the **cp** command.

In order to use **uucp** to transfer files, you must know the path name of the source file (the original location in which the file is stored), as well as the destination name (the file or directory in which you want to place the copy of the file).

For information about BNU path names, refer to "Path Names Used with BNU Commands" in topic 8.6.3.

Transferring a Single File Locally

Use the form **uucp source_file(s) destination_name**.

Copy Within the Same Directory

1. To copy the file **prog.b** into the file **prog.c**, which is located in same directory on your local cluster, type the following:

```
uucp prog.b prog.c
```

2. Press **Enter**.

   If the file **prog.c** does not exist, **uucp** creates it.

Copy to Another Directory

1. To copy the **jones** file into a new directory named **clients** on your local cluster, type the following:

```
uucp /usr/bin/work/jones /usr/bin/clients
```

2. Press **Enter**.

   The system creates the **clients** directory and then copies the **jones** file into it. The new path name of the **jones** file is **/usr/bin/clients/jones**.

Transferring Multiple Files Locally

Use the form **uucp source_file(s) destination_name**.

1. To copy several files at the same time from different directories on your local cluster to a new local destination, type the following:

```
uucp listing /clients/smith /usr/sales/tom
```

2. Press **Enter**.

   In this case, the **listing** file is in the current working directory, and the **smith** file is in a directory called **/clients**. This command copies both files into an existing directory called **/usr/sales/tom**.

8.7.5 Remote Transfers

To transfer a file to a remote system, add the name of the remote system, with an exclamation point, to the path name of the target destination file that will receive the copy. Use the following form:

system_name!

In general, files transferred from a local to a remote system are placed in the public directory on the remote computer. The full name of this directory is **/usr/spool/uucppublic**, but you can use a tilde (~) and the name of the **uucp** user (~**uucp**) as a short-hand way of specifying this directory.

Subtopics
8.7.5.1 Sending Files to a Remote System
8.7.5.2 Receiving Files from a Remote System

8.7.5.1 Sending Files to a Remote System

You can send files to a remote system directly or through one or more intermediate systems.

Sending Files Directly to a Remote System

Use the following form:

uucp [**options**] **source_file(s) system_name!destination_name**

1. To send a copy of your file called **/meteors** to the file **/solar/stats** in the public directory on the remote system **galaxy**, type the following at the prompt:

   ```
   uucp /meteors galaxy!~uucp/solar/stats
   ```

2. Press **Enter**.

   Remember that the **~uucp** entry preceding the name **/solar/stats** is a short-hand method of specifying the public directory. You can also enter the full destination path name:

   ```
   galaxy!/usr/spool/uucppublic/solar/stats
   ```

Sending Files through Intermediate Systems

Use the following form:

uucp [**options**] **source_file(s) system_name!destination_name**

To send files through intermediate systems, add the name of each remote system followed by an exclamation point, like this:

system_name!

1. To send a copy of **/meteors** to the file **/solar/stats** on system **galaxy** by way of the intermediate system **milkyway**, type the following:

   ```
   uucp /meteors milkyway!galaxy!~uucp/solar/stats
   ```

2. Press **Enter**.

   BNU routes the transfer from your system through system **milkyway** and then to the public directory on system **galaxy**.

8.7.5.2 Receiving Files from a Remote System

You can receive files directly from a remote system, or through one or more intermediate systems. This operation works only if **uucp** has write permission in the directory that receives the files.

Receiving Files from a Remote System

Use the following form:

uucp [**options**] **system_name!source_file(s) destination_name**

1. To request that **uucp** gets the file **/cells/type1** from system **biochem** and stores it in a file called **/drmsg/research** on your local cluster, type:

   ```
   uucp biochem!/cells/type1 /drmsg/research
   ```

2. Press **Enter**.

For additional information about the **uucp** command, refer to *AIX Operating System Commands Reference* and *Managing the AIX Operating System*.

8.8 Another Method for Transferring and Handling Files (uuto, uupick)

In addition to the **uucp** command, BNU has another command that enables you to copy files from one AIX system to another AIX system. The **uuto** command actually uses **uucp** to transfer the specified file(s), but **uuto** makes the whole process easier for both the sender and the recipient.

The **uuto** command sends a specified file or files from one system to a specific user ID on another system. The command places the copied file(s) in the public directory on the recipient's computer, and the system notifies the recipient that a file has arrived.

Once the file is in the BNU public directory, the user issues the **uupick** command, which displays a message that file **name** has arrived from system **name**. The user then enters one of the **uupick** options for handling the file, such as deleting it or moving it to another directory.

Following is an overview of the way in which you can use the **uuto** and **uupick** commands to send and receive a file:

1. The sender issues the **uuto** command to copy one or more files to a specific user ID on another AIX system.

2. The **uucp** command then sends the file(s) to the BNU public directory, **/usr/spool/uucppublic**. In this case, **uucp** also creates (if it doesn't already exist) an additional directory called **receive**, plus the directory **/user/system**. The full path name to the copied file is therefore

   /usr/spool/uucppublic/receive/**user_ID/system/file**

   The **rmail** command then notifies the recipient that a file (or files) has arrived.

3. The recipient issues the **uupick** command.

4. The **uupick** command searches the public directory for files sent to the recipient and notifies the recipient about each file it locates.

5. Using a series of **uupick** options, the recipient saves or deletes each file.

You can also use the **uuto** and **uupick** commands to transfer files to a specific ID within the local cluster. Again, **uuto** places the copied file(s) in the BNU public directory on the local cluster.

More information on **uuto** and **uupick** follows. Refer also to *AIX Operating System Commands Reference*.

Subtopics

8.8.1 Sending Files to a Specific ID (uuto)

The **uuto** command copies one or more source files from one AIX system to a specific user on another AIX system. The command stores the file in the public directory on the destination system until the specified user issues the **uupick** command to locate and handle the file.

Subtopics
8.8.1.1 Form of the uuto Command
8.8.1.2 Options Used with the uuto Command
8.8.1.3 Additional Information about the uuto Command

8.8.1.1 Form of the uuto Command

The **uuto** command has the following form:

uuto [**options**] **file_name destination_name**

8.8.1.2 Options Used with the uuto Command

The following options, called flags in the command format, are available for use with the **uuto** command:

-m
-p

Mail Message to Sender (-m)

The **-m** flag notifies you, the sender, when the **uuto** command has successfully copied the source file(s) to the specified user ID on the specified system.

Copy File to Spool Directory (-p)

The **-p** flag sends the source file(s) to the spool directory on your local cluster before actually transferring the copy of the file(s) to the public directory on the specified system. Without this flag, **uuto** copies the source file(s) directly to **/usr/spool/uucppublic/receive/user_ID/system/file(s)**.

You must also type the name of the file you want to send to the specified user and the path name to the destination of the file transfer.

Path Name of Source File (**file_name**)

The **file_name** entry is the path name of the source file. This may be a simple file name if the file you are sending is in the directory from which you are issuing the **uuto** command. Otherwise, give the complete path name of the file.

Path Name of Destination (**destination_name**)

The **destination_name** is the path name to the specific location to which you want to copy the source file. This path name **must** include the user identification of the person to whom you are sending the file. The **destination_name** has the following form:

system!user_ID

where **system** is the name of the remote computer. When copying a file from one location to another location on your local cluster, the **destination_name** can be simply the name of the user to whom you are sending the file.

Sending Files to a Specific User ID

Use the form **uuto** [**options**] **file_name destination_name**.

1. To send a file called **/usr/bin/data/private** to a user with the ID **monique** on remote system **venus**, type the following:

```
uuto /usr/bin/data/private venus!monique
```

2. Press **Enter**.

   The **uuto** command copies the file and sends it to the public

directory on system **venus**. The **rmail** command then sends user **monique** a mail message that the file has arrived. Monique issues the **uupick** command to locate and handle the transferred file.

For more information about using **uupick**, see "Locating Files for a Specific ID (uupick)" in topic 8.8.2.

8.8.1.3 Additional Information about the uuto Command

If you use **uuto** for a transfer within the local cluster, omit the system name in the command format.

Instead of typing **uuto /usr/research/file1 zeus!amy**, you can send this file to user **amy** by entering **uuto /usr/research/file1 amy**. This form works only if you and **amy** are both on system **zeus**. No mail message is sent to the recipient in a local transfer of this kind.

8.8.2 Locating Files for a Specific ID (uupick)

When **uuto** copies a file or files to your user ID, BNU places the file(s) in the public directory on your local cluster under the name **/usr/spool/uucppublic/receive/user_ID/system /file(s)**, and **rmail** notifies you that the file has arrived.

When you receive this message, issue the **uupick** command to complete the transfer and handle the file(s).

Subtopics
8.8.2.1 Form of the uupick Command
8.8.2.2 File-Handling Options Used with the uupick Command
8.8.2.3 Receiving and Handling Your Files
8.8.2.4 Options for Handling Your Files

8.8.2.1 Form of the uupick Command

Following is the form of the **uupick** command:

```
uupick
```

8.8.2.2 File-Handling Options Used with the uupick Command

As you can see from its form, **uupick** does not have command flags. It does, however, have options that enable you to handle the file(s) sent to you with **uuto**.

Following is a list of the **uupick** user options; notice that the option is **not** preceded by a hyphen:

asterisk (*)
new-line (carriage return)
a [**dir**]
d
m [**dir**]
p
q or **END OF FILE**
!**command**

After notifying you that a file has been sent from **system**, the **uupick** command displays a question mark (?) as a prompt. This indicates that you can now enter one of the file-handling options.

Display Options (*)

Typing an asterisk following the **?** prompt instructs **uupick** to display all the file-handling options.

Next File (new-line)

Pressing the **Enter** key signals **uupick** to move on to the next file in the directory.

Move All Files (**a** [**dir**])

The **a** [**dir**] option enables you to move all your **uuto** files currently in the public directory into a specified directory on your local or a remote system. The default is your current directory (that is, the directory you were in when you issued the **uupick** command). You can use either a full path name or a relative path name to specify the directory.

Delete File (d)

The **d** option enables you to delete the specified file.

Move Specified File (**m** [**dir**])

The **m** [**dir**] option enables you to move a specified file to a specified directory. Again, the default is your current directory, and you may use either full or relative path names.

Display File (p)

The **p** option enables you to display (print) the contents of the specified file on the screen of your display station.

Quit uupick (**q** or **END OF FILE**)

The **q** option enables you to leave the **uupick** command without actually doing anything about the file(s) in the public directory. You can also use the **END OF FILE** key sequence to quit the command.

Run Specified Command (**!command**)

The **!command** option enables you to leave the **uupick** command and return to the AIX prompt to run a specified AIX Operating System command. After the command executes, the system automatically returns to **uupick** so you can continue handling the **uuto** files in the public directory.

8.8.2.3 Receiving and Handling Your Files

Use the **uupick** command and the appropriate user options to receive and handle the files sent by **uuto** to the public directory on your local cluster.

Receiving Your Files

Use the form **uupick**.

1. When you receive a mail message that a file has been sent to you from a certain system, enter **uupick** following the prompt on the command line.

   ```
   uupick
   ```

   When you receive the file, a message appears in this form:

   ```
   from system system_name: [file file_name]
   [dir directory_name]
   ?
   ```

   For example, if **amy** on system **zeus** sends you a file named **info**, you receive the following message:

   ```
   from system zeus: file info
   ?
   ```

2. Following the **?** prompt, enter a **uupick** user option indicating how you want to handle the file.

   Continue until you have taken care of all the files or use either the **q** option or **END OF FILE** to stop reviewing files.

To display the list of **uupick** user options that are available for handling your file(s), type an asterisk (*) below the **?** prompt.

8.8.2.4 Options for Handling Your Files

Following are the options that you can use to handle the files sent to you with the **uuto** command. Remember that you can use these options only after you have issued the **uupick** command and the **?** prompt appears on your screen.

Option	Action
d	Delete the file.
m [dir]	Move the file to directory **dir**. The default moves the file to the current directory.
a [dir]	Move all files to directory **dir**. The default moves the files to the current directory.
p	Display the contents of the file on screen.
q (END OF FILE)	Quit without handling any files.
!command	Escape to the shell to run **command** and then return to **uupick**.
*****	Display all options.
new-line (carriage return)	Go to the next file.

Handling Your Files

1. Enter an asterisk (*) on the line below the **?** prompt for a list of options available to handle your files in the public directory:

```
?
*
```

   The system displays the following:

```
usage [d] [m dir] [a dir] [p]  [q] [cntl-d] [!cmd] [*] [new-line]
```

2. Enter the appropriate option, or use the **q** option or the **END OF FILE** sequence to leave the **uupick** command.

8.9 Determining the Status of BNU Jobs

BNU has two commands that enable you to get information about the status of a particular operation: **uustat** and **uulog**.

The **uustat** command reports the status of various BNU operations, including the following:

- File transfers initiated with the **uucp** command (discussed in "Sending and Receiving Files (uucp)" in topic 8.7)
- Commands invoked with the **uux** command that are running on designated systems (discussed in "Running Remote Commands (uux)" in topic 8.6)
- Files copied with the **uuto** command (discussed in "Another Method for Transferring and Handling Files (uuto, uupick)" in topic 8.8).

The system administrator uses the **uulog** command to combine individual log files into a single main log file. The command also displays the contents of the log file, which contains information about activities performed by the **uucico** or **uuxqt** programs.

Note: You do not invoke **uucico** or **uuxqt** directly. Instead, you issue **uucp**, which then invokes **uucico**, or you issue **uux**, which invokes **uuxqt**. For information about these programs, and about the **uulog** command, refer to *Managing the AIX Operating System*.

Subtopics

8.9.1 Getting Status Information about BNU Jobs (uustat)

The **uustat** command displays information about the progress of various jobs initiated with BNU commands. This command is particularly useful in monitoring file transfers requested with the **uucp** and **uuto** commands and command executions requested with the **uux** command.

Note: Refer to "Sending and Receiving Files (uucp)" in topic 8.7, "Another Method for Transferring and Handling Files (uuto, uupick)" in topic 8.8, and "Running Remote Commands (uux)" in topic 8.6 for information about these commands.

In addition, **uustat** gives you limited control over jobs that you have queued to run on a remote computer. Not only can you check the general status of BNU connections to other systems and the progress of BNU file transfers and command executions, but you can also use **uustat** to cancel copy requests invoked with the **uucp** command.

8.9.2 Form of the uustat Command

The **uustat** command has the following form:

uustat [**options**]

The status reports generated by **uustat** are displayed on your display station screen in this basic form:

jobid date/time status system_name user_ID size file

8.9.3 Options Used with the uustat Command

You may enter **uustat** with one or more options, which are called flags in the command format. Following are some of the available flags, which are mutually exclusive:

-a
-k jobid
-m
-q
-r jobid

You can also use either or both of the following flags with **uustat**:

-nuser
-sfile

Display All Jobs in Queue (-a)

The **-a** flag instructs **uustat** to display information about all the jobs in the holding queue, regardless of the user who issued the original BNU command.

Note: For information about the BNU queues, refer to "Additional Information about the uustat Command" in topic 8.9.4.

Cancel Job (-k jobid)

The **-k jobid** flag cancels (kills) the BNU process specified by the **jobid**. This is useful, for example, when you want to cancel a file transfer or copy request, a remote printing job, and so on.

You can cancel a job only if you are the user who issued the original BNU command specified by the **jobid**. (A System Administrator with superuser authority can also cancel BNU requests.)

Note: For additional information about canceling a BNU job, see the description of the **uustat** command in *AIX Operating System Commands Reference*.

Most Recent Attempt (-m)

The **-m** flag reports on the status of your most recent attempt to communicate with another computer through the BNU facility.

For example, the status is reported as SUCCESSFUL if the BNU request executed; if the job was not completed, BNU reports an error message, such as LOGIN FAILED.

Jobs Currently in Queue (-q)

The **-q** flag lists the jobs currently queued for each system. These jobs are either waiting to execute or in the process of executing. If a status file exists for the system, BNU reports its date, time, and the status information. Once the process is completed, BNU removes the job listing from the current queue.

Rejuvenate Specified Job (-r jobid)

The **-r jobid** flag rejuvenates the BNU process specified by the job

identification number. This flag enables you to mark files in the holding queue with the current date and time, thus ensuring that the cleanup operation does not delete these files until the job's modification time reaches the end of the allotted period.

Note: For information about the BNU queues, refer to "Additional Information about the uustat Command" in topic 8.9.4. For information about cleaning up BNU queues, refer to the discussions of the **uucleanup** command in *AIX Operating System Commands Reference* and in *Managing the AIX Operating System*.

Jobs on Specified System (-ssystem)

The -**ssystem** flag reports the status of all BNU requests that users have entered to run on the computer specified by the **system** entry.

Jobs Requested by Specified User (-uuser_ID)

The -**uuser_ID** flag reports the status of all BNU requests that have been entered to run by the user specified in the **user_ID** entry.

Note: You can use both the -**ssystem** and the -**uuser_ID** flags with the **uustat** command to get a status report on all BNU requests entered by a specified user on a specified system. For detailed information about the options available with the **uustat** command, refer to *AIX Operating System Commands Reference*.

Determining the General Status of BNU Jobs

Use the form **uustat** [**options**].

All Jobs in the Holding Queue

1. To display the status of all BNU jobs in the holding queue, type the following after the prompt:

```
uustat -a
```

2. Press **Enter**.

   Refer to "Additional Information about the uustat Command" in topic 8.9.4 for a sample of the output generated by this command.

All Jobs in the Current Queue

1. To report the status of all the BNU jobs either currently executing or queued to run on each system, type the following:

```
uustat -q
```

2. Press **Enter**.

   Sample output for this example is shown in "Additional Information about the uustat Command" in topic 8.9.4.

Determining the Status of Specific Jobs

Use the form **uustat** [**options**].

All Jobs in Holding Queue for Specified System

1. To report the status of all jobs in the holding queue for a specified system, enter the following, including the **system** name:

```
uustat -s venus
```

2. Press **Enter**.

   See "Additional Information about the uustat Command" in topic 8.9.4 for the sample output for this command.

All Jobs Requested by Specific User

1. To report the status of jobs requested by a specified user, enter the following, including the **user_ID**:

```
uustat -u amy
```

2. Press **Enter**.

   Sample output for this example is shown in "Additional Information about the uustat Command" in topic 8.9.4.

8.9.4 Additional Information about the uustat Command

The **uustat** command produces information about the status of various requests that users have issued with one of the BNU commands. The type of information that **uustat** displays depends on the flag you enter with it.

Because the **-q** and **-a** flags produce different types of listings, this chapter distinguishes between the two types of information by referring to the output of the **uustat -q** command as the current queue, and the output of the **uustat -a** command as the holding queue.

The current queue lists the BNU jobs either queued to run or currently executing on one or more remote systems. The holding queue lists all jobs that have not executed during a set period of time.

Note: After the set time period has elapsed, the entries in the holding queue are deleted either manually, with the BNU **uucleanup** command, or automatically, with the file **/usr/spool/cron/crontabs/uucp**, which is started by the daemon **/etc/cron**. For detailed information about cleaning up BNU queues, refer to the discussions of the **uucleanup** command in *AIX Operating System Commands Reference* and *Managing the AIX Operating System*.

When you enter the **uustat -a** command to examine the status of all BNU jobs in the holding queue, the system displays the following type of output:

```
heraC3113   11/06-17:47 S hera   amy 289   D.venus471afd8
zeusN3130   11/06-09:14 R zeus   geo 338   D.venus471bc0a
merlinC3120 11/05-16:02 S merlin amy 828   /u/amy/tt
merlinC3119 11/05-12:32 S merlin msg rmail amy
```

The first field is the job ID of the operation, which is followed in the second field by the date and time that the BNU command was issued. The third field is either an S or an R, depending on whether the job is to send or receive a file.

The fourth field is the name of the system on which the command was entered, followed by the **user_ID** of the person who issued the command in the fifth field.

The sixth field is the size of the file or, in the case of a remote execution like the last entry in the example, the name of the remote command. When the size is given, as in the first three lines of the example output, the file name is also displayed.

The file name can be either the name given by the user, as in the **/u/amy/tt** entry, or a name that BNU assigns internally to data files associated with remote executions, such as **D.venus471afd8**.

When you issue the **uustat -q** command to report the status of all the BNU jobs either currently executing or queued to run on each computer, the system displays the following type of output:

```
merlin   2C       09/12-09:14   SUCCESSFUL
hera     4C       09/12-10:02   NO DEVICES AVAILABLE
zeus     1C  (2)  09/12-10:12   CAN'T ACCESS DEVICE
```

This output tells how many command (**C.**) files are waiting for each system. The date and time refer to the last time BNU tried to communicate with that system, and the message at the end of the line reports the status of each interaction. The number in parentheses (2) in the third line of this example indicates that the file has been in the queue for two days.

When you enter the **uustat -ssystem** command, BNU displays the following type of output for the specified system:

```
arthurC3114 11/06-16:50 S arthur  daemon 427 D.venus471994d
arthurN3219 11/05-10:12 S arthur  msg    278 D.hera471eac5
```

The **uustat -uuser_ID** command produces output similar to that produced by the -**s** flag.

The **k jobid** flag cancels a process **only** when that job is still on the local cluster. Once the BNU facility has moved the request to a remote system for execution, you cannot use this flag to kill the remote job.

Issuing **uustat** without any flags displays the status information for your personal BNU jobs (that is, for all the BNU commands that you have issued since the last time the holding queue was cleaned up).

In a status report, a number in parentheses next to the number of command file (a **C.** file) or an execute file (an **X.** file) represents the age in days of the oldest **C.** or **X.** file for that system.

The retry field indicates how many times the local cluster has tried to communicate with a specified remote system since the last successful BNU connection.

For more information about the **uustat** command, refer to *AIX Operating System Commands Reference* and *Managing the AIX Operating System*.

9.0 Chapter 9. Using TCP/IP

Subtopics

9.1 CONTENTS

9.2 About This Chapter

This chapter shows you how to use a **network**, such as Ethernet, to communicate from one PS/2 to another or from a PS/2 to another system. You do this with TCP/IP (Transmission Control Protocol/Internet Protocol).

TCP/IP, which is a Licensed Program available for the AIX Operating System, must be installed as a separate facility. The hardware and software needed for the network must also be installed, if not done previously.

If you need information about either of these procedures, see *AIX Operating System TCP/IP User's Guide*.

9.3 Before You Begin

If your system has not been customized for the network, follow the instructions in Appendix A of the manual *AIX Operating System TCP/IP User's Guide* after the installation of TCP/IP and the network is complete.

Customizing the system for the network includes the following tasks:

Using the **devices** command to add the network device, configure ports for Telnet (the protocol that opens the connection to the system), and record the correct interrupt level

Modifying several file

Defining routes, if needed, using the **route** command.

9.4 Overview of TCP/IP

TCP/IP provides the commands you need to perform the following tasks:

Transfer files between two PS/2s or between a PS/2 and another syste

Log in remotely to another PS/2 or to other system

This chapter discusses the basic commands you need to perform the tasks listed above. For more information, including network management tasks not discussed here, see *AIX Operating System TCP/IP User's Guide* and *Managing the AIX Operating System*.

When using AIX, the user often has a choice between the commands used by UNIX System V and those used by BSD4.3 UNIX. If he has been trained in either of these systems, he may use the commands with which he is already familiar. This speeds up training for the whole installation.

The TCP/IP commands are an example of this cross-compatibility. Here is a list of the functions you can perform, along with their command names from the two systems. These programs are generally equivalent in function, but differ in small details of command options offered and the syntax used. See *AIX Operating System Commands Reference* for details.

Table 9-1. Comparison of TCP/IP Commands

System V	**BSD4.3**	**Function**
tn (telnet)	**rlogin**	Allows the user to login on another host.
ftp	**rcp**	Copies files from one machine to another.
write	**talk**	Allows users to communicate via messages.

In addition, there are some useful BSD4.3 commands which have no System V equivalent. See *AIX Operating System Commands Reference* for details.

Command	**Function**
finger	Gives information on the person connected with any login id.
rsh	Starts a new shell on a remote host.
rwho	Gives a list of users logged on to a remote machine.

9.5 Requesting Information about Users (finger)

To request information about users on a specified system, use the **finger** command.

The **finger** command has the following basic format:

finger [**user**]**@system_name**

The **finger** command displays information about current users on the specified system. If you do not provide a user name, the **finger** command provides a list of all the current users.

If you do specify a user name, the system displays the following information:

Login nam

Full nam

Terminal name and write status (an * indicates that write status is denied)

Idle tim

Login tim

Office location, office phone, and home phone (if known

Contents of certain files in the **login** directory.

The **user** entry is the name of the user on the remote system.

The **@system_name** entry is the name of a remote system.

Requesting User Information

To see a list of current users on a remote system:

1. Type **finger @system_name** after the **$** _ on the command line.

   For example, to see a list of current users on remote system **chelsea**, type:

   ```
   finger @chelsea
   ```

2. Press the **Enter** key.

To see information about a specific user on a remote system:

1. Type **finger user@system_name** after the **$** _ on the command line.

   For example, to see information about a user named **jones** on remote system **chelsea**, type:

   ```
   finger jones@chelsea
   ```

2. Press the **Enter** key.

9.6 Requesting Information about Remote Systems (ping)

To determine the status of the network and various remote systems, use the **ping** command.

The **ping** command has the following basic format:

ping [**options**] **system_name**

The **ping** command determines the status of the network and other systems.

The **system_name** entry is the name of remote system.

ping offers options of interest to system programmers (See the *AIX Operating System TCP/IP User's Guide* for details). However, its main use to users is to establish whether a given system is available for TCP/IP communication. If you are trying to establish a connection with a user on some other system and not getting the results you expect, you can use **ping** as a quick test of whether that machine is up or down.

+--- **Determining Whether another System is Available** ----------+
¦ |
¦ 1. Issue the **ping** command, followed by the name of the system whose ¦
¦ TCP/IP connection you wish to test. ¦
¦ |
¦ ping oolong ¦
¦ |
¦ 2. A response such as the following is displayed: ¦
¦ |
¦ PING oolong: 56 data bytes ¦
¦ 64 bytes from 130.200.22.1: icmp_seq=0. time=40. ms ¦
¦ |
¦ ----oolong.acme.com PING Statistics ---- ¦
¦ 1 packets transmitted, 1 packets received, 0% packet loss ¦
¦ round-trip (ms) min/avg/max = 40/40/40 ¦
¦ |
¦ 3. The fourth line tells you that a packet was sent to a system ¦
¦ called *oolong* and a packet was returned. This means that *oolong* ¦
¦ is available for communication over the TCP/IP connection. If ¦
¦ *oolong* had been unavailable, no packet would have been returned ¦
¦ and 100% loss would be displayed. ¦
¦ |
+---+

9.7 Transferring Files (ftp)

You can transfer files between two AIX systems or between an AIX and a remote system with the **ftp** command. This transfer operation includes the following steps:

1. The **ftp** command makes a connection to the other system.

2. Once the connection is made, you issue subcommands that instruct the system to transfer the file.

   See "Subcommands for ftp" in topic 9.7.1 for information about these subcommands.

The **ftp** command has the following basic format:

ftp system_name

The **ftp** command makes a connection to another system so files can be transferred. This is the interface to the File Transfer Protocol (FTP).

The **system_name** entry is the name of the system you want to reach. This may be a PS/2 or another type of system to which you have a connection. A system is sometimes called the host computer.

When you see the **ftp>** prompt, give the subcommands that you need to make the file transfer.

+--- **Making the ftp Connection** ----------+

1. Enter **ftp system_name** after the **$** _ on the command line.

   If you want to reach a system named **host2**, enter:

   ```
   ftp host2
   ```

   When the connection is made, you are notified and prompted for a login name: **Name (host2:local username)**

2. To log in to the remote system with your local system name, press the **Enter** key.

   For example, if you used **smith** on the local system, press the **Enter** key when you see: **Name (host2:smith)**

   To log in to the remote system with a different user name, type the name after the displayed information and press the **Enter** key. To log in as **sam**, add the name as shown:

   ```
   Name (host2:smith) sam
   ```

3. If prompted, type a valid password, then press the **Enter** key. (The password does not appear on the screen.) The prompt for the previous example is:

```
Password (host2:sam)
```

4. The prompt changes to **ftp>**. You now can enter any **ftp** subcommand. See the list of subcommands and the steps for transferring files that follow.

Subtopics
9.7.1 Subcommands for ftp

9.7.1 Subcommands for ftp

The following set of **ftp** subcommands lets you transfer files and perform related tasks, such as changing the type of file transfer, displaying information, and changing directory and file names. For a complete list of subcommands, refer to the *AIX Operating System TCP/IP User's Guide*.

! (exclamation mark)
Invokes a shell on the local system so you can give a shell command.

append localfile [remotefile]
Appends a local file to a file on the remote system. If no remote file name is given, the local file name is used.

ascii Sets the file-transfer type to support network ASCII. This is the default. File transfer may be more efficient with binary-image. Refer to the **binary** subcommand below.

binary Sets the file-transfer type to support binary-image transfer. This can be more efficient than an ASCII transfer, the default.

cd directoryname
Changes the working directory on the remote system to the directory you name.

delete filename Deletes the file you name on the remote system.

dir [directoryname]
Displays the contents of the directory you name on the remote system. If you omit a directory name, **ftp** displays the contents of the current directory.

get remotefile [localfile]
Retrieves a file from the remote system and stores it on your local system. If no local file name is given, the remote file name is used.

help [subcommand]
Displays a help message about the subcommand you name. If you do not specify a subcommand, **ftp** displays a list of known subcommands.

lcd [directoryname]
Changes the working directory on your system to the directory you name. If you do not specify a directory, **ftp** uses your **login** ($HOME) directory.

ls [directoryname]
Displays an abbreviated list of the contents of the directory on the remote system. If you do not specify a remote directory (sometimes called a foreign directory), the contents of the current directory are listed.

mget (remotefiles)
Retrieves multiple files from the remote system and stores them on your local system. The remote file names are used.

mput (localfiles)

Sends multiple files from your local system to the remote system. The local file names are used.

put localfile [remotefile]
Stores a local file on the other system. If no remote file name is given, the local file name is used.

pwd Displays the name of the current directory on the remote system.

quit Ends the File Transfer Protocol session and exits the **ftp** program.

rename source destination
Changes a file name on the remote system.

+--- **Using ftp Subcommands** ----------+

When the **ftp>** prompt appears, you are logged in to the remote system and can transfer files or do other tasks related to file transfer.

1. Type the subcommand for file transfer or a related task, adding any required file or path name. Then press the **Enter** key.

2. Continue entering subcommands until all the work is finished.

   One possible sequence is given in the next quick reference box.

3. To exit **ftp**, enter the **quit** subcommand.

```
quit
```

+----------+

Before you transfer any files, you may want to issue other subcommands. The example in the quick reference box that follows shows a typical sequence that changes the type of file transfer from the default (ASCII) and puts a file into a specific directory on the remote system.

+--- **Sample Sequence for File Transfer** ----------+

1. Change the file transfer type to binary by entering:

```
binary
```

2. Check to see which working directory you are in on the remote system by entering the **pwd** command.

```
pwd
```

3. Change the remote working directory by entering the **cd** subcommand. For example, to change to the **projects** directory, enter the

following:

```
cd projects
```

4. Transfer one file from the local to the remote system by entering the **put** subcommand. For example, to transfer **/blue** to **/u/sam/blue**, you would enter the following:

```
put /blue /u/sam/blue
```

File names must follow the conventions used by the **sh** command.

For more information about transferring files, consult *AIX Operating System TCP/IP User's Guide*.

9.8 Using a Remote Login (telnet, tn, tn3270)

You can log in to another system with which you have a connection using the **telnet** command. The **telnet** command implements the Telnet protocol, which opens a connection to the system.

Using a remote login consists of using the following:

1. The **telnet** command, which logs in to a remote system.

2. Subcommands, which you enter to manage the remote session.

Logging In To and Using a Remote System

1. Type **telnet system_name** after the prompt on the command line. For example, if you want a connection to **syst2**, enter:

   ```
   telnet syst2
   ```

   If you do not give a system name, you get the **telnet** prompt.

   ```
   telnet>
   ```

   At this point, you should use the **telnet** subcommand **open** to establish a connection with a remote system. For example, to establish a connection to **syst2**, enter:

   ```
   open syst2
   ```

   If you do not specify a system name, the system prompts you with **(to)**. After **(to)**, enter the name of the system with which you want to connect. For example:

   ```
   telnet>open
   (to)   syst2
   ```

   When the command is accepted, several lines of message text appear on the display, ending with the login prompt.

2. When prompted, enter your login name. You will then be prompted for your password. Enter your password.

   When the system prompt appears, you are logged in to the remote system and can transmit text or do related tasks, using the **telnet** subcommands.

3. If you want to receive status or help information, type the escape character ^T (**Ctrl-T**), after which you should be able to type any **telnet** subcommand. For example:

```
telnet>help
```

4. To close the connection and exit the program, you can either type **logout** from the system prompt, or you can type the escape character ^T (**Ctrl-T**) and then type the **telnet** subcommand **quit** from the **telnet** prompt.

For a complete listing of the **telnet** subcommands, refer to *AIX Operating System TCP/IP User's Guide*.

10.0 Chapter 10. Using Asynchronous Terminal Emulation (ATE)

Subtopics

10.1 CONTENTS

10.2 About This Chapter

This chapter explains the procedures you follow to communicate between a Personal System/2 and a remote computer system using Asynchronous Terminal Emulation (ATE).

The ATE facility must be configured and customized for your particular site before you can use it for remote communication. Included in this chapter is an overview of the prerequisite tasks so you can check to be sure the system is ready to use.

Note: Although customization and configuration are prerequisites for using ATE, some tasks, like giving port commands, require superuser authority and others require some data processing experience. Refer to *Managing the AIX Operating System* for a detailed discussion about customizing ATE and for information about configuring ports.

To install ATE, use the ATE diskette that comes with the AIX Operating System. For more information, see *AIX PS/2 Installation and Customization Guide* or *AIX/370 Installation and Customization Guide,* depending on the computer system you are using.

10.3 Overview of Asynchronous Terminal Emulation (ATE) Tasks

Before you can use ATE, you must learn the basic ATE tasks and understand the commands on the main menus that perform these tasks.

You perform the tasks described in this chapter to establish a connection and log in to a terminal on another system (such as a data base service). You can then give commands, use files, and run programs on that system. Figure 10-1 illustrates the basic setup of the ATE type of communication, in which the PS/2 emulates a remote terminal.

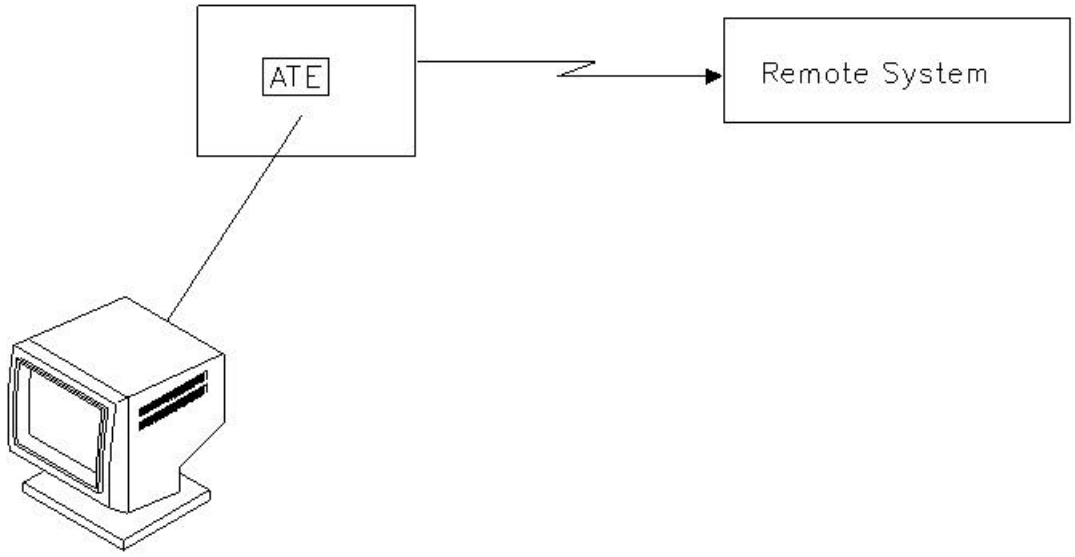


Figure 10-1. Basic Setup of ATE Communication

From the Unconnected Main Menu which appears when you start the ATE program, you can establish a connection to another terminal in two different ways:

- Make a direct connection
- Use a dialing directory and make a telephone connection

You can also check to be sure the settings for your local system and for the connection are correct and change the settings for the current session, if necessary. Tasks related to establishing a connection and changing settings are described in "Unconnected Main Menu" in topic 10.3.1.

Once a connection is made, you can display the Connected Main Menu and select commands to perform the following operations:

- Send a file

Receive a file

Interrupt a session with a break signal

Terminate a connection

These tasks are described in "Connected Main Menu" in topic 10.3.2.

As you do the above tasks, you may need to perform one of the following operations:

Get help

Enter an operating system command

Leave (quit) ATE

These last three tasks start from commands found on both of the main menus.

The following pages provide an overview of the main menus and discuss how to use menus and control keys.

Subtopics
10.3.1 Unconnected Main Menu
10.3.2 Connected Main Menu

10.3.1 Unconnected Main Menu

The Unconnected Main Menu, shown in Figure 10-2, displays any time you use the **ate** command, provided that Asynchronous Terminal Emulation is installed.

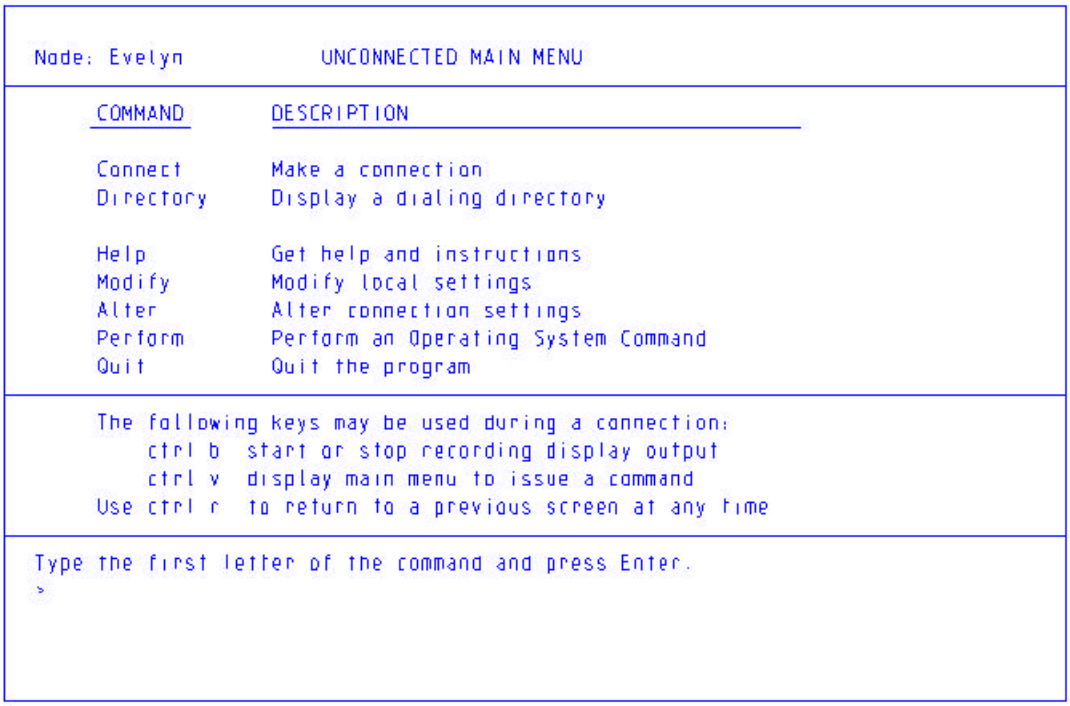


Figure 10-2. Unconnected Main Menu

With the Unconnected Main Menu you can perform the following tasks:

- Make an asynchronous connection to a remote computer
- Display and use a phone directory to make a connection
- Request help information
- Request the Modify Menu to change variables that pertain to your loca terminal (for the current session).
- Request the Alter Menu to change variables that affect dat transmission (for the current session).
- Run an operating system command (some, like port commands, requir superuser authority).
- Stop using Asynchronous Terminal Emulation

To view the Connected Main Menu, give the **connect** command from the Unconnected Main Menu. Then use the **Ctrl-V** control key.

10.3.2 Connected Main Menu

The Connected Main Menu, shown in Figure 10-3, contains the commands you need to transmit files.

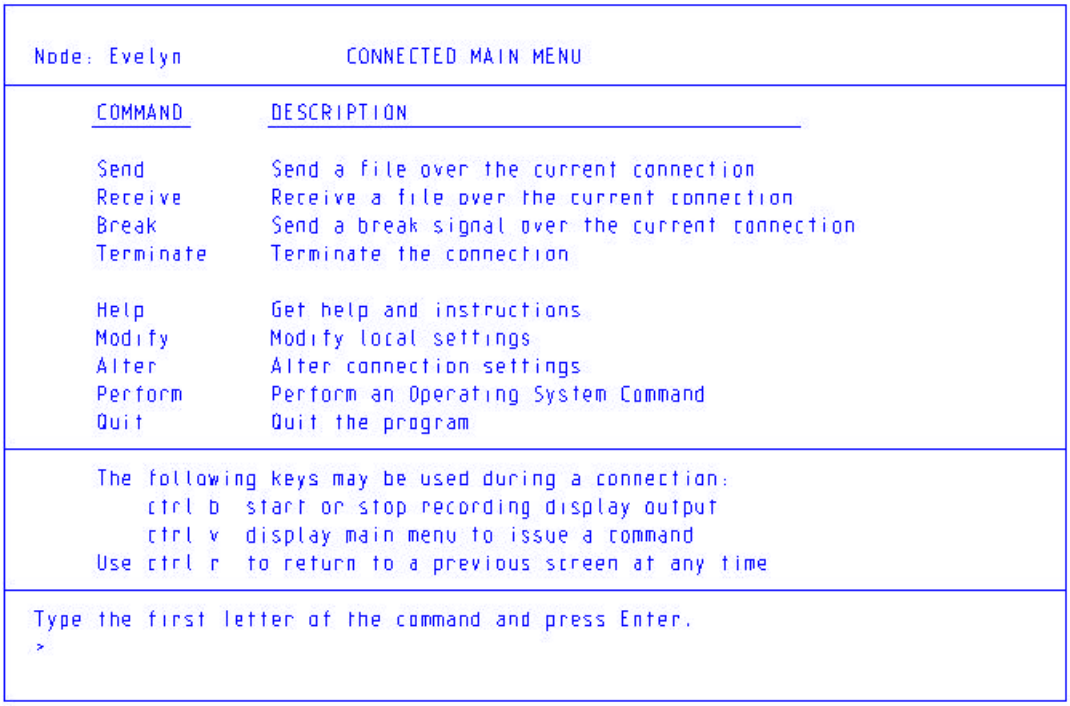


Figure 10-3. Connected Main Menu

With the Connected Main Menu you can perform these tasks:

Send a file to a remote system

Receive a file from a remote system

Interrupt a connection with a break signal

Terminate a connection

Request help information

Request the Modify Menu to change variables that pertain to your loca terminal (for the current session).

Request the Alter Menu to change variables that affect dat transmission (for the current session).

Run an operating system command (some, like port commands, requir superuser authority).

Stop using Asynchronous Terminal Emulation

10.4 Giving Commands

To give a command, type the first letter of the command after the > (prompt) at the bottom of the menu and press **Enter**. Notice that the menu prompt is a different symbol than the shell prompt, $.

For example, to display a dialing directory, type the first letter of the **directory** command and press **Enter**:

```
d
```

Do **not** try to enter values by using cursor movement keys to mark the command in the menu. The system interprets the cursor movement keys as typed input and fails to recognize the command. If you use cursor movement keys, the following error message appears:

```
062-003  The 'command-name' command is not valid.
         Enter the first letter of a command from
         the list on the menu.
```

If you see the above message, press **Enter** to redisplay the menu. Then try the command again.

Subtopics
10.4.1 Using Control Keys

10.4.1 Using Control Keys

Three control keys appear on both the Unconnected and the Connected Main Menus. These control keys do not function until a connection is made to a remote system by giving the **connect** command.

To use a control-key combination, hold the **Ctrl** key while you press the named letter key.

The functions of the control keys follow:

Ctrl-B Starts or stops saving data displayed on your screen during a connection. This control key has a switch or toggle effect. The first time you use **Ctrl-B** you save data. The next time you use it, you stop saving data. This key combination does not function while you are transferring files.

Data is saved in the default file, unless you request a specific file. Changing the default file is discussed in *Managing the AIX Operating System*.

If you press **Ctrl-B** when there is no connection, you receive an error message.

```
062-003  The 'command-name' command is not valid.
         Enter the first letter of a command from
         the list on the menu.
```

Since control characters do not display, the command name in the message will be blank.

Ctrl-V Displays the Connected Main Menu so you can issue a command. Use this control key to display the Connected Main Menu after the **connect** command makes a connection.

If you press **Ctrl-V** from the Unconnected Main Menu and then press **Enter**, you receive error message 062-003 (above).

Ctrl-R This sequence returns you to the previously displayed screen. The actual screen you see depends on the screen in use when you pressed the key combination.

Warning: You can use **Ctrl-R** to stop a file transfer in progress.

Subtopics
10.4.1.1 Using Ctrl-R from the Unconnected State
10.4.1.2 Using Ctrl-R from the Connected State

10.4.1.1 Using Ctrl-R from the Unconnected State

When you are not connected, using the **Ctrl-R** sequence produces the following results:

Where You Are:	**When You Use Ctrl-R:**
Unconnected Main Menu	Ignored.
Directory Menu	Unconnected Main Menu returns.
Help Menu	Unconnected Main Menu returns.
Modify Menu	Unconnected Main Menu returns.
Alter Menu	Unconnected Main Menu returns.
During a perform	Displays the message: Press any key.
During a quit	Ignored.
Making a connection	Terminates the connection. The menu you were using returns: Unconnected Main Menu or the Directory Menu.

10.4.1.2 Using Ctrl-R from the Connected State

When you are connected, using the **Ctrl-R** sequence produces the following results:

Where You Are:	**When You Use Ctrl-R:**
Connection Screen	Displays the message: **Disconnect (y/n)?** If you choose **yes**, either the Unconnected Main Menu or the Directory Menu appears, depending on which one you used to make the connection. If you choose **no**, the connection screen returns.
Connected Main Menu	Connection screen returns.
Help Menu	Connection screen returns.
Modify Menu	Connection screen returns.
Alter Menu	Connection screen returns.
During a perform	Displays the message: **Press any key**
During a quit	Ignored.
During a terminate	Ignored.
During a break	Ignored.
During a send	Connection screen returns.
During a receive	Connection screen returns.

10.5 Prerequisite Tasks

Before you use the ATE program, you should check the settings of the local system, the connection, and the ports. You must know the following:

Characteristics of the remote por

Characteristics of the local por

Settings for your local syste

Settings needed to connect to a remote system

At the remote system, as a minimum you must know the transmission rate, the file transfer protocol, the bit length (per character), the parity, and which remote port is ready to receive calls. Someone at the remote system can check these and other characteristics for you so you can make your local port compatible.

At the local port, as a minimum you must be sure that the transmission rate, the file transfer protocol, the parity, and the bit length are the same as those of the remote system, and know which local call-out port is configured for the task you need.

You can check and change the settings of your local system for the current session by using the **modify** command. These settings include the following:

Naming the file that captures incoming dat

Adding linefeeds at the end of each line of incoming dat

Setting echo mod

Emulating a DEC VT10

Writing incoming data to a capture fil

Establishing a transmitter on/off signal (Xon/Xoff)

When you leave ATE, the settings for the current session disappear, and the values in the default file are in effect the next time you start ATE.

See *Managing the AIX Operating System* for more information about modifying local settings.

You can check and change the connection settings for the current session by using the **alter** command. These settings include the following:

Bit length per characte

Number of stop bit

Parit

Transmission rat

Device name of por

Modem dialing prefi

```
Modem dialing suffi

Wait before redialin

Number of redial attempt

File transfer protoco

Pacing character or integer
```

When you leave ATE, the settings for the current session disappear, and the values in the default file are in effect.

See *Managing the AIX Operating System* for more information about the **alter** command.

If a port is ready to receive calls, it is ***enabled***. If a port is ready to call out, it is ***disabled***. You can change the port configuration with the **devices** command. Remember that changing the port configuration requires superuser authority.

See *Managing the AIX Operating System* for more information about using ports.

10.6 Starting ATE

To start the ATE program, type **ate** after the **$** _ and press **Enter**:

```
$ate
```

The **ate** command displays the first of two main menus, the Unconnected Main Menu, from which you establish a connection to another terminal. See "Unconnected Main Menu" in topic 10.3.1.

10.7 Making a Connection (ATE connect)

Before you begin, review the list of prerequisite tasks and make sure that the configuration is complete. Remember that checking and changing the port configuration requires superuser authority and that if you leave ATE after you change local settings or connection settings, the changes disappear.

The ATE **connect** command makes a connection between your terminal and a remote terminal with the method you select:

Manual dialin

Automatic dialin

Direct connection

The **connect** command is given from the Unconnected Main Menu, which appears when you start the program.

Subtopics
10.7.1 Manual Dialing
10.7.2 Automatic Dialing
10.7.3 Direct Connection

10.7.1 Manual Dialing

Manual dialing means that you dial the number yourself over a telephone line.

Dialing Manually

1. Be sure all the prerequisite tasks are complete. See "Prerequisite Tasks" in topic 10.5.

2. Invoke ATE by entering the following on the command line and pressing the **Enter** key.

   ```
   ate
   ```

   (Remember, entering a command means that you type the command, then press **Enter**. In this case, you should type **ate**, then press **Enter**.)

3. If you do not need to change the current settings, skip this step. To change settings for the current session, use the **modify** or the **alter** command when the Unconnected Main Menu appears.

4. To make the connection, enter the letter **c**.

   ```
   > c
   ```

5. When the following prompt appears, press **Enter** to signal that you want to dial manually:

   ```
   Type the phone number of the connection for
   auto dialing, or the name of the port for direct
   connect, and press Enter.  To manually dial a
   number, just press Enter.  To redial the last
   number (0), type 'r' and press Enter.
   >
   ```

6. Dial the telephone number when message 062-007 appears.

For information about changing local settings, see *Managing the AIX Operating System*.

The prompt for dialing the telephone number appears as messag 062-007. You can read more about this message in *AIX Operating System Messages*.

```
062-007  Please dial a telephone number to make the
         requested connection.
```

If you hear a busy signal and want to end the connection request press **Ctrl-R**.

No connection is established if the line is busy or if the part doesn't answer or if you used an unrecognized number. The following message appears after 45 seconds:

```
062-009  The 'connect' command cannot complete because
         the line was busy, or the modem did not detect
         a carrier signal.  Make sure the number is
         correct and try again, or try the same
         number later.
```

Check the number and try again.

Message 062-009 also appears if you use the **Ctrl-R** control keys. This stops a connection from being established. If you pressed **Ctrl-R** by mistake, try again.

10.7.2 Automatic Dialing

Automatic dialing means that you use a modem to dial a number over a telephone line.

Dialing Automatically

1. Be sure all the prerequisite tasks are complete. See "Prerequisite Tasks" in topic 10.5.

2. Invoke ATE by entering the following on the command line:

   ```
   ate
   ```

3. If you do not need to change the current settings, skip this step.

   When the Unconnected Main Menu appears, use the **modify** or the **alter** command to change settings for the current session.

   See *Managing the AIX Operating System* for information about changing local settings.

4. To make the connection, enter **c telephonenumber** and [**portname**] on the command line.

   ```
   > c 5551111 tty6
   ```

Your modem instruction book should tell you how to format the number perhaps by including spaces or dashes.

Although the example above contains a port name, the [ ] (brackets) i the instruction indicate that a port name is optional.

Refer to *Managing the AIX Operating System* for information about port names.

If you give the ATE **connect** command without a telephone number or port name, the following prompt appears:

Type the phone number of the connection for auto dialing, or the name of the port for direct connect, and press Enter. To manually dial a number, just press Enter. To redial the last number (0), type 'r' and press Enter. >

Type the telephone number to dial automatically:

> 5551111

Press **Enter**.

10.7.3 Direct Connection

You can make a direct connection if you have a link to another system.

Making a Direct Connection

1. Be sure all the prerequisite tasks are complete. See "Prerequisite Tasks" in topic 10.5.

2. Invoke ATE by typing the following on the command line and pressing the **Enter** key:

   ```
   ate
   ```

3. If you do not need to change the current settings, skip this step.

   When the Unconnected Main Menu appears, use the **modify** or the **alter** command to change settings for the current session.

   See *Managing the AIX Operating System* for information about changing local settings.

4. To make the connection, type **c portname** on the command line.

   ```
   > c tty6
   ```

5. Press **Enter** twice.

   The login prompt appears.

Before you can make a connection to another computer, you must disabl your port so that no one else can log in to your system while you are trying to make a connection. To do this, someone with superuser authority must use the **pdisable** command.

If you do not disable your system login, an error message appears. You must also be certain that the port of the other computer is ready to receive calls.

If you give the ATE **connect** command without typing the port name, the following prompt appears:

```
Type the phone number of the connection for
auto dialing, or the name of the port for
direct connect, and press Enter.  To manually
dial a number, just press Enter.  To redial the
last number (0), type 'r' and press Enter.
>
```

Type the port name on the command line and press **Enter**.

10.8 Displaying a Dialing Directory (directory)

The **directory** command displays a dialing directory. You can establish a connection to a remote computer by selecting one of the directory entries from the displayed directory. The information in this section tells you how to do this.

If you need to create a dialing directory, refer to "Creating a Dialing Directory File" in topic 10.9.

The **directory** command must be given from the Unconnected Main Directory.

Displaying a Dialing Directory

1. Type **d filename**

   ```
   > d bulletin_bds
   ```

2. Press **Enter**.

Subtopics
10.8.1 Additional Information about the directory Command
10.8.2 Selecting a Telephone Number from a Directory

10.8.1 Additional Information about the directory Command

If you gave the **directory** command without typing the **filename**, the following prompt appears:

```
Type the file name of the directory you want
to display and press Enter.  To use the current
directory (/usr/lib/dir), just press Enter.
>
```

If you see this prompt, type the name of the directory you want to display, unless the file name is inside the **( )**. Press **Enter**.

To use the current directory (the one displayed in parentheses), just press **Enter**.

Asynchronous Terminal Emulation comes with a sample dialing director called **/usr/lib/dir**. The first time you use the **directory** command the sample directory appears. After that, the last directory you used becomes the current directory.

10.8.2 Selecting a Telephone Number from a Directory

After displaying a directory, you select the entry to which you want to establish a connection by responding to a prompt. Figure 10-4 shows a dialing directory.

```
#        NAME            TELEPHONE (first digits)  RATE  LEN STOP PAR ECHO LF's

 0 CompuAid              555-0000                  1200   7    1   2    0    0
 1 Stock_Info            555-1111                  1200   7    1   2    0    0
 2 Info_Index            555-2222                  1200   7    1   2    0    0
 3 Electronic_Mail       555-3333                  1200   8    1   0    0    1
 4 The_Origin            555-4444                  1200   7    1   2    0    0
 6 John's_Extension      8,1111                    1200   8    1   0    0    0
 7 LD_Info               111,555-1212              1200   8    1   0    0    0
 8 Low_Cost_LD           555-5555,666666,800-555-77 1200  8    1   0    0    0
 9 Joe's_Pizza           9,555-8888                1200   8    1   0    0    0
10 bulletin_board        555-9999                   300   8    1   0    0    0
11 The_Lab               8,2222                    9600   8    1   0    0    0

 Enter directory number or e (Exit)
 >
```

Figure 10-4. Dialing Directory

1. Type the number of the entry you want (0 through 19).
2. Press **Enter**.

10.9 Creating a Dialing Directory File

This section tells you how to create a dialing directory to use with the **directory** command. For information on how to use the directory that you create, see "Displaying a Dialing Directory (directory)" in topic 10.8.

A dialing directory file is similar to a page in a telephone book. Each file contains telephone numbers for remote systems you can call with Asynchronous Terminal Emulation.

Subtopics
10.9.1 Sample Dialing Directories
10.9.2 Dialing Directory File Format

10.9.1 Sample Dialing Directories

To help you create dialing directory files, Asynchronous Terminal Emulation comes with a sample dialing directory **/usr/lib/dir**. See "Dialing Directory File Format" in topic 10.9.2.

Creating a Directory from the Model

1. Select a unique name for your dialing directory file. This name must be a string of 40 characters or less. The file can be in any directory that has write permission.
2. Copy the sample directory to your new directory file, using the **cp** command.
3. Using your favorite text editor, replace the sample entries with those you want in your directory, completing the information in each of the eight fields. (See "Dialing Directory File Format" in topic 10.9.2.)
4. Save the new file.

If you need to give a directory write permission, use the **chmod** command.

See *AIX Operating System Commands Reference* for instructions on using the **cp** and the **chmod** commands.

Limit the file to 20 entries, numbered from 0 through 19. The AT program only uses the first 20 entries in a dialing directory; if you add more than 20 entries, it ignores the extra entries.

If you are using the INed editor, refer to *AIX PS/2 INed* for help in creating files.

An alternate way to create a new directory follows:

Creating a Directory without the Model

1. Create a new file with any text editor.
2. Enter each telephone number on a separate line.

   Each entry must include the eight fields described in the next section, with at least one space between each field. Do not exceed the maximum number of characters in each field.

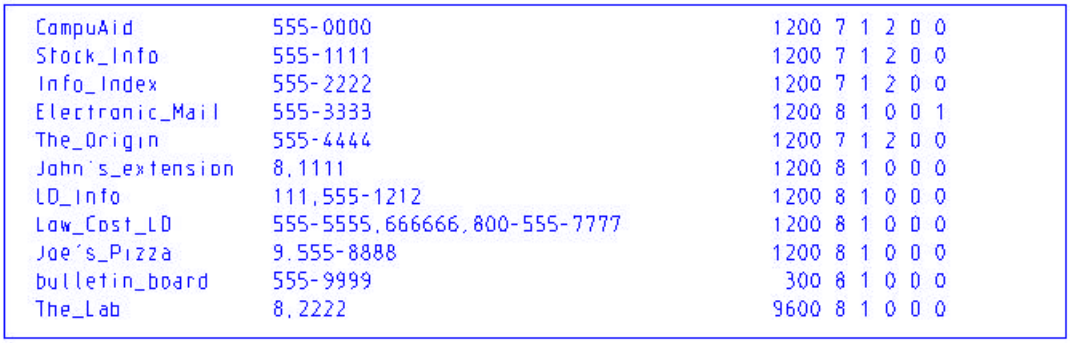
   Two sample entries follow:

```
CompuAid          555-0000 1200 7 1 2 0 0
Stock_Info       555-1111 1200 7 1 2 0 0
```

3. Save the file.

10.9.2 Dialing Directory File Format

Figure 10-5 shows the sample directory you copy to use as a model.



```
CompuAid            555-0000                          1200 7 1 2 0 0
Stock_Info          555-1111                          1200 7 1 2 0 0
Info_Index          555-2222                          1200 7 1 2 0 0
Electronic_Mail     555-3333                          1200 8 1 0 0 1
The_Origin          555-4444                          1200 7 1 2 0 0
John's_extension    8,1111                            1200 8 1 0 0 0
LD_Info             111,555-1212                      1200 8 1 0 0 0
Low_Cost_LD         555-5555,666666,800-555-7777      1200 8 1 0 0 0
Joe's_Pizza         9,555-8888                        1200 8 1 0 0 0
bulletin_board      555-9999                           300 8 1 0 0 0
The_Lab             8,2222                            9600 8 1 0 0 0
```

Figure 10-5. Sample Dialing Directory

A description of each field follows:

name Name that identifies a telephone number.

Options: Any combination of 20 characters or less. Use the _ (underscore) instead of a blank between words in a name. An example is **data_bank**.

number Telephone number to be dialed.

Options: A telephone number of 40 digits or less.

Consult your modem user's guide for a list of acceptable digits and characters. For example, if you must dial 9 to access an outside line, include a 9 and a , (comma) before the telephone number.

9,5551111

Only the first 26 characters (including digits, commas, and dashes) display in the Directory Menu.

rate Transmission or bit rate (bits per second). Determines the

number of characters transmitted per second. Select a bit rate that is compatible with the capability of the communication line you are using.

Options: 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19200.

length Number of bits that make up a character.

Options: 7 or 8.

stop Stop bits that signal the end of a character.

Options: 1 or 2.

parity Checks whether a character was successfully transmitted to or from a remote system.

For example, if you select even parity, when the number of 1 bits in the character is odd, the parity bit is turned on to make an even number of 1 bits.

Options: 0=none, 1=odd, 2=even.

echo Determines whether typed characters display locally.

Options: 0=off (no echo), 1=on (echo).

linefeed Adds a linefeed character at the end of each line of data coming in from a remote system.

The linefeed character is similar in function to the carriage return and newline characters.

Options: 0=off (no linefeed), 1=on.

10.10 Sending a File (send)

The **send** command on the Connected Main Menu sends a file from your local system to a remote system, if a connection is established. Before you can send the file, you must prepare the remote system to receive it, as follows:

Preparing a Remote System to Receive a File

To send a file to another system using **xmodem** protocol:

1. Make sure that your local system is disabled and the remote system is enabled. Someone with superuser authority must do this.

2. Make a connection to the other system by entering the **connect** command:

   ```
   c devicename
   ```

3. When the login prompt appears, enter:

   ```
   loginname
   ```

4. When the password prompt appears, enter the password (if any). If there is no password, just press **Enter**.

5. Give the **xmodem** command on the other system by entering:

   ```
   xmodem -r filename
   ```

After you have set up the remote computer to receive the file, you must perform the following steps to send the file to the remote system.

Sending a File to a Remote System

1. Press **Ctrl-V** to display the Connected Main Menu.

2. Enter **s filename** on the command line, using the name of the file you want to send.

   For example, to send a file named **myfile**, enter:

   ```
   > s myfile
   ```

If you give the **send** command without typing the **filename**, the following prompt appears:

```
Type the name of the file you wish to send
and press Enter.  To use the last file
name (    ), just press Enter.
>
```

If you see this prompt, type the name of the file you want to send, unless the file name is inside the **( )**. Press **Enter**.

10.11 Receiving a File (receive)

The **receive** command stores incoming data from a remote system in the file that you designate. You must enter this command from the Connected Main Menu.

Before you can receive the file, you must send it from the remote system.

Sending a File from a Remote System (xmodem)

To send a file from another PS/2 using the **xmodem** protocol:

1. Make sure that your local port is disabled and that the PS/2 remote port is enabled. A superuser must do this.
2. Make a connection to the other PS/2 by entering the **connect** command:

   ```
   c devicename
   ```

3. When the login prompt appears, enter:

   ```
   loginname
   ```

4. When the password prompt appears, enter the password (if any). If there is no password, just press **Enter**.
5. Give the **xmodem** command on the other system by entering:

   ```
   xmodem -s filename
   ```

After you have sent the file from the remote system, you perform the following steps to receive it:

Receiving a File from a Remote System

1. Press **Ctrl-V** to display the Connected Main Menu.
2. Enter **r filename** on the command line.

   For example, to store data in a file called **infile**, enter:

   ```
   > r infile
   ```

If you give the **receive** command without typing the **filename**, the following prompt appears:

```
Type the name of the file you wish to store
the received data in and press Enter.  To
use the last file name (    ), just press Enter.
>
```

If you see this prompt, type the name of the file in which you want to store the incoming data, unless the file name is inside the **( )**. Press **Enter**.

To use the file name displayed in parentheses, just press **Enter**.

10.12 Interrupting a Session (break)

The **break** command sends a break signal to the remote system to which your terminal is connected. The break signal interrupts current activity on the remote computer.

Warning: This signal may disconnect the current session. You may lose data.

This command must be given from the Connected Main Menu.

Interrupting a Session

1. Type **b** on the command line.

   ```
   > b
   ```

2. Press **Enter**.

10.13 Terminating a Session (terminate)

The **terminate** command stops a **connect** session and returns you to the Unconnected Main Menu. This command must be given from the Connected Main Menu.

Ending a Session

1. Type **t** on the command line.

   ```
   > t
   ```

2. Press **Enter**.

10.14 Getting Help (help)

The **help** command displays a description of each command for which you request help, and provides instructions for using the command.

You may request **help** from either the Unconnected Main Menu or the Connected Main Menu for all the commands that appear on that menu. Help may be requested for several commands at the same time.

To access the general help screen, type **h** and press **Enter**.

Viewing a Help Message

1. Type **h** [**commandinitial**] on the command line.

   For example, to receive help on the **receive** and **terminate** commands, type:

   ```
   > h r t
   ```

2. Press **Enter**.

Help information for each command displays individually, in the orde requested. As you finish reading each help message, press **Enter** to view the next command description.

The brackets [ ] indicate that you may type as many command initial as you wish. Following is a list of the valid command initials.

Initial	Command
c	connect
d	directory
s	send
r	receive
b	break
t	terminate
m	modify
a	alter
p	perform
q	quit

10.15 Running an Operating System Command (perform)

The **perform** command lets you run an AIX shell command from Asynchronous Terminal Emulation.

You can give the **perform** command from either the Unconnected Main Menu or the Connected Main Menu.

Running a Shell Command from ATE

1. Type **p shellcommand**

   For example, to display the data in **cheerfile** using the **cat** shell command, type:

   ```
   > p cat cheerfile
   ```

2. Press **Enter**.

If you give the **perform** command without typing the name of the shell command, the following prompt appears:

```
Type an operating system command and press Enter.
>
```

If you see this prompt, type the name of the command you want to run. Press **Enter**.

Note: For information on the **cat** command, see *AIX Operating System Commands Reference*.

10.16 Leaving ATE (quit)

The **quit** command exits the Asynchronous Terminal Emulation program and returns you to the operating system.

Leaving the ATE Program

1. Type **q** on the command line.

   ```
   > q
   ```

2. Press **Enter**.

The system prompt appears, indicating that you may give any shell command.

```
$ _
```

A.0 Appendix A. Using Advanced Bourne Shell Features

Subtopics

A.1 CONTENTS

A.2 Before You Begin

This appendix explains the advanced features of the Bourne shell. Many of the features covered in this appendix can be used either on the command line (to affect how an individual command runs) or in shell procedures (programs).

Please note two important differences between this appendix and the previous chapters:

> This appendix is not for the novice computer user. Unless you hav experience with computer programming, you probably should skip this material until you are thoroughly familiar with both the AIX Operating System and the earlier chapters of this book.
>
> This appendix is more like reference material, less like trainin material. It is organized according to the features of the shell, not according to the jobs you do with those features.

If you work through this appendix from beginning to end, you should have a general understanding of what you can do with the shell. However, this appendix probably will be most useful when you use it as reference material--when you have a unique job to do and need to understand what shell features can help you do it.

A.3 Shell Variables

Like variables in other programming languages, ***shell variables*** are names to which you can assign values. For example, you can assign the value **U. S. A.** to the variable **place** with the following statement:

```
$ place='U. S. A.'
$ _
```

From then on, you can use the variable **place** just as you would use its value. To display the value of variable, type a **$** (dollar sign) before the name of the variable. In the following example, the **echo** command displays the value of **place**:

```
$ echo $place
U. S. A.
$ _
```

The shell provides two kinds of variables:

- ***User-defined variables*** (names to which you assign a ***character string***--one or more characters--as a value). Generally, user-defined variables can be set on the command line or in a shell procedure.

- ***Positional parameters*** (variables in shell procedures that refer to values on the command line).

Subtopics
A.3.1 User-Defined Variables
A.3.2 Positional Parameters

A.3.1 User-Defined Variables

A ***user-defined variable*** is a name to which you assign a specific string value (one or more characters). The name consists of any number of characters chosen from the set of 52 ASCII letters, 10 digits, underscore, and the extended characters. The name cannot begin with a digit. To create a user-defined variable, use an assignment statement of the form **name=value**. There must be no spaces on either side of the equal sign.

Note: For Japanese locale users, the names given to shell variables and the equal sign used to give them a value must be expressed in ASCII characters. The string of characters which represent the value of the variable, however, can be any character set you want ASCII, Double-wide Roman, Kana or Kanji. If the value is a file name, the content of that file can be Japanese text.

Then, to use the value of the variable, type a **$** before the name of the variable. You can use user-defined variables both on the command line and in shell procedures. A special type of user-defined variable, ***keyword arguments***, can be used only on the command line. (Keyword arguments are explained under "Keyword Arguments" in topic A.3.1.5.)

In the following example, the statement **s=stringofletters** creates the variable **s**:

```
$ s=stringofletters
$ echo $s
stringofletters
$ echo s
s
$ _
```

The command **echo $s** returns the value of **s**. The command **echo s** simply returns the character **s** because a **$** does not precede the variable name **s**.

One convenient use for shell variables is as a short notation for long path names. For example, if you routinely use files in the directory **$HOME/personal/correspond/from**, you can assign that path name to the variable name **from**:

```
$ from=$HOME/personal/correspond/from
$ _
```

With this value for the variable **from**, you can use **$from** instead of entering the path name, for example:

```
$ cp tom_5_10_85 $from
$ _
```

Subtopics
A.3.1.1 Multiple Assignments
A.3.1.2 Making Commands Conditional--The ¦¦ and && Operators
A.3.1.3 Using Braces as Delimiters
A.3.1.4 Quoting in Variable Assignments
A.3.1.5 Keyword Arguments

A.3.1.1 Multiple Assignments

For convenience, you can make more than one variable assignment on a single command line.

```
$ t=text  d=data
$ _
```

It is important to remember, however, that the shell assigns values to variables in right-to-left order. This becomes significant when the variables are interrelated as in the following example:

```
$ a=$b  b=abc
$ _
```

In this example, **b=abc** is evaluated first and **b** is given the value **abc**. Then **a=$b** is evaluated. This gives **a** whatever value has been assigned to **b**. The value of **a** is also set to **abc**.

Suppose, however, that you had not done the above. Suppose that no values had yet been assigned to either **a** or **b**, and you entered the assignments in the reverse order:

```
$ b=abc  a=$b
```

This assignment gives **a** the value of a null string. The value of **b** is **abc**, exactly as you would expect. But the expression **a=$b** is evaluated first, before **b** has any assigned value. The result is that **a** is set to a null string.

A.3.1.2 Making Commands Conditional--The ¦¦ and && Operators

When you connect commands with the **¦¦** or **&&** operators, the shell runs the first command and then runs the remaining commands only under the following conditions:

¦¦ The shell runs the next command if the current command does not complete (that is, if the command fails [returns a nonzero value]).

&& The shell runs the next command if the current command completes (that is, the command succeeds [returns a value of zero]).

In the following example, the shell checks the exit status of **cmd1**:

```
$ cmd1 ¦¦ cmd2
$ _
```

If **cmd1** fails, the shell runs **cmd2**. (If **cmd1** succeeds, the shell abandons the command line and prompts you for another command.)

In the next example, the shell again checks the exit status of **cmd1**:

```
$ cmd1 && cmd2 && cmd3 && cmd4 && cmd5
```

If **cmd1** succeeds, the shell runs **cmd2**. If **cmd2** succeeds, the shell runs **cmd3**, and so on through the series until a command fails or the last command ends. If any command on the command line fails, the shell abandons the command line and prompts you for another command.)

A.3.1.3 Using Braces as Delimiters

Use braces **{ }** to separate the name of a variable from any characters that follow immediately. If the character immediately following the variable name is a letter, underscore, or digit, braces are required. In the following example, the purpose of the **echo** command is to display the two strings **fun** and **ction** without a space between them.

```
$ a='Form follows fun'
$ echo "${a}ction"
Form follows function
_
```

The braces indicate that the enclosed **a** is a variable name and that its value should be followed immediately by the string **ction**. (Compare this use of braces with the use described under "Using { } (Braces)" in topic 4.4.3.2.)

A.3.1.4 Quoting in Variable Assignments

Certain characters (summarized under "Shell Reserved Characters and Words" in topic A.10) have special meanings to the shell. In variable assignments, you may need to use these characters literally--that is, without their special meanings. To use a special character literally, you must ***quote*** it (using the same quoting conventions described under "Quoting" in topic 4.4.4). The shell takes literally all characters enclosed in single quotes (') except for the single quote itself. Within double quotes, shell takes blanks, tabs, semicolons, and new-lines literally, but substitutes the values for variable names.

Note: In variable assignments, you do not need to quote the special pattern-matching characters (*** ? [...]**), because pattern-matching does not apply in this context.

Single Quotes in Variable Assignments: In the following example, the value assigned to the variable **stuff** is enclosed in single quotes:

```
$ stuff='echo $ ?  $ *; ls * ¦  wc'
$ _
```

The value of the variable **stuff** is the literal string **echo $? $ *; ls * ¦ wc**. The shell does not run any of the commands (**echo, ls,** or **wc**) and does not give the reserved characters (**$? * ¦**) their special meanings.

Double Quotes in Variable Assignments: Within double quotes (**"**), the reserved characters **$**, ` (grave accent), and **** keep their special meanings. The shell takes literally all other characters within the double quotes.

In the following example, the first line of assignments gives values to the variables **h**, **o**, and **c**, and the second line gives a value to the variable **e**:

```
$ h=hydrogen  o=oxygen  c=carbon
$ e="Three elements:  $h; $o; $c"
$ echo $e
Three elements:  hydrogen; oxygen; carbon
$ _
```

Because the value of **e** is enclosed in double quotes, the shell takes literally the blanks and semicolons in the string. At the same time, the **$** keeps its special meaning, and causes the shell to substitute the values of the variables **h**, **o**, and **c**. Thus, the command **echo $e** returns the value of **e** with the substituted values of **h**, **o**, and **c**.

To quote the **$**, `, or **\** characters within double quotes, place a **\** (backslash) immediately before the character. See "Command Substitution" in topic A.4.2 for an explanation of grave accents (`) and how they work in quoted strings.

Variable Substitution and Quoted Blanks: Ordinarily, the shell interprets blanks between words on a command line as delimiters (separators) between a command and its arguments (and between the arguments themselves). However, after the shell substitutes the value of a variable, it still takes quoted blanks literally (that is, blanks are not reinterpreted as delimiters, even though the string is no longer enclosed in quotes). For example, the following lines assign the same value to **$first** and **$second**:

```
$ first='a string with embedded blanks'
$ second=$first
$ _
```

Compare the assignment in this example with the assignment **first=a string with embedded blanks** (the same string without the quotes). The shell would read **first=a** as a keyword argument (see "Keyword Arguments" in topic A.3.1.5), **string** as the command name, and **with**, **embedded**, and **blanks** as arguments to **string**.

A.3.1.5 Keyword Arguments

The variables that affect a command are called the ***environment*** of the command. The environment can include variables called ***keyword arguments***--variables you set on the command line when you enter the command. In the following example, the keyword argument assigns the value **fred** to the variable name **user** and that assignment becomes part of the environment of the **echo** command:

```
$ user=fred echo $user
fred
$ _
```

Note: The variable assignment in this example affects only the environment of the command, not the environment of the shell from which the command is run.

Keyword arguments usually precede the command name. However, if you set the **-k** flag, the shell takes all arguments of the form **variable=value** as keyword arguments:

```
$ set -k usr=fred command black=green tree=frog
$ _
```

(For more information about using keyword arguments, see "The export Command" in topic A.4.3. For more information about shell flags, see "Shell Flags" in topic A.9.)

A.3.2 Positional Parameters

A ***positional parameter*** is a name that refers to a string in a particular position on the command line. The positional parameter **$0** refers to the first string on the command line (usually the name of a command), **$1** refers to the second string (the first argument to the command), and so on. When you run a command, the shell creates a positional parameter for each string on the command line up to **$9**.

Positional parameters allow a shell procedure to take information from the command line (for example, to assign a value to a variable each time the procedure runs). For example, in the following procedure, the **echo** command displays the second argument on the command line (the value of positional parameter $3):

Note: The line that begins with the # character is a comment, not a functional part of the procedure.

```
# posparm3 procedure--demonstrate how pos. parameters work
echo $3
```

Note: To follow this example on your system, first use an editor to create the **posparm3** file and then give the file execute status with the **chmod** command (**chmod +x posparm3**).

You can enter **posparm3** with more than two arguments (character strings). The procedure returns only the value of positional parameter $3. In the following example, the **posparm3** procedure has five arguments:

```
$ posparm3 Bob Jones Dept. 546 Accounting
Dept.
$ _
```

Each time you run this procedure, you can use different arguments.

The shell automatically creates positional parameters for arguments in positions up to $9. To use arguments in positions numbered higher than 9, you can use the **$*** notation described under "The shift Command" in topic A.4.4, or a **for** loop, described in "for--Looping over a List" in topic A.6.4.

A.4 How the Shell Uses Variables

"Shell Variables" in topic A.3 describes the different types of shell variables. This section explains, first, how the shell ordinarily uses variables and, second, how you can control the way the shell treats variables.

Subtopics

A.4.1 Parameter Substitution

As explained under "User-Defined Variables" in topic A.3.1, the shell substitutes the value of a variable (or parameter) for the name of the variable when you type a **$** before the name. In the following example, **echo** returns the value of variable **p**:

```
$ p=45ABA54
$ echo $p
45ABA54
$ _
```

If a variable is not set (does not have a value assigned), the shell ordinarily substitutes the null string for the name of the variable. (For example, if **q** does not have a value, then **echo $q** returns nothing to the display.) However, with the notation **${variable-string}**, you can cause the shell to return a string (rather than the null) when a variable does not have a value to substitute. In the following example, variable **q** does not have a value:

```
$ echo ${q-x}
x
$ _
```

Since **q** does not have a value, **echo** returns **x** (the default string).

If you use any special characters in a default string, quote them with the usual shell quoting conventions. In the following example, **echo** returns ***** if variable **q** is not set:

```
$ echo ${q-'*'}
*
$ _
```

You also can use the value of another variable as a default string. For example, the following **echo** command returns the value of **$1** if **q** is not set:

```
$ echo ${q-$1}
$ _
```

If **$1** is not set, the shell returns the null string.

The notation **${variable=string}** actually assigns a value to a variable that is not set. In the following example, if **q** is not set, the shell assigns it the value **12B1**:

```
$ echo ${q=12B1}
$ _
```

Note: The **${variable=string}** notation does not work for positional parameters.

If an appropriate default string does not exist, you can use the following notation:

```
$ echo ${q?message}
12B1
$ _
```

The shell substitutes the value of **q** if it has one. If **q** does not have a value, the shell returns the word **message** and ends the procedure. If a procedure can run only with certain parameters set, you can use the **:** command to determine whether those parameters have values. The following line begins a procedure that must have three variables set before it can run:

```
: ${user?} ${acct?} ${bin?}
```

The **:** command does nothing once the shell evaluates its parameters. If any of its parameters (**user**, **acct**, and **bin** in this example) are not set, the **:** command causes the shell to abandon the procedure.

A.4.2 Command Substitution

When you enclose a command in ` ` (grave accents), the shell replaces the name of the command with the standard output of that command. For example, if the current directory is **/u/fred/bin**, then the following assignments are equivalent:

```
d=`pwd`
d=/u/fred/bin
```

Within ` ` (grave accents), the shell quoting conventions explained under "Quoting" in topic 4.4.4 apply, with one exception: you must use the \ (backslash) to quote a ` (grave accent).

When ` ` (grave accents) appear in strings that are enclosed in double quotes (**" . . . ` command name ` . . ."**), they are not quoted, as is explained under "Double Quotes in Variable Assignments" in topic A.3.1.4). That is, the shell reads them as command substitution reserved characters. In the following example, double quotes cause the shell to take literally the blanks and **?** character in the value assigned to the variable **where**:

```
$ where="Where am I?  `pwd`"
$ echo $where
Where am I?  /usr/fred/bin
$ _
```

The shell substitutes the standard output of **pwd** (the current directory) for **`pwd`**.

Within double quotes, a command or reserved character enclosed in ` ` (grave accents) retains its special meaning, even though it is part of a quoted string. When grave accents appear in strings enclosed in single quotes (**' . . . ` command name ` . . . '**), the shell takes literally the grave accents and any commands or reserved characters they enclose. (For more information on using single quotes, see "Single Quotes in Variable Assignments" in topic A.3.1.4.)

A.4.3 The export Command

With the **export** command, you can set user-defined variables to apply to any subsequent commands (rather than setting the keyword argument for each command when you enter it). This process, called ***marking the arguments for export***, is useful if, for example, you want the same user-defined variables to be part of the environment for several different commands. In the following example, after values are assigned to variable names **user** and **box**, the **export** command marks the variables **user** and **box** for export:

```
$ user=jason box=square
$ export user box
$ echo $user $box
jason square
$ _
```

The subsequent **echo** command displays the values of **user** and **box**. To get a list of the variables currently marked for export, enter **export**.

If you want the value of a variable to remain constant, declare it ***readonly***. In the following example, the variables **user** and **box** are declared **readonly**:

```
$ readonly user box
$ _
```

After this declaration, the values of **user** and **box** cannot be changed by a subsequent variable assignment.

To get a list of all variables declared **readonly**, simply enter **readonly**. Once you declare a variable readonly, you can change its value only by deleting the variable and then re-creating it. To delete the variable, use the shell **unset** command (described under **sh** in *AIX Operating System Commands Reference*); if you do not delete the variable, it is automatically deleted when you log out.

A.4.4 The shift Command

The **shift** command, used with positional parameters, shifts arguments to the left one string at a time. For example, the **shift** command discards the value of positional parameter $1, replaces $1 with $2, replaces $2 with $3, and so on. The **shift** command does not shift the $0 positional parameter (the name of the procedure). With each shift, the positional parameter with the highest number becomes ***unset*** (that is, there is no longer an argument in that position). An argument can cause the **shift** command to shift more than one string at a time (for example, **shift 2** causes **shift** to move two strings at a time).

To demonstrate the **shift** command, the following procedure uses three shell features that have not yet been introduced in this book. The **while** statement means **as long as a specified condition exists.** The **test** command determines whether or not a condition exists (in the example, the command **test $# != 0** means **test for a total count of positional parameters that is not equal to 0**. The control command pair **do** and **done** create a ***loop***--a series of commands to be repeated until a specified condition changes; in the example, the commands are **echo**, which displays the values of all the positional parameters, and **shift**, which shifts the positional parameters to the left one at a time. Thus, the procedure displays the values of the positional parameters, shifts the arguments to the left, displays their values again, and so on, continuing until only the $0 positional parameter (the one that refers to **echo**) remains. (The line that begins with the # character is a comment, not a functional part of the procedure.)

Note: To follow this example on your system, first use an editor to create the file **ripple** and then give the file execute status with the **chmod** command (for example, **chmod +x ripple**).

```
#          ripple command
while test $# != 0
do
           echo $1  $2  $3  $4  $5  $6  $7  $8  $9
           shift
done
```

To run the **ripple** command, enter **ripple string1 string2 string3 ...**:

```
$ ripple 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
4 5 6 7 8 9
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
$ _
```

At most, the **echo** command in **ripple** displays the values of nine arguments.

To pass more than nine arguments from the command line to a procedure, use the **$*** notation. For example, in the **ripple** procedure, the **echo** command could be written **echo $***:

```
#          ripple command
while test $# != 0
```

```
do
            echo $*
            shift
done
```

If there are nine or fewer arguments, these two versions of the **echo** command produce the same result. However, if there are more than nine arguments, only the command **echo $*** accesses all of them.

You can also use a **for** loop to pass additional arguments to the shell. See "for--Looping over a List" in topic A.6.4 for more information on using a **for** loop.

A.4.5 The set Command

The shell automatically assigns positional parameters from command line arguments. However, you can assign values to positional parameters from within shell procedures with the **set** command.

In the following example, the **set** command is one line from a shell procedure:

```
set   abc  def  ghi
```

This **set** command first makes these assignments:

```
$1   =   abc
$2   =   def
$3   =   ghi
```

Second, it unsets (clears) the remaining positional parameters (from $4 on), even if they were set before (for example, if there were arguments to the invoking command). The **set** command cannot assign a value to $0, which always refers to the name of the shell procedure.

A.4.6 The read Command

Like the ` (grave accents) used in command substitution, the **read** command lets you assign values to variables indirectly. The **read** command takes a line from its standard input (usually the keyboard) and assigns words from that line, one by one, to named variables. In the following example, the **read** command assigns words to three named variables, **first**, **init**, and **last**:

```
read first init last
```

With the input line **B. T. Andover**, the **read** command produces the same results as would the following variable assignments:

```
first=B. init=T. last=Andover
```

If there are excess words on the input line (that is, if there are more words than there are variables), they are assigned to the last variable.

A.5 Special Shell Variables

The shell program uses several special variables. The shell sets some of these variables, and you can set or reset all of them. When you use shell variables, you must precede them with a dollar sign ($). Following is a partial list of the special variables and a brief description of how the shell uses each one. (For a complete list of the shell variables, see **sh** in *AIX Operating System Commands Reference*.)

MAIL The path name of the file where your mail is deposited. To have the shell notify you when new mail arrives, you must set **MAIL**, and this is usually done in the file **.profile** in your login directory. (When you login, you receive an announcement of any mail in your standard mail file, whether **MAIL** is or is not set.)

HOME The name of your login directory. If you use the **cd** (change directory) command without arguments, **cd** changes the current directory to the value of **HOME**. (In shell procedures, you can use **HOME** to avoid having to use full path names--something that is especially helpful if the path name of your login directory changes.) **HOME** is set by the **login** command.

PATH A list of directories that contain commands. When the shell runs a command, it searches a list of directories for a file of that name that can be run. If **PATH** is not set, the shell searches the current directory, **/bin**, and **/usr/bin**. When **PATH** is set, its value is an ordered list of directory names separated by colons, as is shown in the following example:

```
PATH=:/u/fred/bin:/bin:/usr/bin
```

This **PATH** tells the shell to search the current directory (specified by the null string before the first **:** [colon]), **/u/fred/bin**, **/bin**, and **/usr/bin**, in that order. Thus, the **PATH** variable lets you have a personal directory of commands that you can access regardless of your current directory. Usually, **PATH** is set in the **.profile** file.

LANG Sets formats for the way language-related data is entered from the keyboard, displayed on your screen and processed by AIX programs. This includes the language, the character set, and a battery of formats for date, time, numbers and currency. **LANG** selects all these items in the coordinated set of formats used by a whole national language group. You can also set the format for each of these data items individually. See "environment" in the *AIX Technical Reference* for full details.

CDPATH The variable that tells the shell where to search for the argument to a **cd** command whenever that argument is not null and does not begin with **/**, **.**, or **..**. The value of **CDPATH** is an ordered list of directory path names separated by colons.

A null character anywhere in the list represents the current directory. If the list begins with a colon, a null character is assumed to be before the colon. Initially, **CDPATH** is not set, which means that the shell searches only the current directory. In the following example, the **cd** command is set to search the current directory first, and then to search the home directory:

```
CDPATH=:$HOME
```

Usually **CDPATH** is set in your **.profile** file. If the **cd** command changes to a directory that is not a descendent of the current directory, the shell writes the full path name of the new directory on the diagnostic output.

PS1 The variable that specifies the primary prompt string--the string that the shell displays when it is ready to accept a command. The standard primary prompt string is **$** (a dollar sign followed by a blank). PS1 is usually set in the file **$HOME/.profile**. If PS1 is not set, shell uses the standard primary prompt string. To change the primary prompt string, edit **$HOME/.profile** and either change the value of PS1 (if it is set) or add the line **PS1=string** (where string is the primary prompt string you choose).

PS2 The variable that specifies the secondary prompt string--the string that the shell displays when it requires more input after a new-line (that is, after you press **Enter**). The standard secondary prompt string is **>** (a **>** symbol followed by a blank). PS2 is usually set in the file **$HOME/.profile**. If PS2 is not set, shell uses the standard secondary prompt string. To change the secondary prompt string, edit **$HOME/.profile** and either change the value of PS2 or add the line **PS2=string**.

IFS The variable that specifies what characters can be used as internal field separators (IFS); these are the characters the shell uses during blank interpretation. The shell initially sets **IFS** to include the blank, tab, and new-line characters.

? The exit status (return code) of the last command run, given as a decimal string. Most commands return a zero exit status if they complete successfully and a nonzero exit status otherwise.

The number of positional parameters, given as a decimal string.

$ The process number of the current shell, given as a decimal string. All existing processes have unique process numbers.

! The process number of the last process run in the background, given as a decimal string.

- The current shell flags.

A.6 Shell Control Commands

Often, you may want a shell procedure to do one thing under certain conditions and another thing when those conditions change (or are never met). For example, you may want part 1 of a procedure to run if the procedure receives **yes** as input, and part 2 if the procedure receives **no** as input. With input **yes**, control of the procedure goes to part 1. With input **no**, control goes to part 2.

The shell provides the following six ***control commands***, or command pairs, that let you pass control to various parts of a procedure or control how a procedure ends:

break and **continue** (loop control)

case (multiway branch)

exit and **trap** (process ending control)

for (looping over a list)

if (structured conditional branch)

while and **until** (conditional looping).

This section lists the control commands in alphabetical order.

Note: These conditional commands handle variables with Japanese character values.

Subtopics

A.6.1 break and continue--Loop Control

The **break** command ends a **while**, **until**, or **for** loop. The **continue** command starts the next instance of a loop. Both **break** and **continue** work only when they are used between **do** and **done**.

In the following example, the **case** command compares input from the keyboard (entered in response to the prompt **Please enter data**) with the three strings, **done**, **" "**, and *****. (The lines that begin with # are comments, not functional parts of the procedure.)

```
#This procedure is interactive; 'break' and 'continue'
#commands are used to allow the user to control data entry.
while true
do
        echo "Please enter data"
        read response
        case "$response" in
              "done")    break         #no more data
                         ;;
              " ")       continue
                         ;;
              *)
                         process the data here
                         ;;
        esac
done
```

If the entered data matches **done**, the **break** command ends the loop and causes the procedure to start again after **done** (the end of the enclosing loop). If the entered data matches **" "** (a space), the **continue** command causes the procedure to start again at the **while** (or **until** or **for**) that begins this enclosing loop. If the entered data matches *****, the procedure processes the data and then completes normally (**esac** ends the **case** statement).

The **break** command exits from the innermost enclosing loop and causes the procedure to start again after the next (unmatched) **done**. To restart the procedure more than one level up from the loop containing **break**, use **break n**, where **n** specifies the number of levels.

The **continue** command causes the procedure to start again at the nearest enclosing **while**, **until**, or **for** (that is, the one beginning the innermost loop containing the **continue**). To restart at any loop other than the innermost enclosing one, use **continue n**, where **n** specifies how many loops (levels) up the **continue** is to operate.

A.6.2 case--The Multiway Branch

With the **case** command, you can create multiway branches. The following example shows the general format for the **case** command:

```
case string in
   pattern) command list;;
   .
   .
   .
   pattern) command list;;
esac
```

The shell attempts to match **string** with each **pattern** in turn. When the shell finds a **pattern** that matches **string**, it runs the command list following that **pattern**. The shell matches only one pattern (that is, if more than one **pattern** in the list matches **string**, the shell runs the command list after the first matching pattern, and then ends the **case** command.

The **;;** (double semicolon) symbol causes the shell to break out of the **case** procedure. It is required after all but the last command list. You can use the standard shell pattern-matching characters with **case**. (Pattern-matching characters are described under "Matching Patterns" in topic 4.4.5.)

A.6.3 The exit and trap Commands

With the **exit** and **trap** commands, you can control the way a process ends (terminates). The **exit** command ends a process before the process reaches end-of-file. If **exit** has an argument, it sets the exit status of the process to the value of that argument. For example, the command **exit 0** in a procedure causes the exit status of that procedure to be **0** (that is, successful). If the argument is omitted, **exit** uses the exit status of the last command that ran.

Ordinarily, an interrupt signal from the keyboard ends a shell procedure. The **trap** command can be set to do any routine tasks necessary to make the termination of a process orderly. In the following example, the **trap** command is set to remove temporary files when signal 2 (interrupt from the keyboard) is received:

```
trap 'rm /tmp/ps$$; exit' 2
```

The **exit** command is required. Without the **exit**, the procedure would start again at the point where the interrupt was received.

A.6.4 for--Looping over a List

With the **for** command, you can perform an operation for each of several files, or run a command for each of several arguments. The next example shows the general format of a shell **for** command:

```
for variable in word list
do
        command list
done
```

A word list is a series of strings separated by blanks. The shell runs commands in the command list once for each word in the word list; **variable** takes each word in the word list in turn as its value.

In the following **for** command, the shell runs **wc** for each of the files **top**, **middle**, and **bottom**:

```
for counts in top middle bottom
do
    wc $counts >> countfile
done
```

Each time **wc** runs, its output is directed to a file named **countfile**.

After it is evaluated the first time, the word list is fixed. The **for** command ends when there are no more words in the word list.

You can use the **for** command without the **in word list** statement. In this case, the current positional parameters are used instead of the word list. This feature is convenient if you need to write a command that performs the same command list for an unknown number of arguments.

A.6.5 if--The Structured Conditional Branch

The **if** command can be used with or without an **else** clause. The following example includes an **else** clause:

```
if command list 1
then
      command list 2
else
      command list 3
fi
```

In this example, the shell runs **command list 1**. If the exit status of **command list 1** is zero, the shell runs **command list 2**. If the exit status of **command list 1** is not zero, the shell runs **command list 3**. The word **fi** marks the end of the **if** command.

A.6.6 The setspath Command

setspath [**LOCAL|site|cpu**]...

The **setspath** command sets the site path. Sites may be specified by name or by number. CPU types may be specified by the same names which are used in hidden directories. You may also specify the argument as LOCAL to select the local machine. For more information, refer to the **sh** command in *AIX Operating System Commands Reference*.

A.6.7 while and until--Conditional Looping

Following is an example of the general format for the **while** command:

```
while command list 1
do
        command list 2
done
```

The shell runs **command list 1**. If **command list 1** is successful the shell runs **command list 2**. The shell repeats this sequence until **command list 1** is not successful, and then ends the loop.

The **until** command causes the shell to run a loop as long as the first command list is **not** successful (that is, **until** and **while** test for opposite conditions). In the next example, the shell runs **command list 1** and then checks its exit status (successful or unsuccessful):

```
until command list 1
do
        command list2
done
```

If **command list 1** is not successful, the shell runs **command list 2**. The shell repeats this sequence until **command list 1** is successful, and then ends the loop.

A.7 Inline Input (Here) Documents

The shell commands and procedures usually read their input from the keyboard or from a file (as is explained under "Redirecting Input and Output" in topic 4.3.3). A command in a procedure also can read its input from data contained in the procedure file (***inline input***). The inline input portion of a procedure file is often called a ***here*** document.

A here document begins with the **<<** symbol and ends with a specified string, as the following example shows:

```
for i
do
        grep $i <<!
        ted   abc123
        fred  def456
        tom   ghi789
        sam   jkl198
        frank mn0765
        bill  pqr432
!
done
```

The string that follows **<<** (in this example, **!**) appears on a line by itself, it marks the end of the here document. In this example, the shell takes the data between **<<!** and **!** as the standard input for **grep**.

The shell substitutes the values of any variables or parameters in the here document before it makes the data available as input to a command. To prevent the shell from substituting the value of specific variables, quote the **$** reserved character with a ****. To prevent the shell from substituting the values of all variables in a here document, quote the end marking string (**!** in the previous example) with a **** (backslash).

A.8 Standard Error and Other Output

Generally, when a command starts, three files already are open: ***standard input***, ***standard output***, and ***standard error***. If you want to redirect standard input or standard output (for example, cause a command to take its input from a file rather than from the keyboard), you can use the procedures explained under "Redirecting Input and Output" in topic 4.3.3. However, if you want to redirect standard error (or other) output, you must use the methods explained in this section; the methods in this section also can be used to redirect standard input and standard output.

A number, called a ***file descriptor***, is associated with each of the files a command ordinarily uses:

Table A-1. Standard File Descriptors

File	File Descriptor	Device
standard input	0	keyboard
standard output	1	display
standard error	2	display

To redirect standard error output, type a **2** (the file descriptor number) before one of the output redirection symbols (> and >>) and a file name after the symbol. For example, the following command adds standard error output from the **cc** command to the file **ERRORS**:

```
$ cc testfile.c 2>>ERRORS
$ _
```

Commands may produce output besides standard and standard error. With the method just described for redirecting standard error output, you can redirect output associated with any file descriptor from 0 through 9. For example, if **cmd** writes output to file descriptor 9, you can redirect that output to the file **savedata** with the following command:

```
$ cmd 9>savedata
```

If a command produces output to several different file descriptors, you can redirect each one independently, as the following example shows:

```
$ cmd > standard     2> error     9> data
$ _
```

The command **cmd** directs standard output to file descriptor 1, standard error to file descriptor 2, and output for a data file to file descriptor 9.

AIX Operating System commands generally use only file descriptors 0, 1, and 2. For more information about using file descriptors to redirect input and output, see **sh** in *AIX Operating System Commands Reference*.

A.9 Shell Flags

The shell provides two different types of flags:

Set flags. These flags, which are put into effect by the ***set*** command, alter the way the shell runs.

Command line flags. These flags, which are entered on the command line, alter the way the shell starts. Command line flags cannot be set with the **set** command.

Note: The flags listed in this section must be entered in ASCII characters.

Subtopics
A.9.1 Set Flags
A.9.2 Command Line Flags

A.9.1 Set Flags

To put a set flag into effect, enter **set -f** (where **-f** is the name of one or more flags preceded by a hyphen). In the following example, the **set** command turns on the **x** and **v** flags.

```
set -xv
```

To remove a set flag, enter **set +flag** (where **+flag** is the name of one or more flags preceded by a plus sign):

```
set +xv
```

Following is a list of set flags that often are useful in shell procedures.

Flag	Explanation
-e	Causes the shell to exit immediately if any command exits with nonzero exit status.
-u	Causes the shell to treat use of an unset variable as an error. This flag can be used to perform a global check on variables.
-t	Causes the shell to exit after reading and running the commands on the remainder of the current input line.
-n	Prevents the shell from running commands in a procedure. For example, to check a procedure for syntax errors without running the commands, enter **set -nv** at the beginning of the file.
-k	Causes the shell to treat all arguments on the command line of the form **variable=value** as keyword arguments. When **-k** is not set, only arguments of this type that appear before the command name are treated as keyword arguments.

In addition to these set flags, there are two others--the **x** and **v** flags--that are useful for debugging shell procedures. The **x** and **v** flags are usually set from the keyboard.

Flag	Explanation
-x	Causes the shell to print commands and their arguments as they are run. The **-x** flag does not print shell control commands, such as **for**, **while**, **case**, and **if**. **Note:** The **-x** flag traces only the commands that are run, while the **-v** flag causes the shell to print each line of input until a syntax error is found.

Flag	Explanation
-v	Causes the shell to print input lines as they are read. This flag is helpful in finding syntax errors. The commands on each input line are run after that input line is printed, unless the **-n** flag is also in effect.

A.9.2 Command Line Flags

The following list contains descriptions of the four shell command line flags. These flags are specified on the command line and cannot be turned on with the **set** command.

Flag	Explanation
-i	Starts an interactive shell. (If this flag is specified or if both input and output are connected to the display station, the shell is interactive.)
-s	Causes the shell to read commands from standard input. (If this flag is specified, or if input is not redirected, the shell reads commands from standard input.) The shell output is written to file descriptor 2 (stderr). When you log in to the system, your initial shell operates as if the **-s** flag is turned on.
-c	Causes the shell to read commands from the first string following the flag. Remaining arguments are ignored. Double quotes should be used to enclose a multiword string in order to allow for variable substitution.
-r	Starts the restricted shell. In the restricted shell, certain commands are not available. For example, the **cd** command produces an error message, and you cannot set PATH. For more information on the restricted shell, see **sh** in *AIX Operating System Commands Reference*.

A.10 Shell Reserved Characters and Words

Note: All of the reserved characters and words listed in this section must be entered in ASCII characters.

Subtopics

A.10.1 Syntactic

Table A-2. Shell Reserved Characters and Words--Syntactic

|	pipe symbol
&&	AND symbol
||	OR symbol
;	command separator
;;	case delimiter
&	background commands
( )	command grouping
<	input redirection
<<	input from a here document
>	output creation
>>	output append

A.10.2 Patterns

Table A-3. Shell Reserved Characters and Words--Pattern-Matching	
*	match any character(s) including none
?	match any single character
[...]	match any of the enclosed characters

A.10.3 Substitution

Table A-4. Shell Reserved Characters and Words--Substitution	
${...}	substitute shell variable
` ... `	substitute command output

A.10.4 Quoting

Table A-5. Shell Reserved Characters and Words--Quoting	
[\]	quote the next character
'...'	quote the enclosed characters except for ' (the single quote itself)
"..."	quote the enclosed characters except for the $, `, [\], and "

A.10.5 Reserved Words

Table A-6. Shell Reserved Characters and Words--Reserved Words
if then else elif fi
case in esac
for while until do done
{ } [ ] test

B.0 Appendix B. Creating and Editing Files with ed

Subtopics

B.1 CONTENTS

B.2 About This Appendix

This appendix explains how to create, edit (modify), display, and save text files with **ed**, a line editing program. If your system has another editing program, you may wish to learn how to do these tasks with that program.

A good way to learn how **ed** works is to try the examples in this appendix on your system. Since the examples build upon each other, it is important for you to work through them in sequence. Also, to make what you see on the screen consistent with what you see in this guide, it is important to do the examples just as they are given.

In the examples, everything you should type is printed in **special characters, like this**. When you are told in the text to ***enter*** something, you should type all of the information for that line and then press the **Enter** key.

Note: A line editing program allows you to work with the contents of a file one line at a time. Regardless of what text is on the screen, you can edit only the ***current*** line. If you have experience with a screen editing program, you should pay careful attention to the differences between that program and **ed**. For example, with the **ed** program, you cannot use the **Cursor Up** and **Cursor Down** keys to change your current line.

B.3 Understanding Text Files and the Edit Buffer

A ***file*** is a collection of data stored together in the computer under an assigned name. You can think of a file as the computer equivalent of an ordinary file folder--it may contain the text of a letter, a report, or some other document, or the source code for a computer program. File names can be up to 255 characters long and can contain letters, numbers, periods, commas, underscores, and some other characters.

The ***edit buffer*** is a temporary storage area that holds a file while you work with it--the computer equivalent of the top of your desk. When you work with a text file, you place it in the edit buffer, make your changes to the file (edit it), and then transfer (copy) the contents of the buffer to a permanent storage area.

The rest of this appendix explains how to create, display, save, and edit (modify) text files.

B.4 Creating and Saving Text Files

To follow this procedure, you must be logged in to the AIX Operating System and have the **$** (shell) prompt on your screen.

To Create and Save a Text File

1. At the **$** (shell) prompt, enter:

   ed filename

   Where **filename** is the name of the file you want to create or edit.

2. When you receive the **?filename** message, enter:

   a

3. Enter your text.

4. To stop adding text, enter a **.** (period) at the start of a new line.

5. Enter:

   w

   to copy the contents of the edit buffer into the file **filename**.

6. Enter:

   q

   to end the **ed** program.

Subtopics

B.4.1 Starting the ed Program

To start the **ed** program, enter a command of the form **ed filename** after the **$** (shell) prompt. (In place of **filename**, enter the name you want to assign to the file.)

In the following example, the **ed afile** command starts the **ed** program and indicates that you want to work with a file named **afile**:

Note: If you intend to work through the examples, start with this one.

```
$ ed afile
?afile
_
```

The **ed** program responds with the message **?afile**, which means that the file does not now exist. You can now use the **a** (append) subcommand (described in the next section) to create **afile** and put text into it.

B.4.2 Entering Text--The a (append) Subcommand

To put text into your file, enter **a**. The **a** subcommand tells **ed** to add or append, the text you type to the edit buffer. Type your text, pressing **Enter** at the end of each line. When you have entered all of your text, enter a **.** (period) at the start of a new line.

Note: If you do not press **Enter** at the end of each line, the **ed** program automatically moves your cursor to the next line after you fill a line with characters. However, **ed** treats everything you type before you press **Enter** as one line, regardless of how many lines it takes up on the screen; that is, the line ***wraps around***.

The following example shows how to enter text into the file **afile**:

```
a
The only way to stop
appending is to type a
line that contains only
a period.
.
_
```

If you stop adding text to the buffer and then decide you want to add some more, enter another **a** subcommand. Type the text and then enter a period at the start of a new line to stop adding text to the buffer.

If you make errors as you type your text, you can correct them--before you press **Enter**. Use the **Backspace** key to erase the incorrect character(s). Then type the correct characters in their place.

B.4.3 Displaying Text--The p (print) Subcommand

Use the **p** (print) subcommand to display the contents of the edit buffer. To display a single line, use the subcommand **np** (where **n** is the number of the line):

```
2p
appending is to type a
_
```

To display a series of lines, use the **n,mp** subcommand (where **n** is the starting line number and **m** is the ending line number):

```
1,3p
The only way to stop
appending is to type a
line that contains only
_
```

To display everything from a specific line to the end of the buffer, use the **n,$p** subcommand (where **n** is the starting line number and **$** stands for the last line of the buffer). In the following example, **1,$p** displays everything in the buffer:

```
1,$p
The only way to stop
appending is to type a
line that contains only
a period.
_
```

Note: Many examples in the rest of this appendix use **1,$p** to display the buffer's contents. In these examples, the **1,$p** subcommand is optional, but convenient--it lets you verify that the subcommands in examples work as they should. Another convenient **ed** convention is **,p**, which is equivalent to **1,$p**--that is, it displays the contents of the buffer.

B.4.4 Saving Text--The w (write) Subcommand

The **w** (write) subcommand ***writes***, or copies, the contents of the buffer into a file. You can save all or part of a file under its original name or under a different name. In either case, **ed** replaces the original contents of the file you specify with the data copied from the buffer.

Subtopics

B.4.4.1 Saving Text Under the Same File Name

To save the contents of the buffer under the original name for the file, enter **w**:

```
w
78
_
```

The **ed** program copies the contents of the buffer into the file named **afile** and displays the number of characters copied into the file (78). This number includes blanks and characters such as **Enter** (sometimes called ***newline***) which are not visible on the screen.

The **w** subcommand does not affect the contents of the edit buffer. You can save a copy of the file and then continue to work with the contents of the buffer.

The stored file is not changed until the next time you use **w** to copy the contents of the buffer into it. As a safeguard, it is a good practice to save a file periodically while you work on it. Then, if you make changes (or mistakes) that you do not want to save, you can start over with the most recently saved version of the file.

Note: The **u** (undo) subcommand restores the buffer to the state it was in before it was last modified by an **ed** subcommand. The subcommands that **u** can reverse are: **a**, **c**, **d**, **g**, **G**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **V**, and **n**.

B.4.4.2 Saving Text Under a Different File Name

Often, you may need more than one copy of the same file. For example, you could have the original text of a letter in two files--one to keep as it is, and the other to be revised.

If you have followed the previous examples, you have a file (named **afile**) that contains the original text of your document. To create another copy of the file (while its contents are still in the buffer), use a subcommand of the form **w filename**, as the following example shows:

```
w bfile
78
_
```

At this point, **afile** and **bfile** have the same contents; since each is a copy of the same buffer contents. However, because **afile** and **bfile** are separate files, you can change the contents of one without affecting the contents of the other.

B.4.4.3 Saving Part of a File

To save part of a file, use a subcommand of the form **n,mw filename**, where:

n is the beginning line number of the part of the file you want to save.

m is the ending line number of the part of the file you want to save (or the number of a single line, if that is all you want to save).

filename is the name of a different file (optional).

In the following example, the **w** subcommand copies lines 1 and 2 from the buffer into a new file named **cfile**:

```
1,2w cfile
44
_
```

Then **ed** displays the number of characters written into **cfile** (44).

B.4.5 Leaving the ed Program--The q (quit) Subcommand

Warning: You lose the contents of the buffer when you leave the **ed** program. To save a copy of the data in the buffer, use the **w** subcommand to copy the buffer into a file before you leave the **ed** program.

To leave the **ed** program, enter the **q** (quit) subcommand:

```
q
_
```

The **q** subcommand returns you to the **$** (shell) prompt.

If you have changed the buffer, but have not saved a copy of its contents, the **q** subcommand responds with **?**, an error message. At that point, you can either save a copy of the buffer (with the **w** subcommand) or enter **q** again (which lets you leave the **ed** program without saving a copy of the buffer).

Note: You can log out from the **ed** program by pressing **END OF FILE**. If you have changed the buffer since you last saved it, the system displays the **?** error message.

B.5 Loading Files into the Edit Buffer

Before you can edit a file, you must load it into the edit buffer. You can load a file either at the time you start the **ed** program or while the program is running.

To Load Files into the Edit Buffer

ed filename

This starts **ed** and loads the file **filename** into the edit buffer.

OR

e filename

When **ed** is running, this loads the file **filename** into the buffer, erasing any previous contents of the buffer.

OR

nr filename

When **ed** is running, this reads the named file into the buffer after line **n**. (If you do not specify **n**, **ed** adds the file to the end of the buffer.)

Subtopics
B.5.1 Using the ed (edit) Command
B.5.2 Using the e (edit) Subcommand
B.5.3 Using the r (read) Subcommand

B.5.1 Using the ed (edit) Command

To load a file into the edit buffer when you start the **ed** program, simply type the name of the file after the command **ed**. The **ed** command in the following example invokes the **ed** program and loads the file **afile** into the edit buffer:

```
$ ed afile
78
_
```

The **ed** program displays the number of characters that it read into the edit buffer (**78**).

If **ed** cannot find the file, it displays **?filename**. To create that file, use the **a** (append) subcommand (described under "Entering Text--The a (append) Subcommand" in topic B.4.2) and the **w** (write) subcommand (described under "Saving Text--The w (write) Subcommand" in topic B.4.4).

B.5.2 Using the e (edit) Subcommand

Once you start the **ed** program, you can use the **e** (edit) subcommand to load a file into the buffer. The **e** subcommand replaces the contents of the buffer with the new file. (Compare the **e** subcommand with the **r** subcommand, described under "Using the r (read) Subcommand" in topic B.5.3, which adds the new file to the buffer.)

Warning: When you load a new file into the buffer, the new file replaces the buffer's previous contents. Save a copy of the buffer (with the **w** subcommand) before you read a new file into the buffer.

In the following example, the subcommand **e cfile** reads the file **cfile** into the edit buffer, replacing **afile**:

```
e cfile
44
e afile
78
_
```

The **e afile** subcommand then loads **afile** back into the buffer, deleting **cfile**. The **ed** program returns the number of characters read into the buffer after each **e** subcommand (**44** and **78**).

If **ed** cannot find the file, it returns **?filename**. To create that file, use the **a** (append) subcommand, described under "Entering Text--The a (append) Subcommand" in topic B.4.2, and the **w** (write) subcommand, described under "Saving Text--The w (write) Subcommand" in topic B.4.4.

You can edit any number of files, one at a time, without leaving the **ed** program. Use the **e** subcommand to load a file into the buffer. After making your changes to the file, use the **w** subcommand to save a copy of the revised file. (See "Saving Text--The w (write) Subcommand" in topic B.4.4 for information about the **w** subcommand.) Then use the **e** subcommand again to load another file into the buffer.

B.5.3 Using the r (read) Subcommand

Once you have started the **ed** program, you can use the **r** (read) subcommand to read a file into the buffer. The **r** subcommand adds the contents of the file to the contents of the buffer. The **r** subcommand does not delete the buffer. (Compare the **r** subcommand with the **e** subcommand, described under "Using the e (edit) Subcommand" in topic B.5.2, which deletes the buffer before it reads in another file.)

With the **r** subcommand, you can read a file into the buffer at a particular place. For example, the **4r cfile** subcommand reads the file **cfile** into the buffer following line 4. The **ed** program then renumbers all of the lines in the buffer. If you do not use a line number, the **r** subcommand adds the new file to the end of the buffer's contents.

The following example shows how to use the **r** subcommand with a line number:

```
1,$p
The only way to stop
appending is to type a
line that contains only
a period.
3 r cfile
44
1,$p
The only way to stop
appending is to type a
line that contains only
The only way to stop
appending is to type a
a period.
$ _
```

1,$p displays the four lines of **afile**. Next, the **3 r cfile** subcommand loads the contents of **cfile** into the buffer, following line 3, and shows that it read 44 characters into the buffer. The next **1,$p** subcommand displays the buffer's contents again, which lets you verify that the **r** subcommand read **cfile** into the buffer after line 3.

If You Are Working the Examples

If you are working the examples on your system, do the following before you go to the next section:

1. Save the contents of the buffer in the file **cfile**. Enter:

   w cfile

2. Load **afile** into the buffer. Enter:

   e afile

B.6 Displaying and Changing the Current Line

The **ed** program is a ***line editor***. This means that **ed** lets you work with the contents of the buffer one line at a time. The line you can work with at any given time is called the ***current line***, and it is represented by the symbol . (called **dot**). To work with different parts of a file, you must change the current line.

To Display the Current Line

To display the current line, enter:

p

OR

To display the line number of the current line, enter:

.=

Note: You cannot use the **Cursor Up** and **Cursor Down** keys to change the current line. To change the current line, use the **ed** subcommands described in the following sections.

To Change Your Position in the Buffer

To set your current line to line number **n**, enter:

n

To move the current line forward through the buffer one line at a time:

Press **Enter**

To move the current line backward through the buffer one line at a time, enter:

-

To move the current line **n** lines forward through the buffer, enter:

.+n

To move the current line **n** lines backward through the buffer, enter:

.-n

Subtopics
B.6.1 Finding Your Position in the Buffer
B.6.2 Changing Your Position in the Buffer

B.6.1 Finding Your Position in the Buffer

When you first load a file into the buffer, the last line of the file is the current line. As you work with the file, you usually change the current line many times. You can display the current line or its line number at any time.

To display the current line, enter **p**:

```
p
a period.
$ _
```

The **p** subcommand displays the current line (**a period.**). Because the current line has not been changed since you read **afile** into the buffer, the current line is the last line of the buffer.

Enter **.=** to display the line number of the current line:

```
.=
4
$ _
```

Since **afile** has four lines, and the current line is the last line in the buffer, the **.=** subcommand displays **4**.

You also can use the **$** (the symbol that stands for the last line in the buffer) with the **=** subcommand to determine the number of the last line in the buffer:

```
$=
4
$ _
```

The **$=** subcommand is an easy way to find out how many lines are in the buffer.

Note: The **ed $** symbol has no relationship to the **$** (shell) prompt.

B.6.2 Changing Your Position in the Buffer

You can change your position in the buffer (change your current line) in one of two ways:

Specify a line number (an absolute position)

Move forward or backward relative to your current line

To move the current line to a specific line, enter the line number; **ed** displays the new current line. In the following example, the first line of **afile** becomes the current line:

```
1
The only way to stop
$ _
```

To move the current line forward through the buffer one line at a time, press **Enter**, as the following example shows:

```
appending is to type a

line that contains only

a period.

?
$ _
```

Notice that when you try to move beyond the last line of the buffer, **ed** returns **?**, an error message. You cannot move beyond the end of the buffer.

To set the current line to the last line of the buffer, enter **$**.

To move the current line backward through the buffer one line at a time, enter **-** (hyphens) one after the other.

```
-
line that contains only
-
appending is to type a
-
The only way to stop
-
?
$ _
```

When you try to move beyond the first line in the buffer, you receive the **?** message. You cannot move beyond the top of the buffer.

To move the current line forward through the buffer more than one line at a time, enter **.n** (where **n** is the number of lines you want to move):

```
.2
line that contains only
_
```

To move the current line backward through the buffer more than one line at

a time, enter **.-n** (where **n** is the number of lines you want to move):

```
.-2
The only way to stop
_
```

B.7 Locating Text

If you do not know the number of the line that contains a particular word or another string of characters, you can locate the line with a ***context search***.

To Make a Context Search

To search forward, enter:

/string to find/

To search backward, enter:

?string to find?

Subtopics
B.7.1 Searching Forward through the Buffer
B.7.2 Searching Backward through the Buffer
B.7.3 Changing the Direction of a Search

B.7.1 Searching Forward through the Buffer

To search forward through the buffer, enter the string enclosed in **//** (slashes):

```
/only/
line that contains only
_
```

The context search (**/only/**) begins on the first line after the current line, and then locates and displays the next line that contains the string **only**. That line becomes the current line.

If **ed** does not find the string between the first line of the search and the last line of the buffer, then it continues the search at line 1 and searches to the current line. If **ed** searches the entire buffer without finding the string, it displays the **?** error message:

```
/random/
?
_
```

Once you have searched for a string, you can search for the same string again by entering **//**. The following example shows one search for the string **only**, and then a second search for the same string:

```
/only/
The only way to stop
//
line that contains only
_
```

B.7.2 Searching Backward through the Buffer

Searching backward through the buffer is much like searching forward, except that you enclose the string in question marks (**??**):

```
?appending?
appending is to type a
_
```

The context search begins on the first line before the current line and then locates the first line that contains the string **appending**. That line becomes the current line. If **ed** searches the entire buffer without finding the string, it stops the search at the current line and displays the message **?**.

Once you have searched backward for a string, you can search backward for the same string again by entering **??**.

B.7.3 Changing the Direction of a Search

You can change the direction of a search for a particular string by using the **/** and **?** search characters alternately:

```
/only/
line that contains only
??
The only way to stop
_
```

If you go too far while searching for a character string, it is convenient to be able to change the direction of your search.

B.8 Making Substitutions--The s (substitute) Subcommand

Use the **s** (substitute) subcommand to replace a ***character string*** (a group of one or more characters) with another. The **s** subcommand works with one or more lines at a time and is especially useful for correcting typing or spelling errors.

To Make Substitutions

To substitute **newstring** for **oldstring** at the first occurrence of **oldstring** in the current line, enter:

s/oldstring/newstring/

To substitute **newstring** for **oldstring** at the first occurrence of **oldstring** on line number **n**, enter:

ns/oldstring/newstring/

To substitute **newstring** for **oldstring** at the first occurrence of **oldstring** in each of the lines **n** through **m**, enter:

n,ms/oldstring/newstring/

Subtopics

B.8.1 Substituting on the Current Line

To make a substitution on the current line, first make sure that the line you want to change is the current line. In the following example, the **/appending/** (search) subcommand locates the line to be changed. Then the **s/appending/adding text/p** (substitute) subcommand substitutes the string **adding text** for the string **appending** on the current line. The **p** (print) subcommand displays the changed line.

```
/appending/
appending is to type a
s/appending/adding text/
p
adding text is to type a
_
```

Note: For convenience, you can add the **p** (print) subcommand to the **s** subcommand (for example, **s/appending/adding text/p**). This saves you from having to type a separate **p** subcommand to see the result of the substitution.

A simple **s** subcommand changes only the first occurrence of the string on a given line. To learn how to change all occurrences of a string on the line, see "Changing Every Occurrence of a String" in topic B.8.4.

B.8.2 Substituting on a Specific Line

To make a substitution on a specific line, use a subcommand of the form **ns/oldstring/newstring/**, where **n** is the number of the line on which the substitution is to be made. In the following example, the **s** subcommand moves to line number 1 and replaces the string **stop** with the string **quit** and displays the new line:

```
1s/stop/quit/p
The only way to quit
_
```

The **s** subcommand changes only the first occurrence of the string on a given line. To learn how to change all occurrences of a string on the line, see "Changing Every Occurrence of a String" in topic B.8.4.

B.8.3 Substituting on Multiple Lines

To make a substitution on multiple lines, use a subcommand of the form **n,ms/oldstring/newstring/**, where **n** is the first line of the group and **m** is the last. In the following example, the **s** subcommand replaces the first occurrence of the string **to** with the string **TO** on every line in the buffer.

```
1,$s/to/TO/
1,$p
The only way TO quit
adding text is TO type a
line that contains only
a period.
_
```

The **1,$p** subcommand displays the contents of the buffer, which lets you verify that the substitutions were made.

B.8.4 Changing Every Occurrence of a String

Ordinarily, the **s** (substitute) subcommand changes only the first occurrence of a string on a given line. However, the **g** (global) operator lets you change every occurrence of a string on a line or in a group of lines.

To make a global substitution on a single line, use a subcommand of the form **ns/oldstring/ newstring/g**. In the following example, **3s/on/ON/gp** changes each occurrence of the string **on** to **ON** in line 3 and displays the new line:

```
3s/on/ON/gp
line that cONtains ONly
-
```

To make a global substitution on multiple lines, specify the group of lines with a subcommand of the form **n,ms/oldstring/ newstring/g**. In the following example, **1,$s/TO/to/g** changes the string **TO** into the string **to** in every line in the buffer:

```
1,$s/TO/to/g
1,$p
The only way to quit
adding text is to type a
line that cONtains ONly
a period.
_
```

B.8.5 Removing Characters

You can use the **s** (substitute) subcommand to remove a string of characters (that is, to replace the string with "nothing"). To remove characters, use a subcommand of the form **s/oldstring//** (with no space between the last two / characters).

In the following example, **ed** removes the string **adding** from line number 2 and then displays the changed line:

```
2s/adding//p
text is to type a
_
```

B.8.6 Substituting at Line Beginnings and Ends

Two special characters let you make substitutions at the beginning or end of a line:

Special Substitution Characters

^ (circumflex)	Makes a substitution at the beginning of the line. (To get the ^ character, press **Shift-6**.)
$ (dollar sign)	Makes a substitution at the end of the line. (In this context, the **$** character does not stand for the last line in the buffer.)

To make a substitution at the beginning of a line, use the **s/^/newstring** subcommand. In the following example, one **s** subcommand adds the string **Remember,** to the start of line number 1. Another **s** subcommand adds the string **adding** to the start of line 2:

```
1s/^/Remember, /p
Remember, The only way to quit
2s/^/adding/p
adding text is to type a
_
```

To make a substitution at the end of a line, use a subcommand of the form **s/$/newstring**. In the following example, the **s** subcommand adds the string **Then press Enter.** to the end of line number 4:

```
4s/$/  Then press Enter./p
a period.  Then press Enter.
_
```

Notice that the substituted string includes two blanks before the word **Then** to separate the two sentences.

B.8.7 Using a Context Search

If you do not know the number of the line you want to change, you can locate it with a context search. See "Locating Text" in topic B.7 for more information on context searches.

For convenience, you can combine a context search and a substitution into a single subcommand: **/string to find/s/oldstring/ newstring/**.

In the following example, **ed** locates the line that contains the string **, The** and replaces that string with **, the**:

```
/, The/s/, The/, the/p
Remember, the only way to quit
_
```

Also, you can use the search string as the string to be replaced with a subcommand of the form **/string to find/s//newstring/**. In the following example, **ed** locates the line that contains the string **cONtains ONly**, replaces that string with **contains only**, and prints the changed line:

```
/cONtains ONly/s//contains only/p
line that contains only
_
```

B.9 Deleting Lines--The d (delete) Subcommand

Use the **d** (delete) subcommand to remove one or more lines from the buffer. The general form of the **d** subcommand is **starting line,ending lined**. After you delete lines, **ed** sets the current line to the first line following the lines that were deleted. If you delete the last line from the buffer, the last remaining line in the buffer becomes the current line. After a deletion, **ed** renumbers the remaining lines in the buffer.

To Delete Lines from the Buffer

To delete the current line, enter:

d

To delete line number **n** from the buffer, enter:

nd

To delete lines numbered **n** through **m** from the buffer, enter:

n,md

Subtopics
B.9.1 Deleting the Current Line
B.9.2 Deleting a Specific Line
B.9.3 Deleting Multiple Lines

B.9.1 Deleting the Current Line

If you want to delete the current line, simply enter **d**. In the following example, the **1,$p** subcommand displays the entire contents of the buffer, and the **$** subcommand makes the last line of the buffer the current line:

```
1,$p
Remember, the only way to quit
adding is to type a
line that contains only
a period.  Then press Enter.
$
a period.  Then press Enter
d
_
```

The **d** subcommand then deletes the current line (in this case, the last line in the buffer).

B.9.2 Deleting a Specific Line

If you know the number of the line you want to delete, use a subcommand of the form **nd** to make the deletion. In the following example, the **2d** subcommand deletes line 2 from the buffer:

```
2d
1,$p
Remember, the only way to quit
line that contains only
_
```

The **1,$p** subcommand displays the contents of the buffer, showing that the line was deleted.

B.9.3 Deleting Multiple Lines

To delete a group of lines from the buffer, use a subcommand of the form **n,md**, where **n** is the starting line number and **m** is the ending line number of the group to be deleted.

In the following example, the **1,2d** subcommand deletes lines 1 through 2:

```
1,2d
1,$p
?
_
```

The **1,$p** subcommand displays the **?** message, indicating that the buffer is empty.

If you are following the examples on your system, you should restore the contents of the buffer before you move on to the next section. The following example shows you how to restore the contents of the buffer:

```
e afile
?
e afile
78
_
```

This reads a copy of the original file **afile** into the buffer.

Note: You must enter **e afile** twice, as shown above, because the contents of the buffer have not been saved. The first **e afile** indicates your intent to edit **afile**. The system responds with **?**, which is an error message indicating that you have not yet written the contents of the buffer. When you reenter **e afile**, you indicate to **ed** that you understand that the current contents of the buffer will be lost.

B.10 Moving Text--The m (move) Subcommand

Use the **m** (move) subcommand to move a group of lines from one place to another in the buffer. After a move, the last line moved becomes the current line.

To Move Text

Enter a subcommand of the form **x,ymz** where:

x is the first line of the group to be moved.
y is the last line of the group to be moved.
z is the line the moved lines are to follow.

In the following example, the **1,2m4** subcommand moves the first two lines of the buffer to the position following line 4:

```
1,2m4
1,$p
line that contains only
a period.
The only way to stop
appending is to type a
_
```

The **1,$p** subcommand displays the contents of the buffer, showing that the move is complete.

To move a group of lines to the top of the buffer, use 0 as the line number for the moved lines to follow. In the next example, the **3,4m0** subcommand moves lines 3 through 4 to the top of the buffer:

```
3,4m0
1,$p
The only way to stop
appending is to type a
line that contains only
a period.
_
```

The **1,$p** subcommand displays the contents of the buffer, showing that the move has been made.

To move a group of lines to the end of the buffer, use **$** as the line number for the moved lines to follow:

```
1,2m$
1,$p
line that contains only
a period.
The only way to stop
appending is to type a
_
```

Note: Type **u** to undo this last command before continuing.

B.11 Changing Lines of Text--The c (change) Subcommand

Use the **c** (change) subcommand to replace one or more lines with one or more new lines. The **c** subcommand first deletes the line(s) you want to replace and then lets you enter the new lines, just as if you were using the **a** (append) subcommand. When you have entered all of the new text, type a **.** (period) on a line by itself. The general form of the **c** subcommand is **starting line,ending linec**.

To Change Lines of Text

1. Enter a subcommand of the form:

   n,mc

   where:

   n is the number of the first line of the group to be deleted.
   m is the number of the last line of the group (or the only line) to be deleted.

2. Type the new line(s), pressing **Enter** at the end of each line.

3. Enter a period on a line by itself.

Subtopics
B.11.1 Changing a Single Line
B.11.2 Changing Multiple Lines

B.11.1 Changing a Single Line

To change a single line of text, use only one line number with the **c** (change) subcommand. You can replace the single line with as many new lines as you like.

In the following example, the **2c** subcommand deletes line 2 from the buffer, and then you can enter new text:

```
2c
appending new material is to
use the proper keys to create a
.
1,$p
The only way to stop
appending new material is to
use the proper keys to create a
line that contains only
a period.
_
```

The period on a line by itself stops **ed** from adding text to the buffer. The **1,$p** subcommand displays the entire contents of the buffer, showing that the change has been made.

B.11.2 Changing Multiple Lines

To change more than one line of text, give the starting and ending line numbers of the group of lines to be with the **c** subcommand. You can replace the group of lines with one or more new lines.

In the following example, the **2,3c** subcommand deletes lines 2 through 3 from the buffer, and then you can enter new text:

```
2,3c
adding text is to type a
.
1,$p
The only way to stop
adding text is to type a
line that contains only
a period.
_
```

The period on a line by itself stops **ed** from adding text to the buffer. The **1,$p** subcommand displays the entire contents of the buffer, showing that the change has been made.

B.12 Inserting Text--The i (insert) Subcommand

Use the **i** (insert) subcommand to insert one or more new lines into the buffer. To locate the place in the buffer for the lines to be inserted, you can use either a line number or a context search. The **i** subcommand inserts new lines before the specified line. (Compare the **i** subcommand with the **a** subcommand, explained under "Entering Text--The a (append) Subcommand" in topic B.4.2, which inserts new lines after the specified line.)

To Insert Text

1. Enter a subcommand of one of the following types:

   ni

   where **n** is the number of the line the new lines will be inserted above.

   /string/i

   Where **string** is a group of characters contained in the line the new lines will be inserted above.

2. Enter the new lines.

3. Enter a period on a line by itself.

Subtopics
B.12.1 Using Line Numbers
B.12.2 Using a Context Search

B.12.1 Using Line Numbers

If you know the number of the line where you want to insert new lines, you can use an insert subcommand of the form **ni** (where **n** is a line number). The new lines you type go into the buffer before line number **n**. To end the **i** subcommand, enter a **.** (period) on a line by itself.

In the following example, the **1,$p** subcommand prints the contents of the buffer. Then the **4i** subcommand inserts new lines before line number 4.

```
1,$p
The only way to stop
adding text is to type a
line that contains only
a period.
4i
--repeat, only--
.
1,$p
The only way to stop
adding text is to type a
line that contains only
--repeat, only--
a period.
_
```

After **4i** you enter the new line of text and enter a period on the next line to end the **i** subcommand. A second **1,$p** subcommand displays the contents of the buffer again, showing that the new text has been inserted.

B.12.2 Using a Context Search

Another way to specify where the **i** subcommand inserts new lines is to use a context search. With a subcommand of the form **/string/i**, you can locate the line that contains **string** and insert new lines before that line. When you finish inserting new lines, enter a period on a line by itself.

In the following example, the **/period/i** subcommand inserts new text before the line that contains the string **period**:

```
/period/i
and in the first position--
.
1,$p
The only way to stop
adding text is to type a
line that contains only
--repeat, only--
and in the first position--
a period.
_
```

The **1,$p** subcommand displays the entire contents of the buffer, showing that the **i** subcommand has inserted the new text.

B.13 Copying Lines--The t (transfer) Subcommand

With the **t** (transfer) subcommand, you can copy lines from one place in the buffer and insert the copies elsewhere. The **t** subcommand does not affect the original lines. The general form of the **t** subcommand is **starting line,ending line tline to follow**.

To Copy Lines

Enter a subcommand of the form:

n,mtx

where:

n is the first line of the group to be copied.
m is the last line of the group to be copied.
x is the line the copied lines are to follow.

To copy lines to the top of the buffer, use 0 as the line number for the copied lines to follow. To copy lines to the bottom of the buffer, use **$** as the line number for the copied lines to follow.

In the following example, the **1,3t4** subcommand copies lines 1 through 3, and inserts the copies after line 4:

```
1,3t4
1,$p
The only way to stop
adding text is to type a
line that contains only
--repeat, only--
The only way to stop
adding text is to type a
line that contains only
and in the first position--
a period.
_
```

The **1,$p** subcommand displays the entire contents of the buffer, showing that **ed** has made and inserted the copies and that the original lines are not affected.

B.14 Using System Commands from ed

Sometimes you may find it convenient to use a system command without leaving the **ed** program--perhaps to use one of the commands covered in Chapter 3, "Using the File System." Use the **!** character to leave from the **ed** program temporarily.

To Use a System Command from ed

Enter:

!command name

In the following example, the **!ls** command temporarily suspends the **ed** program and runs the **ls** (list) system command (a command that lists the files in the current directory):

```
!ls
afile
bfile
cfile
!
_
```

The **ls** command displays the names of the files in the current directory (**afile**, **bfile,** and **cfile**) and then displays another **!** character. The **ls** command is finished, and you can continue to use **ed**.

You can use any system command from within the **ed** program. You can even run another **ed** program, edit a file, and then return to the original **ed** program. From the second **ed** program, you can run a third, use a system command, and so forth.

B.15 Ending the ed Program

This completes the introduction to the **ed** program. To save your file and end the **ed** program, do the steps in the following box:

Saving a File and Ending ed

1. Enter:

   w

2. Enter:

   q

For a full discussion of the **w** and **q** subcommands, see "Saving Text--The w (write) Subcommand" in topic B.4.4 and "Leaving the ed Program--The q (quit) Subcommand" in topic B.4.5, respectively.

For information about other features of **ed**, see **ed** in *AIX Operating System Commands Reference*.

For information about printing the files you create with **ed**, see Chapter 2, "Displaying and Printing Files" in topic 2.0.

B.16 Using ed to Enter Japanese Text

This section describes how to use the **ed** editor to enter text in Japanese. It shows you how to start **ed**, how to add Japanese text to a file, and how to search for Japanese character strings.

Note: In addition to **ed**, you can use the following AIX editing programs with Japanese text:

Ined editor (see *AIX PS/2 INed*)

vi (see *AIX PS/2 Text Formatting Guide*).

Subtopics
B.16.1 Starting ed
B.16.2 Entering ed Commands

B.16.1 Starting ed

There are two ways to start **ed** depending on whether the file has an ASCII name or a Japanese name. If the file name is ASCII, follow these steps:

To begin Editing a File with a Standard ASCII Name

1. Place the keyboard in ASCII mode and type:

   ed filename

   where **filename** is any ASCII string.

2. Press **Enter**.

If the file name is Japanese, follow these steps:

To begin Editing a File with a Japanese Name

1. Place the keyboard in ASCII mode and type:

   ed

2. Place the keyboard in the entry mode for the desired form of Japanese text and type:

   filename

   where **filename** is any string of Kana or Kanji characters.

3. Press **Enter**.

After the program starts, all the subcommands must be entered in ASCII characters. This includes the **a** (append), **p** (print), and **q** (quit) commands. All of the non-letter characters that **ed** uses in its command structure must also be entered in ASCII characters. These include the slash (/), question mark (?), and period (.).

B.16.2 Entering ed Commands

Every time you want to enter an **ed** subcommand, you must switch the keyboard into ASCII mode and type that command plus any of the symbols required for the command structure. After you do this, you can enter Japanese text. Your use of the **ed** program requires that you shift back and forth from ASCII to Japanese text entry modes. Japanese and ASCII entries to **ed** can be combined in two ways:

Sometimes your Japanese text is typed in after you pressed **Enter** and that subcommand begins to execute.

Sometimes the Japanese text is part of the command you are enterin (before you press **Enter** to execute the subcommand).

Following are examples of these two combinations.

Subtopics

B.16.2.1 Adding Text to a File

The **a** (append) command is an example of an **ed** command that requires you to enter text after the command executes.

To Add Japanese Text

1. Put the keyboard in ASCII mode and type:

   a

2. Press **Enter**.

You are immediately placed in input mode and the program is ready to accept whatever text you give it. Place the keyboard in any of the Japanese text entry modes and enter whatever characters you wish. Any mixture of characters is permissible. Press **Enter** at the end of every line.

Other **ed** commands that behave like the **a** command are: **p** (print), **w** (write), **q** (quit), **e** (edit), **r** (read), **s** (substitute), and **m** (move) commands.

To get out of input mode, do the following:

Exiting from the Input Mode

1. Press **Enter** once to put the cursor on a line by itself.
2. Put the keyboard back in ASCII entry mode.
3. Type a single period (.) by itself on a line.
4. Press **Enter**.

You are immediately back in command mode. The **ed** program is once again waiting for you to enter another subcommand.

B.16.2.2 Searching for Japanese Characters in a File

The Context Search facility is an example of a command that requires you to enter text as part of the command.

To Search for a String of Japanese Characters

1. Place the keyboard in ASCII entry mode.
2. Type a single slash (**/**) at the beginning of the line.
3. Switch to one of the Japanese entry modes.
4. Type the string of characters you want to search for.
5. Put the keyboard back in ASCII mode.
6. Type another slash (**/**) at the end of the string.
7. Press **Enter**.

The **ed** program displays the line that contains the string you requested.

B.16.2.3 Things to Consider

The **ed** editor is a simple and basic text editor but it is the foundation on which the more powerful UNIX editors (**ex**, **vi**, and **emacs**) are built. Time spent on learning **ed** thoroughly is time well invested. Using text editors is what most computer users spend most of their time doing and all the UNIX editors work in a way similar to **ed**.

Therefore, it is suggested that you review the exercises given in this appendix and practice each, entering various types of Japanese characters. Practice each function several times until you are sure you understand how that subcommand responds to entries made in ASCII, Romaji, Hiragana, Katakana, and Kanji. Experiment with each function, trying different combinations.

Be careful to avoid the common pitfall of entering doublewidth Roman characters when ASCII characters are needed. They look very similar on the screen, but the system responds very differently to them. This is surprisingly easy to do, especially when making small entries.

A doublewidth Roman period, for example, does not allow you to exit from input mode. Only the ASCII period allows you to do this. Similar caution is necessary for the slash, the question mark, the equal sign, and the numerals. Only ASCII characters perform the associated functions.

If the command you are trying does not give you the expected response, try again and pay special attention to the display width indicator at the bottom of the screen.

Finally, in order to edit a file created by another user, you must be operating in the same locale as was used when the file was created. This means that your shell must be set to evaluate the file according to the same language and file code as those used to create that file. The concept of locale is more fully discussed in Appendix C, "Working in a Japanese Locale."

C.0 Appendix C. Working in a Japanese Locale

This appendix contains information for those users who are working in a Japanese locale. It explains the concept of locale and shows you how to determine which locale you are set to and how to change locales. This appendix also contains information about the layout of the PS/55 keyboard and describes the special function keys for text entry in Japanese. You will learn how to start and stop **jaixterm**, the program that runs under x-server and enables you to enter text in Japanese, and how to switch into the five text entry modes available for entering Japanese text. The last section of this appendix gives you rules for using Japanese characters and describes techniques you can use to create an environment where all of your work can be done in Japanese.

Subtopics

C.1 Locales

Locale dictates which language your system recognizes and which character sets it can input and display. The locale causes messages seen by the user to appear in a selected dialect of a specified language. It also causes things like currency and date formats to be printed as is customary in a specific language and country. Your AIX system is certain to have at least two locales available: English and Japanese. These two languages are encoded according to the following schemes:

pc850 The file code used for English and other Latin-based languages.

pc932 One of several file codes available for Japanese.

There may be other locales available on your system. Check with your system administrator.

Subtopics
C.1.1 Changing Locales: Hardware Considerations
C.1.2 Determining Which Locale Is Set
C.1.3 Changing Your Locale
C.1.4 Matching Locales

C.1.1 Changing Locales: Hardware Considerations

Most users work in a single locale all of the time and do not need to be concerned with changing locales to do their work. Other users, for example those who work in an international environment, may need to change locales frequently.

Whether the locale can be changed depends on the terminal. An ASCII terminal, for example, cannot be used in a Japanese locale, because neither the keyboard nor the screen supports Japanese characters. Similarly, the IBM 5550 supports only the ASCII and Shift-JIS character sets. The 5550 can generate Roman characters but not the extended Roman characters used by some European languages. The 5550 is unlikely to be used for entering anything other than English or Japanese characters. Therefore, the 5550 is always used in an English or Japanese locale, and a user cannot change to a European locale. The PS/55, on the other hand, is often used as a cluster host. The PS/55 displays a full array of character sets, providing that the appropriate fonts are installed. Therefore, the PS/55 used as a cluster host can be changed to different locales by a user.

Workstations that are members of a cluster are either hosts in the cluster or terminals connected to a port by a serial line. The system administrator places an entry in the **/etc/ports** file that specifies a locale and character set for each user. The terminal comes up in that locale, and the login prompt is presented in that character set. The user logs in with ASCII characters from that character set. Japanese language users who interact with the system through a PS/55 have the option of changing their locale; users who interact through a 5550 must remain in the locales set for their terminals by the system administrator.

Workstations that are not part of a cluster are handled differently. They connect over the LAN and do not connect to the same port every time. Unassigned ports are set to the language in which all or most of the users operate: an English locale in an English cluster or a Japanese locale in a Japanese cluster. Terminals that connect over the LAN come up in the appropriate locale. Users of PS/55s can change locale; users of 5550s or ASCII terminals cannot.

C.1.2 Determining Which Locale Is Set

When you first bring up the PS/55, it starts in a mode that recognizes ASCII characters no matter which locale is in effect. You can then log on to the AIX system by entering your login name and password in ASCII characters.

After AIX is running, you can check to see which locale is set. You do this by issuing this command:

```
echo $LANG
```

If your locale is set to Japanese, the system displays one of the following responses:

```
Jp_JP.pc932
japanese
Jp
```

Note: For any session in which you enter Japanese characters, you must be sure that your locale is set to Japanese.

If your locale is set to US English, the system displays one of the following:

```
us
En_US
En_US.pc850
```

If you need to change the locale in which you are operating, see "Changing Your Locale" in topic C.1.3.

C.1.3 Changing Your Locale

Note: Whether the locale can be changed depends on the terminal. See "Changing Locales: Hardware Considerations" in topic C.1.1 for details.

Locale is changed by resetting the values of the **LANG** environment variable. This is normally done during terminal startup by entries in the **.profile** or **login** file, but it can be done from the command line after startup. The commands you enter depend on whether you are using the C shell or the Bourne shell.

Note: The prompt displayed by the shell is often reset to something more informative than the **%** or **$**. If this has been done on your system, you may not be able to tell which shell you are using. Ask your system administrator which shell is set up for you.

From the C Shell: The standard C shell prompt is a percent sign (**%**). If your system displays this prompt, you are using the C shell and you should enter the following command to set your locale to US English:

```
setenv LANG En_US.pc850
```

To set your locale to Japanese, enter:

```
setenv LANG Jp_JP.pc932
```

From the Bourne Shell: The standard Bourne Shell prompt is a dollar sign (**$**). If your system displays this prompt, you are using the Bourne Shell and you should enter these commands to set your locale to US English:

```
LANG=En_US.pc850
export LANG
```

To set your locale to Japanese, enter:

```
LANG=JP_JP.pc932
export LANG
```

C.1.4 Matching Locales

To edit a file created by another user, you must be operating in the same locale that was used when the file was created. This means that your shell must be set to evaluate the file according to the same language and file code as those used to create that file. This is an especially important consideration for Japanese language users because several Japanese locales may be available on your system. One popular encoding scheme for Japanese is called Shift-JIS (pc932), and another is called EUC-Japan. A process operating with a file code of Shift-JIS cannot read text stored using EUC-Japan, and vice versa. If you find a file with characters you cannot read, you can use a program called **iconv** to transform it from U-JIS to Shift-JIS. See " Converting Files" in topic C.4.4 and *AIX Commands Reference* for information about the **iconv** command or ask your system administrator for help.

Files created in standard ASCII format on previous AIX systems can be read by a process operating under any AIX locales because ASCII is a subset of all AIX file codes. The opposite is not true. Files created in any Japanese locale cannot be read by the Base System. Files created under many European locales will be likewise unreadable by a program running under a Japanese locale. This is because many European locales use characters which look like Roman letters, but are not part of the ASCII set. Such characters are often Roman letters with special marks above or beside them to indicate special pronunciation.

C.2 The Japanese Keyboard on a PS/55

A Japanese keyboard is available on a number of machines, including the IBM RT and models 5570 and 5550 of the PS/55. This section describes the PS/55 keyboard, the special keys used for text entry, and the five text entry modes available on this keyboard.

Throughout this section, the term "Roman characters" refers to the letters of the alphabet used in languages derived from Latin. These characters are used in English.

Note: In order to enter and display Japanese characters on the PS/55, you need:

AIX Version 1.2.1

X-Windows

jaixterm

a Japanese keyboard such as the one pictured in the following figure.

Subtopics
C.2.1 Special Keys Used for Japanese Text Entry
C.2.2 Text Entry Modes on the Japanese Keyboard

C.2.1 Special Keys Used for Japanese Text Entry

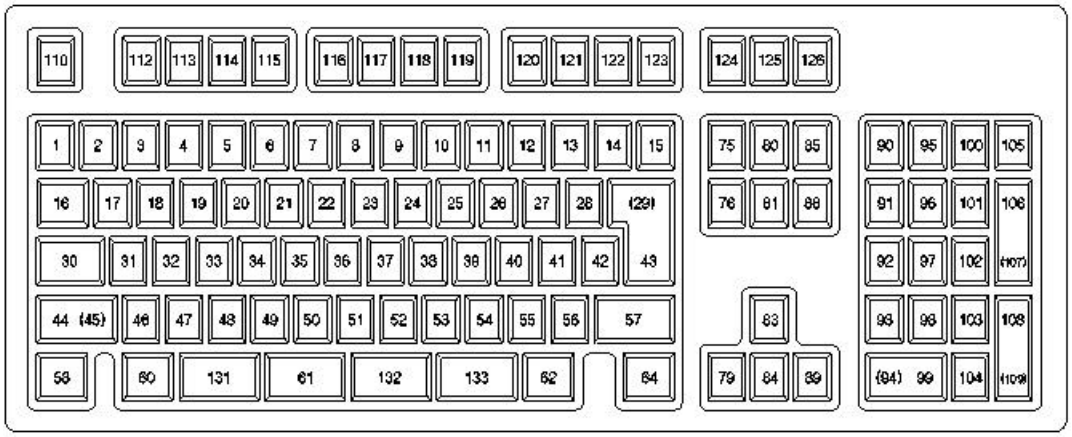


Figure C-1. The PS/55 Keyboard

The PS/55 keyboard has eight special keys designated to perform certain text entry functions. Use the diagram above and an actual keyboard to locate the following:

Enter
In text entry, this key performs two functions. A first press of this key signals that a group of characters are to be placed, as a group, in a buffer. This grouping mechanism is used in the process of transforming Hiragana to Kanji. The first press of the **Enter** key allows consecutive groups of Kana to be placed in the buffer. When the desired string has been assembled in the buffer, a second press of the **Enter** key sends the whole buffer to whichever program is currently waiting to receive input from the keyboard.

Hiragana Mode Selector
This key switches the keyboard into Hiragana entry mode. Entries made in this mode may later be transformed into Kanji.

Katakana Mode Selector
This key switches the keyboard into Katakana mode. Entries made in this mode may later be transformed into Kanji.

Roman Mode Selector
This key selects Roman mode for standard alphanumeric ASCII input. Half-width Roman entries are standard ASCII characters and are valid as command name entries for the standard commands delivered with the AIX system. Double-width Roman entries are valid only as text within a file.

Display Width Selector
This key switches the keyboard from single-width or double-width characters (also known as half-width or full-width, respectively).

Kanji Transform
This key transforms Hiragana or Katakana characters to Kanji characters. The characters to be transformed are always displayed on the screen as Hiragana or Katakana, but they can be entered when the keyboard is in Hiragana, Katakana, or Romaji entry mode.

Cancel Kanji Transform
This key cancels an attempted transformation from Hiragana to Kanji characters.

Romaji Mode Selector
This key used in combination with the **Hiragana Mode Selector** key places the keyboard in Romaji entry mode. In this mode, the characters accepted from the keyboard are Romaji, but the characters shown on the screen are Hiragana. This allows the user to spell out Japanese syllables in Roman letters. The input method turns these letters into Hiragana, which may then be further transformed into Kanji. This is a two-step process of spelling out Kanji in Roman letters.

These keys allow you to vary three components of the entries you are making: the character set accepted from the keyboard, the character set displayed on the screen, and the display width in which the characters are shown. These three values are independent of each other. You may select them separately. However, you should realize that some combinations do not make sense and cannot be recognized by the AIX system.

Subtopics
C.2.1.1 Character Set and Display Width

C.2.1.1 Character Set and Display Width

The character width displayed on the screen can be:

half-width One character per display column. This is also called single-width.

full-width One character for every two display columns. This is also called double-width.

Note: Both Hiragana and Kanji characters can be shown only in full-width display mode; both character sets would suffer too much from loss of resolution if shown in single-width mode.

The character set displayed on the screen can be any of the following:

Roman Single-width or double-width

Katakana Single-width or double-width

Hiragana Double-width only

Kanji Double-width only.

C.2.2 Text Entry Modes on the Japanese Keyboard

The Japanese keyboard has five text entry modes:

ASCII mode

This mode is the only valid mode for entering the standard AIX commands delivered with your AIX Operating System. It can also be used as an entry mode for the text within a file.

The keyboard inputs only the ASCII values marked on the various keys. Any ASCII character can be input, even if it does not constitute a full syllable. Each character is read from the keyboard as one byte and stored as one byte on disk. Each character takes up one display column on the screen.

You switch the keyboard into this mode by pressing the **Roman Mode Selector** and the **Display Width Selector** keys. You know you are in ASCII mode when the last line of the display reads **Roman, half width.**

Double-width Roman mode

This mode is only valid for entering text within a file. It cannot be used to enter the standard AIX commands delivered with the AIX Operating System.

Roman letters may be used to enter text in either of two modes, half-width and full-width. Half-width Roman letters are called ***ASCII mode***. Full-width Roman letters are called ***Double-width Roman***.

The purpose of Double-width Roman mode is the entry of Roman text within a text file which is composed primarily of Kana or Kanji. Japanese characters are usually printed and displayed in double-width format. Single-width ASCII characters look very strange in this context. Double-width Roman characters look much more appropriate.

In Double-width Roman mode, the keyboard inputs only the Roman letters marked on the various keys. Any Roman character can be input, even if it does not constitute a full syllable. Each character is read from the keyboard as two bytes and stored as two bytes on disk. Each takes up two display columns on the screen.

You switch to this mode by pressing the **Roman Mode Selector** and the **Display Width Selector** keys. You know the keyboard is in this input mode when the last line of the display reads **Roman, full width.**

Katakana mode

This mode is valid only for entering text within a file. It cannot be used to enter the standard AIX commands delivered with the AIX Operating System.

In this mode, the keyboard accepts only Japanese syllables. Some are stored as one byte and some as two bytes in length. The characters you type may be marked on the keyboard with either Katakana or Hiragana characters, but they always appear on the screen as Katakana. You can shift from full-width to half-width display characters while remaining in this mode.

You switch to this mode by pressing the **Katakana Mode Selector** key. You know that the keyboard is in this input mode when the last line of the display reads **Katakana half-width** or **Katakana full-width.**

Hiragana mode

This mode is valid only for entering text within a file. It cannot be used to enter the standard AIX commands delivered with the AIX Operating System.

Hiragana are the characters most often used to spell out Kanji in phonetic form. Like Romaji, the AIX system assumes that any Hiragana strings you enter are to be transformed at some point into Kanji. When you request this transformation to be done (by pressing the **Kanji Transform** key) the system will search memory for Kanji which match the sounds you entered.

In this Hiragana mode, the keyboard accepts only Japanese syllables. All are stored as two bytes in length. The characters you type may be marked on the keyboard with either Katakana or Hiragana characters, but they always appear on the screen as Hiragana. You cannot shift from full-width to half-width display characters while remaining in this mode. Only full-width Hiragana are accepted from the keyboard or shown on the screen.

You switch to this mode by pressing the **Hiragana Mode Selector** key. You know that the keyboard is in this input mode when the last line of the display reads **Hiragana half-width** or **Hiragana full-width**. (The **half-width** message, in this case, is false. The input method considers your input to be **full-width** no matter what the message says.)

Romaji mode

This mode is valid only for entering text within a file. It cannot be used to enter the standard AIX commands delivered with the AIX Operating System. This is a Roman-alphabet input mode coupled with a Hiragana display mode or Katakana display mode.

Romaji is a text entry method which maps the phonetic syllables of the Japanese language to groups of one, two or three Roman letters. It was devised for entering Japanese words from a keyboard that contains only Roman characters, and remains popular among those typists accustomed to a Roman-character keyboard. You may enter any valid Japanese syllable, but a string of Roman characters which does not correspond to a legitimate Kana is not recognized.

In Romaji mode, the keyboard accepts only Japanese syllables, but you can spell them out phonetically in the characters of the Roman alphabet. All characters are read from the keyboard as single-byte Roman characters, but, as the input method detects the proper spelling of any valid Japanese Kana character, that character appears on the screen in double-width Hiragana.

You switch to this mode by pressing the **Alt** - **Hiragana Mode Selector** keys. That is, press the **Alt** key and hold it down. Then, press the **Hiragana Mode Selector** key.

You know that the keyboard is in this input mode when the last line of the display reads **Hiragana, full-width R** or **Hiragana, half-width R.**

C.3 Entering Japanese Text

Whenever you want to enter Japanese text, you must start **jaixterm**, a program that runs under **x-server**. The **jaixterm** program recognizes multibyte characters and can display them on your screen. This section shows you how to start and stop **jaixterm**.

Subtopics
C.3.1 Starting jaixterm
C.3.2 Stopping jaixterm

C.3.1 Starting jaixterm

1. At the AIX prompt, enter:

   ```
   xinit
   ```

   The **x-server** program begins and displays the **aixterm** prompt.

2. At the **aixterm** prompt, enter:

   ```
   jaixterm
   ```

   Most of the screen becomes blank. The last line displays **Roman, half-width** in Japanese. This means you are in ASCII mode and can immediately enter AIX commands or you can switch to any of the other text entry modes, such as Hiragana, and enter text. For information about the text entry modes available and how to switch to them see "Text Entry Modes on the Japanese Keyboard" in topic C.2.2.

C.3.2 Stopping jaixterm

To stop **jaixterm** and return to the AIX prompt, do the following:

1. Press the **Roman Mode Selector** and the **Display Width Selector** keys.

   The last line of the display reads **Roman**, **half width**, which means you are in ASCII mode.

2. Press **Ctrl-Alt-Backspace**.

   The **jaixterm** program stops and the AIX prompt is displayed.

C.4 Working with Characters, Commands, and Files

This section covers the rules for using Japanese characters and describes techniques you can use to create an all-Japanese user environment. It also shows you how to convert files written in one code (for example, pc850) to another code (for example, ebcdic).

Subtopics

C.4.1 Rules for Using Japanese Characters

In general, AIX permits almost any interaction that can be handled in ASCII characters to be performed in Japanese characters. There are significant exceptions to this rule, and those are noted below:

Items which can be expressed in Japanese characters are:

- File name
- The contents of text file
- File and directory names in path name
- Aliases for shell scripts and command names (C shell users only
- Comments and literal strings in a C program

Items which cannot be entered in Japanese characters are:

- Site names (the names of cluster hosts
- Names of remote machine
- User login names and password
- Group name
- Names of environment variable
- Path name delimiter
- The contents (stanzas) of system attribute file
- The coding in a C progra
- Telephone numbers and dialer strings

All of the items in the previous list **must** be entered in ASCII characters only. Note that there are some interesting combinations of the rules given above:

- File names and path names can be entered in Japanese characters, bu the path name delimiter must be entered as the traditional ASCII slash (/).
- Only C shell users can create Japanese aliases for their commands Bourne shell users must enter command names and shell script names in ASCII.
- The comments and literal strings in a C program can be in Japanes characters, but the coding itself must be in ASCII.
- A telephone number can be written in Japanese characters if it is par of a text file, but must be in ASCII if it is to be passed to a modem.
- All system attribute files (like **/etc/ports**) must be in ASCII, but a user could write his own program in C, and that program could process its own user attribute file with Japanese contents.

The names of environment variables must be in ASCII characters, bu
the values to which they are set can be Japanese strings.

C.4.2 Techniques for Creating an All-Japanese User Environment

Many Japanese language users want to work entirely in Kana and Kanji. This is not possible for programmers and others doing advanced functions. Their work always needs to be some mixture of Japanese and ASCII entries. But typical users, doing a nonprogramming task like word processing, should be able to set up their own personal methodology as a Japanese-only system. Nearly all entries can be in one of the Japanese text entry modes, and most of what appears on the screen should be Japanese script.

The first step is to set up a series of personal directories with Japanese names. These may be filled with files that have Japanese names and contain Japanese text. The more difficult task is setting up a system of often-used commands under Japanese names. There are several ways to do that.

Subtopics

C.4.2.1 Aliasing Program Commands with Japanese Characters

The first technique is *aliasing*. This is an advanced operation available under the C shell only. It sets an entire string of characters equal to some other string. For instance, the ASCII command string **ls -al** could be set equal to a Japanese character string meaning "long list." Each time the Japanese characters for "long list" were entered on the AIX command line, the C shell would read them and execute the command **ls -al**. The user would then be given a long listing of all his files.

It is usually the job of the system administrator to set up a catalog of such aliases for all users on the system. This provides a consistent operating environment. Check with your system administrator to see which aliases are available for your use.

C.4.2.2 Linking Program Files to Japanese Names

The linking operation that you learned in Chapter 3 can also be used to provide alternative names to standard AIX programs. Here again, the system administrator should be the one to link the commands of the AIX system to Japanese strings that express their function. This is a bit more trouble for the user than the aliasing alternative but provides a much higher degree of flexibility. A linked command functions exactly like the standard command except that it can be invoked with a Japanese command name. It must be given its command options in ASCII characters, which requires a bit more effort for the user, but the user may use any legal combination of command flags rather than the limited set available with a set of aliases.

The user can also use the **ln** command to give commands Japanese names. On most systems, the system administrator may set up links for the often-used commands and leave each user to invent links for the infrequently used commands needed in his or her own work.

C.4.2.3 Writing New Programs or Scripts with Japanese Names

Any new programs or shell scripts you write can be given Japanese file names. Indeed, a user could devise a whole repertoire of shell scripts to do all of his ordinary work. For example, a script named "long list" in Japanese characters could have the following contents:

```
ls -al
```

When run, this script gives the user a long listing of the directory in which it was used.

This aliasing technique runs faster, but single-line scripts of this sort are an option for those users who prefer the Bourne shell, under which aliasing is not available. Most Japanese-named shell scripts should probably be new procedures or multi-step operations. The existing one-step functions of the standard AIX commands are best handled by aliasing.

New programs written in C can be given names in Japanese characters and can be written so as to accept their options and arguments in Japanese text.

C.4.3 Restrictions

Never create Japanese-named commands in either of the following ways:

1. By copying standard AIX commands to Japanese-named files with **cp**, or

2. By renaming standard AIX commands with the **mv** command.

There are hundreds of standard AIX commands. Duplicating any significant fraction of these would waste large volumes of disk space.

Renaming standard AIX commands makes them unavailable to users who might want to run them under their standard names. In addition, many AIX system facilities are shell scripts which accomplish their functions by calling standard AIX programs through their established names. Any such shell script will become useless.

Always create your Japanese-named commands with either the link procedure or the alias procedure described above.

C.4.4 Converting Files

On a system that supports more than one character code for files, users may create a file in one code and convert it to another code by means of the **iconv** command. For example, a Japanese language user who uses Shift-JIS may want to communicate with a user who uses U-JIS. The creator of the file needs to convert it to a code that is readable by the recipient.

The **iconv** command is used to make such a conversion. This command operates in standard AIX fashion. Unless an input file is specified, **iconv** takes its input from standard input; therefore, it can be used from the command line or made part of a pipe operation. The character codes that **iconv** can convert are:

Code Page 850 (specified as **pc850**)

Code Page 932 (specified as **pc932**)

ISO 8859-1 (specified as **8859-1**)

U-JIS (specified as **ujis**)

Host Code (specified as **ebcdic**).

The following conversions are supported:

Between pc850 and 8859-

Between pc932 and uji

Between pc850 and ebcdi

Between pc932 and ebcdi

Between ujis and ebcdic

The format of the **iconv** command is:

iconv -fcode-tcode[-dstring][-ofile][file]

The **-f** flag specifies the character code of the input file.

The **-t** flag specifies the character code of the output file.

The **-d** flag specifies the character or string used in the output file to convert an unrecognized character from the input file. If it is omitted, the character used depends on the conversion table for the locale.

The **-o** flag specifies the name of the output file. If it is omitted, standard output is used.

If the name of the input file is omitted, standard input is used.

An example of using **iconv** is:

```
iconv -fpc932 -tujis -oujis.out sjis.in
```

This command converts a Shift-JIS input file named **sjis.in** to a U-JIS output file named **ujis.out**.

One use of **iconv** is to convert files to host code for printing them on VM system printers. For normal printing, this is done by the print drivers; the user does not need to include **iconv** in the standard pipe for sending files to the printer. The user could use **iconv** to convert files to host code when shipping them to a system requiring that code.

For more information about the **iconv** command, see *AIX Commands Reference*.

GLOSSARY Glossary

access. To obtain computing services.

access permission. A group of designations that determine who can access a particular AIX file and how the user may access the file.

account. The login directory and other information that give a user access to the system.

activity manager. A collection of system programs allowing users to manage their activities. Provides the ability to list current activities (Activity List) and to begin, cancel, hide, and activate activities.

All Points Addressable (APA) display. A display that allows each pel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with **character display**.

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic. Pertaining to a set of letters a through z.

alphanumeric character. Consisting of letters, numbers, and often other symbols, such as punctuation marks and mathematical symbols.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute. An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

application. A program or group of programs that directly apply to a particular user problem, such as inventory control, word processing or accounts receivable.

application program. A program used to perform an application or part of an application.

argument. Numbers, letters, or words that change the way a command works.

ASCII. See **American National Standard Code for Information Interchange**.

asynchronous transmission. In data communication, a method of transmission in which the bits included in a character or block of characters occur during a specific time interval. However, the start of each character or block of characters can occur at any time during this interval. Contrast with **synchronous transmission**.

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

authorize. To grant to a user the right to communicate with, or make use of, a computer system or display station.

auto carrier return. The system function that places carrier returns automatically within the text and on the display. This is accomplished by moving whole words that exceed the line end zone to the next line.

backend. The program that sends output to a particular device. There are two types of backends: friendly and unfriendly.

background process. (1) A process that does not require operator intervention that can be run by the computer while the work station is used to do other work. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

backup copy. A copy, usually of a file or group of files, that is kept in case the original file or files are unintentionally changed or destroyed.

backup diskette. A diskette containing information copied from a fixed disk or from another diskette. It is used in case the original information becomes unusable.

bad block. A portion of a disk that can never be used reliably.

base address. The beginning address for resolving symbolic references to locations in storage.

base name. The last element to the right of a full path name. A file name specified without its parent directories.

batch printing. Queueing one or more documents to print as a separate job. The operator can type or revise additional documents at the same time. This is a background process.

batch processing. A processing method in which a program or programs process records with little or no operator action. This is a background

process. Contrast with **interactive processing**.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. Eight bits make a byte. See also **byte**.

block. (1) A group of records that is recorded or processed as a unit. Same as **physical record**. (2) In data communication, a group of records that is recorded, processed, or sent as a unit. (3) A physical block in AIX is 4096 bytes long. (4) A logical block in AIX is 1024 bytes long.

block file. A file listing the usage of blocks on a disk.

block special file. A special file that provides access to an input or output device is capable of supporting a file system. See also **character special file**.

bootstrap. A small program that loads larger programs during system initialization. Sometimes referred to as IPL (Initial Program Load).

bps. Bits per second.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

burst pages. On continuous-form paper, pages of output that can be separated at the perforations.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with **load**.

callouts. An AIX kernel parameter establishing the maximum number of scheduled activities that can be pending simultaneously.

cancel. To end a task before it is completed.

carrier return. (1) In text data, the action causing line ending formatting to be performed at the current cursor location followed by a line advance of the cursor. Equivalent to the carriage return of a typewriter. (2) A keystroke generally indicating the end of a command line.

case sensitive. Able to distinguish between uppercase and lowercase letters.

character. A letter, digit, or other symbol.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display cannot address the screen any less than one character box at a time. Contrast with **All Points Addressable display**.

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with **function keys**.

character position. On a display, each location that a character or symbol can occupy.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character special file. A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. See also **block special file**.

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

child. (1) Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. (2) In the AIX Operating System child is a **process** spawned by a parent process that shares the attributes of the parent process. Contrast with **parent**.

C language. A general-purpose programming language that is the primary language of the AIX Operating System.

class. Pertaining to the I/O characteristics of a device. AIX devices are classified as block or character.

close. (1) To end an activity and remove that window from the display.

code. (1) Instructions for the computer. (2) To write instructions for the computer; to **program**. (3) A representation of a condition, such as an error code.

code segment. See **segment**.

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

column. A vertical arrangement of text or numbers.

column headings. Text appearing near the top of columns of data for the purpose of identifying or titling.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command interpreter. A program (such as the Bourne or C shell) that sends instructions from the command line to the kernel.

command line. The area of the screen where commands are displayed as they are typed.

command line editing keys. Keys for editing the command line.

command name. (1) The first or principal term in a command. A command name does not include parameters, arguments, flags, or other operands. (2) The full name of a command when an abbreviated form is recognized by the computer (for example, print working directory for pwd).

command programming language. Facility that allows programming by the combination of commands rather than by writing statements in a conventional programming language. See **shell procedure**.

communication adapter. A hardware feature enabling a computer or device to become of a data communication network.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compress. (1) To move files and libraries together on disk to create one continuous area of unused space. (2) In data communication, to delete a series of duplicate characters in a character string.

concatenate. (1) To link together. (2) To join two character strings.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

configuration. The group of machines, devices, and programs that make up a computer system. See also **system customization**.

configuration file. A file that specifies the characteristics of a system or subsystem, for example, the AIX queueing system.

consistent. Pertaining to a file system, without internal discrepancies.

console. (1) The main AIX display station. (2) A device name associated with the main AIX display station.

constant. A data item with a value that does not change. Contrast with **variable**.

context search. A search through a file whose target is a character string.

control block. A storage area used by a program to hold control information.

control commands. Commands that allow conditional or looping logic flow in shell procedures.

control program. Part of the AIX Operating System that determines the order in which basic functions should be performed.

controlled cancel. The system action that ends the job step being run,

and saves any new data already created. The job that is running can continue with the next job step.

copy. The action by which the user makes a whole or partial duplicate of already existing data.

coupler. A device connecting a modem to a telephone network.

crash. An unexpected interruption of computer service, usually due to a serious hardware or software malfunction.

current directory. The directory that is the starting point for relative path names.

current line. The line on which the cursor is located.

current working directory. See **current directory**.

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. (2) A marker that indicates the current data access location within a file.

cursor movement keys. The directional keys used to move the cursor.

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See **daemon process**.

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times to more than one task or user, such as sending data to a printer.

data block. See **block**.

data communication. The transmission of data between computers, or remote devices or both (usually over long distance).

data link. The equipment and rules (protocols) used for sending and receiving data.

data stream. All information (data and control information) transmitted over a data link.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

default. A value that is used when no alternative is specified by the operator.

default directory. The directory name supplied by the operating system if none is specified.

default drive. The drive name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

delete. To remove. For example, to delete a file.

dependent work station. A work station having little or no standalone capability, that must be connected to a host or server in order to provide any meaningful capability to the user.

device. An electrical or electronic machine that is designed for a specific purpose and that attaches to your computer, for example, a printer, plotter, disk drive, and so forth.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device manager. Collection of routines that act as an intermediary between device drivers and virtual machines for complex interfaces. For example, supervisor calls from a virtual machine are examined by a device manager and are routed to the appropriate subordinate device drivers.

device name. A name reserved by the system that refers to a specific device.

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic aid. A tool (procedure, program, reference manual) used to detect and isolate a device or program malfunction or error.

diagnostic routine. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

digit. Any of the numerals from 0 through 9.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

discipline. Pertaining to the order in which requests are serviced, for example, first-come-first-served (fcfs) or shortest job next (sjn).

disk I/O. Fixed-disk input and output.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copies from the disk or another diskette.

diskette drive. The mechanism used to read and write information on diskettes.

display device. An output unit that gives a visual representation of data.

display screen. The part of the display device that displays information visually.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

distributed file system. A file system whose files, directories, and other components are stored on different sites in a particular cluster.

distributed operating system.. An operating system where multiple machines cooperate to seem like one machine.

distributed processing. Results when a user involves multiple cluster sites in a single operation--for example, by editing a remote file and starting a task on another cluster site using the **on**, **fast**, **fastsite**, and **migrate** commands.

Distributed Services (DS). A licensed program that allows you to share files with other AIX systems in a network. You can mount the file systems located on other AIX systems to created file trees that are independent of

the file systems.

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

dump diskette. A diskette that contains a dump or is prepared to receive a dump.

dump formatter. Program for analyzing a dump.

EBCDIC. See **extended binary-coded decimal interchange code**.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To modify the form or format of data.

edit buffer. A temporary storage area used by an editor.

editor. A program used to enter and modify programs, text, and other types of documents and data.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

enter. To send information to the computer by pressing the **Enter** key.

entry. A single input operation on a work station.

environment. The settings for shell variables and paths set associated with each process. These variables can be modified later by the user.

error-correct backspace. An editing key that performs editing based on a cursor position; the cursor is moved one position toward the beginning of the line, the character at the new cursor location is deleted, and all characters following the cursor are moved one position toward the beginning of the line (to fill the vacancy left by the deleted element).

escape character. A character that suppresses the special meaning of one or more characters that follow.

Ethernet. A physical medium through which computers in the same or

different clusters can communicate and share files.

EUC. Extended UNIX Code.

exit value. A numeric value that a command returns to indicate whether it completed successfully. Some commands return exit values that give other information, such as whether a file exists. Shell programs can test exit values to control branching and looping. Exit values are also called Return Codes.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

feature. A programming or hardware option, usually available at an extra cost.

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name.

FIFO. See **first-in-first-out**.

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file. See also **label**.

file name. In DOS, that portion of the file name that precedes the extension.

file specification (filespec). The name and location of a file. A file specification consists of a drive specifier, a path name, and a file name.

file system. A collection of files and directories stored on logical and physical devices (such as disks) and logically organized in a hierarchical fashion.

filetab. An AIX kernel parameter establishing the maximum number of files that can be open simultaneously.

filter. A command that reads standard input data, modifies the data, and sends it to standard output.

first-in-first-out (FIFO). A named permanent pipe. A FIFO allows two unrelated processes to exchange information using a pipe connection.

fixed disk. A flat, circular, nonremovable plate with a magnetizable surface layer on which data can be stored by magnetic recording.

fixed-disk drive. The mechanism used to read and write information on fixed disk.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System are almost always preceded by a dash.

font. A family or assortment of characters of a given size and style.

foreground. A mode of program execution in which the shell waits for the program specified on the command line to complete before returning your prompt.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

formatted diskette. A diskette on which control information for a particular computer system has been written but which may or may not contain any data.

free list. A list of available space on each file system. This is sometimes called the free-block list.

free-block list. See **free list**.

full path name. The name of any directory or file expressed as a string of directories and files beginning with the root directory.

function. A synonym for procedure. The C language treats a function as a data type that contains executable code and returns a single value to the calling function.

function keys. Keys that request actions but do not display or print characters. Included are the keys that normally produce a printed character, but when used with the code key produce a function instead. Compare with **character key**.

generation. For some remote systems, the translation of configuration

information into machine language.

Gid. See **group number**.

global. Pertains to information available to more than one program or subroutine.

global action. An action having general applicability, independent of the context established by any task.

global character. The special characters * and ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position.

global search. The process of having the system look through a document for specific characters, words, or groups of characters.

global variable. A symbol defined in one program module but used in other independently assembled program modules.

graphic character. A character that can be displayed or printed.

group name. A name that uniquely identifies a group of users to the system.

group number (Gid). A unique number assigned to a group of related users. The group number can often be substituted in commands that take a group name as an argument.

hardware. The equipment, as opposed to the programming, of a computer system.

header. Constant text that is formatted to be in the top margin of one or more pages.

header label. A special set of records on a diskette describing the contents of the diskette.

here document. Data contained within a shell program or procedure (also called **inline input**).

hexadecimal. Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

hierarchical tree structure. The organization of files on AIX, similar to tree-structured directories, with each file like a small branch of a larger branch that represents the file's parent directory. A directory can also be contained in another higher level directory, with the parent of all directories represented by the tree's root (***root*** or ***root directory***).

highlight. To emphasize an area on the display by any of several methods, such as brightening the area or reversing the color of characters within the area.

history. A C-shell mechanism that lists previously executed commands. These commands can be re-executed with the **!** command.

history file. A file containing a log of system actions and operator responses.

hog factor. In system accounting, an analysis of how many times each command was run, how much processor time and memory it used, and how intensive that use was.

home directory. The directory a user accesses when he logs in. Here he may create and delete files and directories to organize his work. Synonym for **login directory**.

home site. The computer that stores the modifiable copy of a user's home directory. This is the cluster site with the primary copy of his home directory if it is replicated. A user typically logs in to the computer that is his home site.

I/O. See **input/output**.

ID. Identification.

IF expressions. Expressions within a procedure, used to test for a condition.

indirect block. A block containing pointers to other blocks. Indirect blocks can be single-indirect, double-indirect, or triple-indirect.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See **initialize, bootstrap**.

initialize. To set counters, switches, addresses, or contents of storage

to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine.

inline input. See **here document**.

inode. The internal structure for managing files in the system. Inodes contain all of the information pertaining to the node, type, owner, and location of a file. A table of inodes is stored near the beginning of a file system.

i-number. A number specifying a particular inode on a file system.

inodetab. An AIX kernel parameter that establishes a table in memory for storing copies of inodes for all active files.

input. Data to be processed.

input device. Physical devices used to provide data to a computer.

input file. A file opened by a program so that the program can read from that file.

input list. A list of variables to which values are assigned from input data.

input redirection. The specification of an input source other than the standard one.

input-output file. A file opened for input and output use.

input-output device number. A value assigned to a device driver by the guest operating system or to the virtual device by the virtual resource manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input/output (I/O). Pertaining to either input, output, or both between a computer and a device.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with **batch processing**.

interface. A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interleave factor. Specification of the ratio between contiguous physical blocks (on a fixed-disk) and logically contiguous blocks (as in a file).

interrupt. (1) To temporarily stop a process. (2) In data communication, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

interrupt character. A key sequence (**Alt-Pause** on some systems) typed in to cancel a foreground process.

IPL. See **initial program load**.

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job control. A feature that lets the system accept your commands to stop and start processes (jobs) and move them between background and foreground. The commands **ps** and **jobs** report the status of jobs, (each of which is assigned a Process Identification Number or PID to show its process status), and the **kill** command can be used to stop them.

job number. A number assigned to a background process when it is started. The job number is displayed when the process is started and when the **jobs** or **ps** command is invoked. It can also be used to kill the process.

job queue. A list, on disk, of jobs waiting to be processed by the system.

justify. To print a document with even right and left margins.

kbuffers. An AIX kernel parameter establishing the number of buffers that can be used by the kernel.

K-byte. See **kilobyte**.

kernel. The memory-resident nucleus of the AIX Operating System containing functions needed immediately and frequently. The kernel supervises the input and output, manages and controls the hardware, and schedules the user processes for execution.

kernel parameters. Variables that specify how the kernel allocates certain system resources.

key pad. A physical grouping of keys on a keyboard (for example, numeric key pad, and cursor key pad).

keyboard. An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the dialog between the user and the display station

keylock feature. A security feature in which a lock and key can be used to restrict the use of the display station.

keyword. One of the predefined words of a programming language; a reserved word.

keyword argument. One type of variable assignment that can be made on the command line.

kill. An AIX Operating System command that stops a process.

kill character. The character that is used to delete a line of characters entered after the user's prompt.

kilobyte. 1024 bytes.

kprocs. An AIX kernel parameter establishing the maximum number of processes that the kernel can run simultaneously.

label. (1) The name in the disk or diskette volume table of contents that identifies a file. See also **file name**. (2) The field of an instruction that assigns a symbolic name to the location at which the instruction begins, or such a symbolic name.

left margin. The area on a page between the left paper edge and the leftmost character position on the page.

left-adjust. The process of aligning lines of text at the left margin or at a tab setting such that the leftmost character in the line or filed is in the leftmost position. Contrast with **right-adjust**.

library. A collection of functions, calls, subroutines, or other data.

licensed program (LP). Software programs that remain the property of the manufacturer, for which customers pay a license fee.

line editor. An editor that modifies the contents of a file one line at a time.

linefeed. An ASCII character that causes an output device to move forward one line.

link. A connection between an inode * and one or more file names associated with it. Synonym for **UNIX link** or **hard link**.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

loader. A program that reads run files into main storage, thus preparing them for execution.

local. Pertaining to a device directly connected to your system without the use of a communication line. Contrast with **remote**.

<LOCAL> alias. The **<LOCAL>** alias can translate into different strings on different cluster sites for different processes. When **<LOCAL>** is the first component of the destination name for a symbolic link, it is replaced with its alias string, normally /**machinename**.

local cluster site. The site on a cluster that the user is logged in to. The term **local** normally refers to a TCF cluster site.

<LOCAL> file system. The part of the root file system hierarchy comprising system directories and files (such as the /etc/motd "message of the day" file) defined uniquely on a particular computer in the cluster. These files are not replicated. The name of the **<LOCAL>** file system appears in response to the **site-l** command.

location transparency. Allows an object to change location without the user's or program's knowledge if that location is not part of the object's name. For example, /u/joe/glossary may have been a file on **eyore** last week, but it is a file on **pooh** this week. Joe not need to know that the file was on either **eyore** or **pooh**. If, however, Joe wants to find out where the site is located, he may invoke the **where** command.

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

log in. To begin a session at a display station.

log off. See **log out**.

log on. See **log in**.

log out. To end a session at a display station.

logical device. A file for conducting input or output with a physical device.

login directory. See **home directory**.

login shell. The program, or command interpreter, started for a user at log in.

login user ID. The ID the user uses to log in. The system uses this ID to trace all user actions to their source.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

LP. See **licensed program**.

mailbox. An area designated for storage of mail messages directed to a specific system user.

main storage. The part of the processing unit where programs are run.

maintenance system. A special version of the AIX Operating System which is loaded from diskette and used to perform system management tasks.

major device number. A system identification number for each device or type of device.

mapped files. Files on the fixed-disk that are accessed as if they are in memory.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

matrix. An array arranged in rows and columns.

maxprocs. An AIX kernel parameter establishing the maximum number of processes that can be run simultaneously by a user.

memory. Storage on electronic chips. Examples of memory are random access memory, read only memory, or registers. See **storage**.

menu. A displayed list of items from which an operator can make a selection.

message. (1) A response from the system to inform the operator of a condition which may affect further processing of a current program. (2) Information sent from one user in a multi-user operating system to another.

minidisk. A logical division of a fixed disk.

minor device number. A number used to specify various types of information about a particular device, for example, to distinguish among several printers of the same type.

mode word. An inode field that describes the type and state of the inode.

modem. See **modulator-demodulator**.

modulation. Changing the frequency or size of one signal by using the frequency or size of another signal.

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communication line, and converts the signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) See **load module**.

mount. To make a file system accessible.

mountab. An AIX kernel parameter establishing the maximum number of file systems that can be mounted simultaneously.

multiprogramming. The processing of two or more programs at the same time on the same logical system.

multivolume file. A diskette file occupying more than one diskette.

multi-user environment. A computer system that provides terminals and keyboards for more than one user at the same time.

IBM AIX Network File System (NFS). A licensed program that allows you to share files with other computers in one or more networks that have a variety of machine types and operating systems. You can mount file systems located on network servers and use remote files as if they were on your work stations by creating file trees that are independent of the file systems.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

network. A collection of computers that can communicate with each other. A network can consist of several interconnected computers or one computer with a number of remote terminals connected to it. Any of a variety of communication media can be used, such as RS232, Ethernet, PC Net, etc.

new-line character. A control character that causes the print or display position to move to the first position on the next line.

node. An individual element of a full path name. Nodes are separated by slashes (/).

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object code. Machine-executable instruction, usually generated by a compiler from source code written in a higher level language. consists of directly executable machine code. For programs that must be linked, object code consists of relocatable machine code.

octal. A base eight numbering system.

online. Being controlled directly by, or communicating directly with, the computer, or both.

open. To make a file available to a program for processing.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action (such as move, add, multiply, load) that the computer performs when requested.

operator. A symbol representing an operation to be done.

output. The result of processing data.

output devices. Physical devices used by a computer to present data to a user.

output file. A file that is opened by a program so that the program can write to that file.

output redirection. The specification of an output destination other than the standard one.

overflow condition. A condition that occurs when part of the output of an operation exceeds the capacity of the intended storage unit.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write output into a storage or file space that is already occupied by data.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A block of instructions, data, or both.

page space. The area on a fixed disk that temporarily stores instructions or data currently being run. See also **minidisk**.

pagination. The process of adjusting text to fit within margins and/or page boundaries.

paging. The action of transferring instructions, data, or both between real storage and external page storage.

parallel processing. The condition in which multiple tasks are being performed simultaneously within the same activity.

parameter. Information that the user supplies to a panel, command, or function.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with **child**.

parent directory. The directory one level above the current directory.

partition. See **minidisk**.

password. A string of characters that, when entered along with a user identification, allows an operator to sign on to the system.

password security. A program product option that helps prevent the unauthorized use of a display station by checking the password entered by each operator at log in.

path name. The sequential list of directory name(s) that identify the location of a particular directory, and directory name(s) and file name that identify the location of a particular file in the file hierarchy. The path name is displayed in response to the **pwd** (print working directory) command (the ~, or tilde, may appear if you're in your home directory). Each file has a full path name, beginning with / (the root directory) and ending with the file's name. The file's relative path name does not begin with /.

pattern-matching character. Special characters such as * or ? that can be used in search patterns. Some used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position. Pattern-matching characters are also called wildcards.

permission code. A three-digit octal code, or a nine-letter alphabetic code, indicating the access permissions. The access permissions are read, write, and execute.

permission field. One of the three-character fields within the permissions column of a directory listing indicating the read, write, and run permissions for the file or directory owner, group, and all others.

phase. One of several stages file system checking and repair performed by the **fsck** command.

physical device. See **device**.

physical file. An indexed file containing data for which one or more

alternative indexes have been created.

physical record. (1) A group of records recorded or processed as a unit. Same as **block**. (2) A unit of data moved into or out of the computer.

PID. See **process ID**.

pipe. To direct the data so that the output from one process becomes the input to another process.

pipeline. A direct, one-way connection between two or more processes.

pitch. A unit of width of typewriter type, based on the number of times a letter can be set in a linear inch. For example, 10-pitch type has 10 characters per inch.

platen. The support mechanism for paper on a printer, commonly cylindrical, against which printing mechanisms strike to produce an impression.

pointer. A logical connection between physical blocks; a link to something else; an address.

port. (1) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer. (2) A part of the system unit or remote controller to which cables for display stations and printers are attached.

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A shell facility for assigning values from the command line to variables in a program.

primary copy. Each replicated file system has a copy designated as the **primary copy**, which is the copy that may be modified. It resides on the **primary site** and its purpose is to guarantee that file updates are kept consistent.

primary site. The cluster site that maintains the primary copy of a replicated file system.

print queue. A file containing a list of the names of files waiting to be printed.

printout. Information from the computer produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

priority number. A number that establishes the relative priority of printer requests.

privileged user. The account with superuser authority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communication facilities, or user errors as the source of the problem.

problem determination procedure. A prescribed sequence of steps aimed at recovery from, or circumvention of, problem conditions.

procedure. See **shell procedure**.

process. A program now running. A **foreground** process executes as soon as you type in the command line and completes before returning the system prompt to accept your next command. You can start one or more **background processes** to run independently while you type in a separate command for another process to run in the foreground.

process accounting. An analysis of the use each process makes of the processing unit, memory, and I/O resources.

process ID (PID). A unique number assigned to a process that is running.

process transparency. The ability to execute and control tasks on any site in the cluster, regardless of where the user is logged in (to find out where that is, type the **site** command). The same system calls and commands are used, no matter where the process is located. For example, a remote job is aborted the same way that a local job is abandoned.

profile. (1) A file containing customized settings for a system or user (2) Data describing the significant features of a user, program, or device.

program. A set of instructions for the computer to interpret and execute.

prompt. A displayed request for information or operator action.

propagation time. The time necessary for a signal to travel from one point on a communication line to another.

protocol. In data communication, the rules for transferring data.

protocol procedure. A process that implements a function for a device manager. For example, a virtual terminal manager may use a protocol procedure to interpret the meaning of keystrokes.

qdaemon. The daemon process that maintains a list of outstanding jobs and sends them to the specified device at the appropriate time.

queue. A line or list formed by items waiting to be processed.

queued message. A message from the system that is added to a list of messages stored in a file for viewing by the user at a later time. This is in contrast to a message that is sent directly to the screen for the user to see immediately.

quit. A key, command, or action that tells the system to return to a previous state or stop a process.

quote. To mask the special meaning of certain characters; to cause them to be taken literally.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

readonly. Pertaining to file system mounting, a condition that allows data to be read, but not modified.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

redirect. To divert data from a process to a file or device to which it would not normally go.

reference count. In an inode, a record of the total number of directory entries that refer to the inode.

relational expression. A logical statement describing the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a

relational condition or a relational expression.

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relative addressing. A means of addressing instructions and data areas by designating their locations relative to some symbol.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remote. Pertaining to a system or device that is connected to your system through a communication line. Contrast with **local**.

remote cluster site. A site on the cluster that the user is not logged in to. The term **remote** normally refers to a TCF cluster site.

replicated root file system. The replicated root file system is a file system with key common files and directories for basic system operation. Almost all system binaries, programs and libraries are in the replicated root file system. Other user and system file systems (like the local file system) are mounted on top of directories in the replicated root file system.

reserved character. A character or symbol that has a special (non-literal) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose, and that must not appear as a user-declared identifier.

reset. To return a device or circuit to a clear state.

restore. To return to an original value or image. For example, to restore a library from diskette.

right adjust. The process of aligning lines of text at the right margin or tab setting such that the rightmost character in the line or file is in the rightmost position.

right justify. See right align.

right margin. The area on a page between the last text character and the right upper edge.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with **left-adjust**.

root. Another name sometimes used for superuser.

root directory. The top level of a tree-structured directory system.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

run. To cause a program, utility, or other machine function to be performed.

run-time environment. A collection of subroutines and shell variables that provide commonly used functions and information for system components.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen. See **display screen**.

scroll. To move information vertically or horizontally to bring into view information that is outside the display screen boundaries.

secondary copy. A read-only copy of the **primary copy** of a replicated file system. Files in the **secondary copy** are automatically modified or deleted when the corresponding file in the **primary copy** is modified or deleted. New files added to the **primary copy** will be automatically added to the **secondary copy** only if the **fstore** value has been set.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

semantic transparency. Allow the same command to function identically from all cluster sites. It provides, for example, for the **grep** command to have the same options and give the same results no matter where it is

invoked.

separator. A character used to separate parts of a command or file.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

server. A program that handles protocol, queuing, routing, and other tasks necessary for data transfer between devices in a computer system.

session. The period of time during which programs or devices can communicate with each other.

session records. In the accounting system, a record of time connected and line usage for connected display stations, produced from log in and log out records.

set flags. Flags that can be put into effect with the shell set command.

shared printer. A printer that is used by more than one work station.

shell. See **shell program**.

shell options. The shell provides two different types of options. **Set options** are put into effect with the **set** command and alter the way the shell runs. **Command line options** are entered on the command line (but not with the **set** command) and alter the way the shell starts.

shell procedure. A series of commands combined in a file that carry out a particular function when the file is run or when the file is specified as an argument to the sh command. Shell procedures are frequently called shell scripts.

shell program. A program that accepts and interprets commands for the operating system (there is an AIX shell program and a DOS shell program).

shell prompt. The character string on the command line indicating the system can accept a command (typically the **$** character).

shell script. See **shell procedure**.

shell variables. Facilities of the shell program for assigning variable values to names.

Shift-JIS. A mixed single- and double-byte code representing the ASCII control and graphic characters and the extended characters for single- and double-byte Katakana and double-byte Hiragana, Kanji, and Roman. Also called code page 932 and sometimes referred to as file code or multibyte code.

size field. In an inode, a field that indicates the size, in bytes, of the file associated with the inode.

software. Programs.

sort. To rearrange some or all of a group of items based upon the contents or characteristics of those items.

source diskette. The diskette containing data to be copied, compared, restored, or backed up.

source program. A set of instructions written in a programming language, that must be translated to machine language and compiled before the program can be run.

special character. A character other than an alphabetic or numeric character. For example, *, +, and % are special characters.

special file. Special files are used in the AIX system to provide an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with **directory** and **file**. See also **block special file** and **character special file**.

spool file. (1) A disk file containing output that has been saved for later printing. (2) A file used in transmitting data among devices.

standalone shell. A limited version of the shell program used for system maintenance.

standalone work station. A work station that can be used to preform tasks independent of (without being connected to) other resources such as servers or host systems.

standard error. The place where many programs place error messages.

standard input. The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. The primary destination of data coming from a command.

Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command.

stanza. A group of lines in a file that together have a common function. Stanzas are usually separated by blank lines, and each stanza has a name.

statement. An instruction in a program or procedure.

status. (1) The current condition or state of a program or device. For example, the status of a printer. (2) The condition of the hardware or software, usually represented in a status code.

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See **memory**.

storage device. A device for storing and/or retrieving data.

string. A series of characters to be taken literally by the system. A string may be specified for a context search, for instance, or for global substitutions.

su. See **superuser**.

subdirectory. A directory contained within another directory in the file system hierarchy.

subprogram. A program invoked by another program, such as a subshell.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subshell. An instance of the shell program started from an existing shell program.

substitution. A procedure used by a text editor like **ed** or **vi** to replace one specified string of characters with another. If a global substitution is made, all occurrences of the specified text pattern are replaced with the new one.

substring. A part of a character string.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

superblock. The most critical part of the file system containing information about every allocation or deallocation of a block in the file system.

superuser. Super user authority; root permissions.

synchronous. Occurring in a regular or predictable sequence.

synchronous transmission. In data communication, a method of transmission in which the sending and receiving of characters is controlled by timing signals. Contrast with **asynchronous transmission**.

system. The computer and its associated devices and programs.

superuser. The user who can operate without the restrictions designed to prevent data loss or damage to the system (User ID 0). Also known as **superuser** or **su**.

superuser authority. The unrestricted ability to access and modify any part of the operating system associated with the user who manages the system. The authority obtained when one logs in as **root**.

symbolic link. Type of file that contains the path name to another file or a directory; it functions as a pointer to the other file or directory. See **link**.

system call. A request by an active process for a service by the system kernel.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system date. The date assigned by the system user during setup and maintained by the system.

system dump. A copy of memory from all active programs (and their associated data) whenever an error stops the system. Contrast with **task dump**.

system management. The tasks involved in maintaining the system in good working order and modifying the system to meet changing requirements.

system parameters. See **kernel parameters**.

system primary site. The machine (cluster site) designated to hold the primary copy of the replicated root file system. When files are changed in the replicated root file system, the primary site for the cluster must be available.

system profile. A file containing the default values used in system operations.

system-replicated file system. One that contains files and directories accessed by many users regardless of the users' specific applications. These system files, programs and directories are replicated on different sites in a cluster.

system unit. The part of the system that contains the processing unit, the disk drives, and the diskette drives.

system user. A person, process, or other resource that uses the facilities of a computer system.

systems network architecture (SNA). A set of rules for controlling the transfer of information in a data communication network.

target diskette. The diskette to be used to receive data from a source diskette.

task. A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

task dump. A copy of memory from a program that failed (and its associated data). Contrast with **system dump**.

TCF. Transparent Computing Facility. TCF is an operating system that automatically allows for data, process, name, location and semantic transparency. Process transparency is the ability to execute and control tasks on any cluster site, no matter where the user program is currently executing. A TCF LPP is required to obtain support.

terminal. An input/output device containing a keyboard and either a display device or a printer. Terminals usually are connected to a computer and allow a person to interact with the computer.

text. A type of data consisting of a set of linguistic characters (for example, alphabet, numbers, and symbols) and formatting controls.

text application. A program defined for the purpose of processing text

data (for example, memos, reports, and letters).

text editing program. See **editor** and **text application**.

texttab. A kernel parameter establishing the size of the text table, in memory, that contains one entry each active shared program text segment.

trace. To record data that provides a history of events occurring in the system.

trace table. A storage area into which a record of the performance of computer program instructions is stored.

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

transfer. To move data from one location to another in a computer system or between two or more systems.

transmission control characters. In data communication, special characters that are included in a message to control communication over a data link. For example, the sending station and the receiving station use transmission control characters to exchange information; the receiving station uses transmission control characters to indicate errors in data it receives.

transparency. The obscuring of machine boundaries in a distributed system. The IX/370 system supports several kinds of transparency, including name, location, semantic, data, and process transparency.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

tree-structured directories. A method for connecting directories such that each directory is listed in another directory except for the root directory, which is at the top of the tree.

truncate. To shorten a field or statement to a specified length.

TTY. Designates a terminal. On a system with more than one terminal, the TTY field of the process status displayed by the **ps** command indicates which terminal started the process.

typematic key. A key that repeats its function multiple times when held down.

typestyle. Characters of a given size, style and design.

Uid. See **user number**.

U-JIS. A mixed single- and double-byte EUC that represents CSO, CS1, and CS2 sets of ASCII and extended characters.

UNIX link. A mechanism that lets you use the **ln** command to assign more than one name to a file. Both the new name and the file being linked to must be in the same file system. A file is deleted when all the UNIX links (including the first link--the original name) have been removed. Synonym for **hard link**

update. An improvement for some part of the system.

user. The name associated with an account.

user account. See **account**.

user ID. See **user number**.

user list. A list, containing the user identification and access levels, of all operators who are allowed to use a specified file or library.

user name. A name that uniquely identifies a user to the system.

user number (Uid). A unique number identifying an operator to the system. This string of characters limits the functions and information the operator is allowed to use. The Uid can often be substituted in commands that take a user's name as an argument.

user profile. A file containing a description of user characteristics and defaults (for example, printer assignment, formats, group ID) to be conveyed to the system while the user is signed on.

user-replicated file system. One that contains files and directories accessed only by specific users or for particular applications. These user files and directories are replicated on different sites in a cluster.

utility. A service; in programming, a program that performs a common service function.

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

value. (1) In Usability Services, information selected or typed into a pop-up. (2) A set of characters or a quantity associated with a parameter or name. (3) In programming, the contents of a storage location.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with **constant**.

verify. To confirm the correctness of something.

version. Information in addition to an object's name that identifies different modification levels of the same logical object.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine. A functional simulation of a computer and its related devices.

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

virtual terminal. Any of several logical equivalents of a display station available at a single physical display station.

Volume ID (Vol ID). A series of characters recorded on the diskette used to identify the diskette to the user and to the system. Also applies to System/370 DASD devices.

wildcard. See **pattern-matching characters**.

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

work file. A file used for temporary storage of data being processed.

work station. A device at which an individual may transmit information to, or receive information from, a computer for the purpose of performing a task, for example, a display station or printer. See **programmable work station** and **dependent work station**.

working directory. See **current directory**.

wrap around. Movement of the point of reference in a file from the end of one line to the beginning of the next, or from one end of a file to the other.

D

F

M

S

X