
Cromemco[®]

Introduction to Cromix-Plus

User's Manual

Cromemco

Introduction to Cromix-Plus

User's Manual

November 1985

023-5012

Rev. C

CROMEMCO, Inc.
P.O. Box 7400
280 Bernardo Avenue
Mountain View, CA 94039

Copyright © 1984, 1985
CROMEMCO, Inc.
All Rights Reserved

This manual was produced using a Cromemco System Three computer running under the Cromemco Cromix Operating System. The text was edited with the Cromemco Cromix Screen Editor. The edited text was proofread by the Cromemco SpellMaster Program and formatted by the Cromemco Word Processing System Formatter II. Camera-ready copy was printed on a Cromemco 3355B printer.

The following are registered trademarks of Cromemco, Inc.

C-Net[®]
Cromemco[®]
Cromix[®]
FontMaster[®]
SlideMaster[®]
SpellMaster[®]
System Zero[®]
System Two[®]
System Three[®]
WriteMaster[®]

The following are trademarks of Cromemco, Inc.

C-10[™]
CalcMaster[™]
Cromix-Plus[™]
DiskMaster[™]
Maximizer[™]
System One[™]
TeleMaster[™]
System 400[™]

TABLE OF CONTENTS

Chapter 1: INTRODUCTION	1
Chapter 2: GETTING STARTED	3
Before You Begin	3
The Terminal Keyboard	3
Logging in to the Cromix-Plus System	4
Some Conventions Used in this Manual	6
Giving Some Typical Commands	6
Command Arguments	7
Command Syntax	7
Displaying the On-line Manual	8
Stopping a Program While it is Executing	9
Displaying Additional Screens of Information	9
The Terminal as a "Carriage-Return" Device	10
Error Messages	10
Changing Your Password	11
Logging Out	12
Intrinsic Commands	12
Privileged Access	13
Chapter 3: WORKING WITH TEXT FILES	15
Naming Files	15
An Additional Consideration When Naming Files	16
Special Characters	17
Creating a Sample File	17
Displaying a List of Files	18
Command Options	18
Access Privileges for the Owner of a File	21
Displaying the Contents of a File	22
Displaying a File with Page Headings and Line Numbers	22
Making a Copy of a File	23
File Links	24
Renaming a File	25
Deleting a File	25
Some Common Error Messages	26
Printing a File	27
General Rules for Command Options	29
The .bin, .com, and .cmd Filename Extensions	30
Chapter 4: CROMIX FILE STRUCTURE	33
The Home Directory	33
Visualizing the Cromix-Plus File Structure	34
Absolute Pathnames	37

How to Make Sure You Have Execute Access for a Directory	41
Displaying the Absolute Pathname of an Executable File	41
Relative Pathnames	42
Changing Directories	43
Creating a Directory	45
Moving Files to a Directory	46
How Move Works	46
Copying Files to Another Directory	46
Renaming Files with Move and Copy	47
Shortcuts for Working Within a Directory Structure	49
Deleting a Directory Structure	51
Copying a Directory Structure	53
How the Shell Looks for Executable Files	54
Special Files in the Home Directory	55
Device Files	56
Chapter 5: THE MAIL UTILITY	59
Sending Mail	59
Correcting Mistakes While Using Mail	60
What Happens to the Mail You Send	60
How Do You Know When You Have Mail	61
Reading Your Mail	61
The mbox File	62
Sending the Same Mail to Several Users	63
Chapter 6: THE CROMIX SHELL	65
The Standard Output	66
The Standard Input	66
The Sort Utility	67
Redirecting Output to a File	68
Appending Output to a File	70
Redirecting Type's Output to a File	71
Redirecting Input From a File	72
Running a Job in the Background	73
Giving Sequential Commands	74
Parentheses on the Command Line	75
Redirecting Error Messages	75
Redirection with Pipes	76
Redirecting Output to a Temporary File	77
The Tee Command	77
Filename Generation	78
Specifying a Range of Characters	80
An Important Consideration Regarding Filename Generation	81
Experimenting with Filename Generation	81

Chapter 7: WRITING COMMAND FILES	83
Command-File Description	83
A Practical Use of the Path Command	84
Redirection Within a Command File	85
The Echo Command	85
Command File Structure	86
The Goto Command	88
The If Command	89
The Shift Command	91
The Rewind Command	91
The Exit Command	91
The Input and Testing Commands	92
The Testing Command	92
The Repeat Command	94
The Find Command	95
Sample Command Files	97

LIST OF ILLUSTRATIONS

Figure 2-1: Cromemco 3102 Terminal	4
Figure 2-2: Typical "Help" Display	8
Figure 4-1: File Structure 1	34
Figure 4-2: File Structure 2	35
Figure 4-3: File Structure 3	36
Figure 4-4: File Structure 4	37
Figure 4-5: File Structure 5	39
Figure 4-6: File Structure 6	43
Figure 4-7: File Structure 7	44
Figure 4-8: File Structure 8	45
Figure 4-9: File Structure 9	48
Figure 4-10: File Structure 10	49
Figure 4-11: File Structure 11	50
Figure 4-12: File Structure 12	53

LIST OF APPENDICES

Appendix A: The Cromix-Plus Command-Line Editor	101
Making Sure the Editor is Enabled	101
Retrieving the Previous Command	102
Appendix B: The Cromemco Screen Program-Entry Editor	105
Selecting the Editor to Create a New File	105
Giving Screen Commands	106
The ESCAPE Key	106
Entering Text in the Insert Mode	106
Basic Editing	107
Moving the Cursor While the Editor is in the Command Mode	107
Delete Mode	108
Xchange (Exchange) Mode	109
Editing a File that is Stored on the Disk	111
Setting Markers with the At Command	111
The Screen Editor's Shell Command	112
Considerations When Using the Editor	112
Command Summary	114
INDEX	121

Chapter 1

INTRODUCTION

The Cromemco Cromix-Plus Operating System is a program that supervises the operation of the computer and its resources, disk drives, printers, terminals, modems, and so on. All computers, even the personal computer you may have at home, require some kind of operating system.

The Cromix-Plus system stores information in files--each containing some related information, and each having a distinct filename. (Physically, these files are located on some storage medium, such as the computer's hard disk.) To ensure that files are easy to access, the Cromix-Plus system organizes all files in directories within a hierarchical file structure (chapter 4).

Password security and a system of access privileges protect files from unauthorized access. If desired, access privileges can be changed to authorize access by selected users.

At any given time, a personal computer can normally do only one job for one user. A Cromemco microcomputer, on the other hand, can be working on numerous jobs for many users. The Cromix-Plus system is a multiuser system. While the computer compiles a program for one user, it may be printing a file for another. From the viewpoint of the Cromix-Plus user, the computer is dedicated to his or her special needs.

Cromix-Plus is also a multitasking operating system. To illustrate, suppose you need to print 10 copies of a lengthy report, and you also need to edit an important file. By printing the report as a "background" task, you can free your terminal for other work, such as editing that file. This is "multitasking," and it means you won't waste time waiting for one job to finish so you can start another.

With some multi-user systems, as the number of users and tasks increase, it takes longer and longer to process individual jobs. Such degradation in performance can be a real problem. With the Cromix-Plus Operating System, degradation is minimal because the Cromix-Plus system takes full advantage of the Cromemco computer's 68000 microprocessor.

Supplied with every Cromix-Plus system is a set of utility programs. These utilities handle the jobs users need done again and again, such as printing a copy of a file (the Spool utility) or giving a file a new filename (the Rename utility). The most versatile of Cromix-Plus utility programs is the Shell utility. The Shell provides the interface between the Cromix-Plus Operating System and its users. It is the Cromix-Plus command interpreter and command processor. The Shell is also programmable, which means you can instruct it to process commands

Cromemco Introduction to Cromix-Plus

1. Introduction

in special ways. Such customized commands can be given from the command line (chapter 6) or from command files (chapter 7).

The Cromix-Plus Operating System is as powerful as many operating systems designed to run on large, mainframe computers. Its capabilities include:

- Device-compatible I/O, to support redirection of input and output.
- Date and time support.
- Numerous file buffers for high-speed execution.
- Resident execution of tasks (jobs are not swapped out to disk).
- RAM-disk feature, providing extra RAM for use as a high-speed disk.

Chapter 2

GETTING STARTED

In this chapter, you'll start using the computer and give some simple commands to the operating system. In the process, you'll become familiar with your terminal keyboard and some of the conventions used in this manual.

If you've used a computer with a multi-user operating system before, much of the information in this chapter will seem familiar. Be sure to at least skim through the chapter before continuing.

Before You Begin

Talk to the System Administrator--the person in charge of your Cromix-Plus System. The System Administrator will create a user account and a directory for you.

The System Administrator will also help you choose a login name and password. Your login name identifies you to the operating system--your password ensures that only you can log in with your login name.

When you have received a login name and password, you can log in to the system. In other words, you can tell the Cromix-Plus system you wish to use the computer, and the system will grant your request.

The Terminal Keyboard

Figure 2-1, a Cromemco 3102 terminal keyboard, points out some important keys.

Cromemco Introduction to Cromix-Plus

2. Getting Started

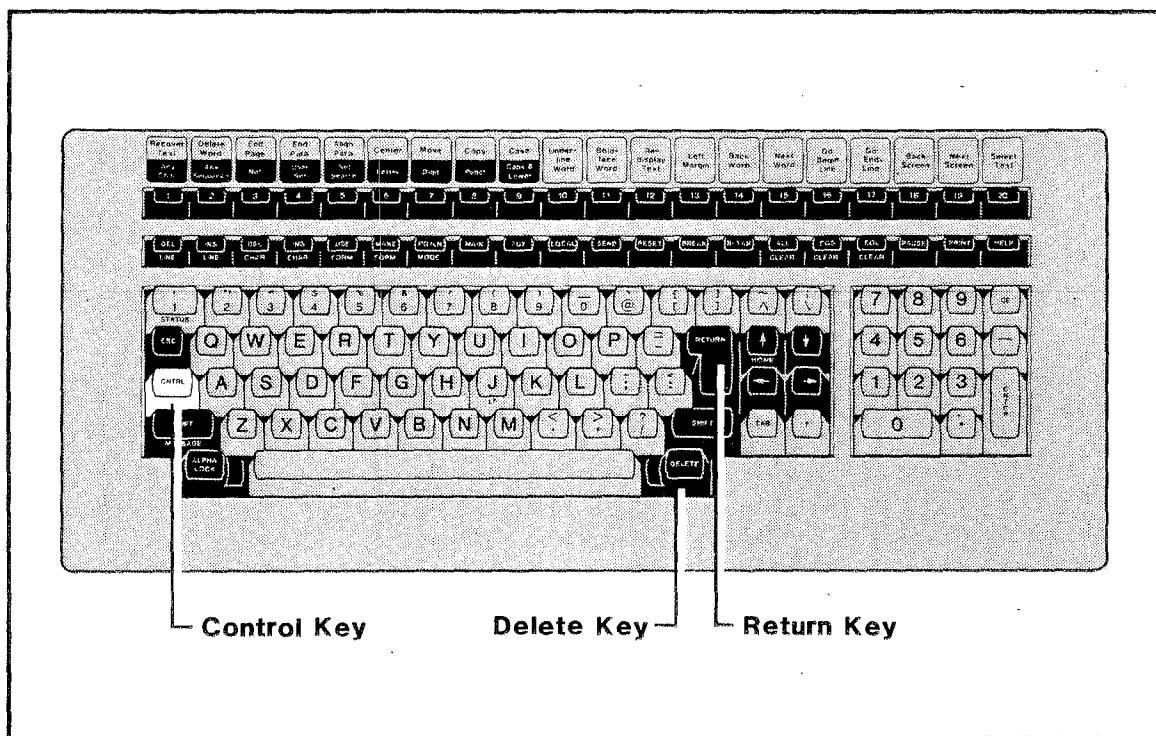


Figure 2-1: CROMEMCO 3102 TERMINAL

The RETURN key is an end-line key. After you've typed an instruction to the operating system, pressing RETURN ends the typed line and enters that instruction. As you're typing text from the keyboard, a RETURN ends the current line so you can begin a new line.

The DELETE key deletes characters on the current line. (DELETE is not an "abort," or program-interrupt, key as it is with some other operating systems.)

The CONTROL key is discussed later in the chapter.

Logging in to the Cromix-Plus System

Many terminals may be connected to a multi-user Cromix-Plus system, and the system keeps track of what terminals are in use. When a terminal is not in use, the system displays the prompt "Login:". This prompt means that you (or any system user) can log in to the Cromix-Plus system from that terminal.

You should see "Login:" on the terminal now. If you do not, make sure the terminal is turned on (there is an ON/OFF switch at the rear of the terminal). Then press the RETURN key on the terminal keyboard a few times.

Cromemco Introduction to Cromix-Plus
2. Getting Started

With the cursor next to "Login:", type your login name. Because the Cromix-Plus system is not sensitive to letter case, you can use any combination of upper- and lowercase letters. After typing the last letter of your login name, press the RETURN key. In response, the system displays a password prompt, as shown here:

```
LOGIN: jim
Password:
```

To protect your password, you will not see what you type when you answer this prompt. (Nor will the cursor move.) After typing the last letter of your password, press RETURN.

Note: If you think you mistyped your password, just press RETURN. The system will give you a new login prompt, so you can start over.

If you correctly typed both your login name and password, your terminal will look similar to this:

```
LOGIN: jim
Password:
```

```
Logged in jim Nov-14-1984 09:53:47 on qtty3
Message of the day: Welcome to the Cromix-Plus Operating System
%
```

Each time you log in, the Cromix-Plus system displays the message of the day. It announces that you are logged in to the system. Of greater importance, however, is the Shell prompt--the percent sign to the left of the cursor.

For users with "privileged access" to the system, the prompt is a pound sign (#) instead of a percent sign (%). Privileged access is briefly discussed at the end of this chapter.

Because the Shell is your means of communicating with the operating system, you will see the prompt again and again. With the prompt displayed, you can give a command to the Shell.

A command is a typed instruction, which you send to the Shell by pressing RETURN. Until you press RETURN, you can correct or change any command. Use the DELETE key to erase characters; then retype the command. If you prefer, erase the entire command line by pressing CONTROL-U (hold down the CONTROL key and type "U"). Under the Cromix-Plus system, CONTROL-U is a "line-kill" key.

Appendix A, "The Cromix-Plus Command-Line Editor," explains some other ways to correct or change a command line.

Some Conventions Used in this Manual

In text, command names are capitalized, as are the names of corresponding utility programs.

Example:

```
Time command
Time utility
```

For each command, there is a sample command line.

```
% time
```

To give the command, type what's printed in **boldfaced** type (using any combination of upper- and lowercase letters), and press the RETURN key. Pressing RETURN sends the command to the system. Under the Cromix-Plus system, RETURN is an "enter" key.

Almost all the commands in this manual can be given by any user--privileged or nonprivileged. Thus, a percent sign (%) represents the Shell prompt in most examples. (The only exceptions are two commands in chapter 7.) Privileged access is discussed at the end of this chapter.

Giving Some Typical Commands

Because the Cromix-Plus System is a multi-user system, it is often helpful to know who is currently logged in. You can find out by giving the Who command:

```
% who
```

When you press RETURN to enter the command, the Shell calls the Who utility program. ("Calling" a program starts it running.) In response, Who executes, displaying a list of users on the terminal screen. A typical display is shown here:

```
% who
betty      qtty1      Nov-14-1984 07:16:22      0      0
jim        qtty2      Nov-14-1984 09:42:29      0      0
fred       qtty3      Nov-14-1984 09:53:47      0      0
%
```

This list of login names and terminal ("qtty") numbers tells you who is currently using the system, from what terminal. Also shown is the time each user logged in to the system.

Cromemco Introduction to Cromix-Plus

2. Getting Started

Note: You can log in from any unused terminal--even if you are already logged in somewhere else. Although the Cromix-Plus System keeps track of the terminal (or terminals) you are using, it does not associate you with a specific terminal.

When a program such as Who is through executing, it returns control to the Shell. The Shell then displays a new prompt so you can give another command. To demonstrate, give the Time command.

```
% time
Wednesday, November 14, 1984          9:58:12
%
```

In response, Time displays today's date and the current time on your terminal screen.

Command Arguments

By typing additional items on a command line, you can modify what a command does. These additional items are called command arguments.

To demonstrate, give this variation of the Who command.

```
% who am i
```

This time, Who displays information about you only (your login name, your terminal, and the time you logged in). A typical display is shown here:

```
% who am i
jim      qtty3      Nov-14-1984 09:53:47      0      0
%
```

Many commands can use arguments. For some commands, arguments are required. For others (such as Who), arguments are optional.

Command Syntax

The required form of a command--how the command and its arguments are grouped on the command line--is called command syntax. When representing command syntax, the Cromix-Plus documentation shows optional arguments in square brackets.

For example, the syntax of the Who command can be represented as follows:

```
who [am i]
```

The command arguments (in brackets) are optional.

A basic understanding of command syntax will help you use the system's on-line manual. There are additional examples of command syntax throughout this manual.

Displaying the On-line Manual

The Help utility displays "pages" from the system's on-line manual. To display information about a particular command, give the Help command with an argument.

The syntax of the Help command is:

```
help [command-name]
```

When you give the command, the argument is the name of any command (such as Who, Time, or Help).

Figure 2-2 shows a typical display.

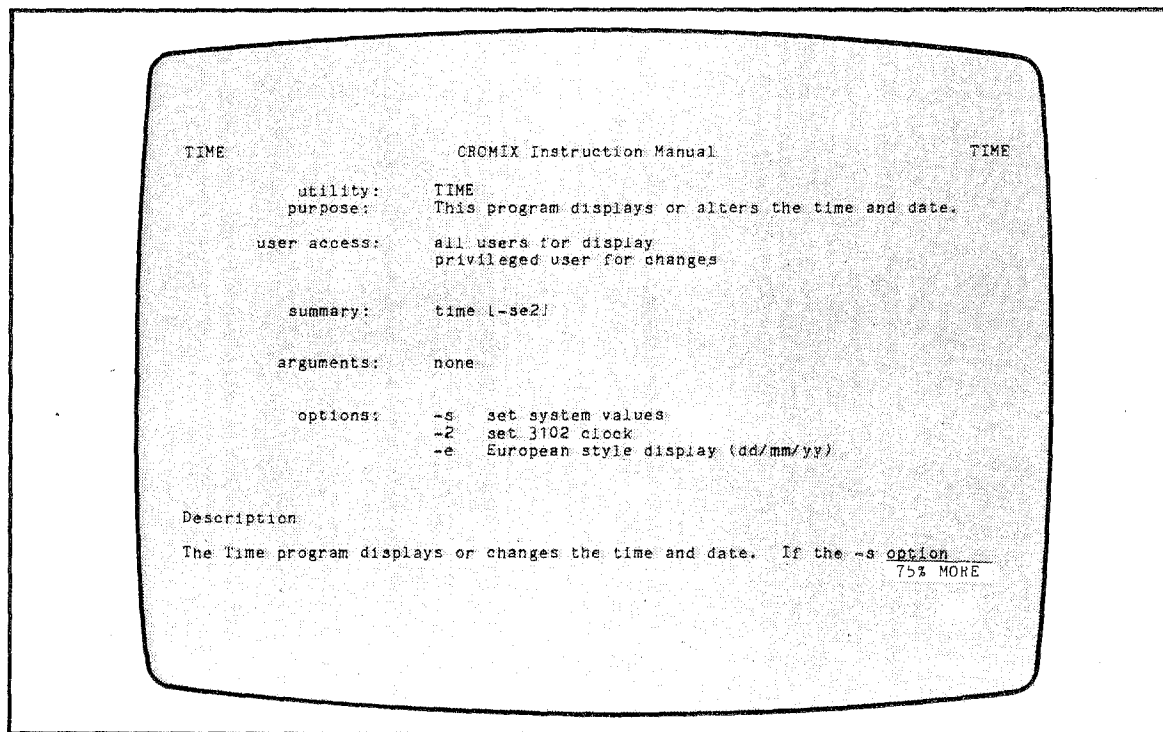


Figure 2-2: TYPICAL "HELP" DISPLAY

"75% MORE" at the bottom righthand corner of the screen indicates there is more information about this particular command.

Pressing RETURN or the SPACE bar brings additional information onto the screen. Pressing the RETURN key brings a new line into view, while pressing the SPACE bar displays an entire screen of new text.

When you're through using Help, type **q** for **quit**. In a few seconds, the Shell will display a new prompt so you can give another command. While a program like Help is running, you can also get a new prompt by pressing CONTROL-C, as explained in the next section.

Note: Other than CONTROL-C, the remaining keys and/or commands that work with Help (such as pressing RETURN or typing **q**) are specific to Help. Help, like a few other programs, has its own set of commands.

Stopping a Program While it is Executing

Almost all the commands in this manual execute utility programs like Who, Time, or Help. You can stop most programs while they are executing by pressing CONTROL-C (hold down the CONTROL key and type "C").

Pressing CONTROL-C sends an "abort" signal (via the Shell) to the program. In response, the program prematurely returns control to the Shell. In other words, the program stops running, and the Shell displays a new prompt.

Displaying Additional Screens of Information

Many programs display output on the terminal screen. Sometimes that output exceeds the capacity of the terminal screen (24 lines). For example, the Query utility displays a summary of Cromix-Plus commands that is longer than 24 lines.

So you can view a program's output at your leisure, the display stops after each screenful. To demonstrate, give the Query command:

```
% query
```

After displaying one screenful of text, your terminal beeps. The beep tells you there is more output. To view the next 24 lines of output, press CONTROL-Q (hold down the CONTROL key and type "Q"). Each time you press CONTROL-Q, you'll see a new screen of text. After displaying the final screen of output, Query returns control to the Shell.

This is how your terminal normally displays output. Occasionally, your terminal will display a program's output without stopping the display after one screen.

Cromemco Introduction to Cromix-Plus

2. Getting Started

If this happens, you can restore the normal mode of operation by giving the following command:

```
% mode pa
```

This command resets one of your terminal's operating characteristics. The command argument, **pa** (for pause), causes your terminal to, once again, pause after each screen of output.

There is additional information about the Mode utility in the Cromix-Plus User's Reference Manual.

The Terminal as a "Carriage-Return" Device

Another of your terminal's operating characteristics is to move the cursor down a line each time you press the RETURN key. This enables you to give commands to the Shell.

Occasionally, without any action on your part, the terminal may not work in this manner. You will press RETURN, and the cursor will not move down a line. To restore the terminal to its normal mode of operation, give the following command:

```
% mode crdev
```

To enter the command, you will have to press the DOWN ARROW key. (Pressing RETURN does nothing.) The command argument, **crdev**, re-establishes your terminal as a carriage-return device. After you give the command, pressing RETURN will again move the cursor down a line.

There is additional information about the Mode utility in the Cromix-Plus User's Reference Manual.

Error Messages

From time to time, all users receive "error messages" from the system. The Cromix-Plus system displays error messages on the terminal screen. Some of these messages come from the Shell; others come from the individual programs.

A common error message is "Command not found"--the message the Shell displays when you give a non-existent command. For example, under the Cromix-Plus system, there is no More utility. Thus, the following command produces an error message:

```
% more  
Command not found: "more"
```

If you ever mistype a command name, you'll see a similar message when you press RETURN. It means the Shell could not carry out your command. (The Shell looked for, but could not locate, the program.)

Some programs generate error messages of their own. Mode is an example. Although many programs ignore bad arguments, Mode displays an error message:

```
% mode who
Illegal argument: "who"
```

Some common error messages associated with the Cromix-Plus utility programs are given throughout this manual.

Changing Your Password

For security, you should occasionally change your password. Whenever you wish to change your password, give the Passwd command:

```
% passwd
```

In response, the Passwd utility prompts for your login name:

```
% passwd
Name:
```

When you type your login name and press RETURN, Passwd displays another prompt. In the example, a user named Jim is changing his password:

```
% passwd
Name: jim
Password:
```

To answer this prompt, type your new password and press RETURN. When you press RETURN, Passwd displays an "encrypted" version of your new password (not what you actually typed).

Cromemco Introduction to Cromix-Plus

2. Getting Started

To respond to the next prompt ("Name:"), simply press RETURN:

```
% passwd
Name: jim
Password: xhbydksy

Name: (RETURN)
%
```

After changing your password in this manner, you must use your new password the next time you log in. (The system no longer recognizes your old password.) If, after several tries, you cannot log in using your new password, see the System Administrator.

Logging Out

When you are through using the computer, log out by giving the Exit command. Exit, like many commands, can be abbreviated on a command line. To give the Exit command, you can either type the command name in full or a just a portion of it:

```
% ex
```

Although the sample command lines in this manual show the abbreviated form for all commands, you can often substitute the full command name (**exit** instead of **ex**).

When you log out, the system redisplay its login prompt. To use the system again from this terminal, you must repeat the login procedure.

Intrinsic Commands

Exit is an example of an "intrinsic" command. In other words, it is "intrinsic" to the Cromix-Plus System--it is not a utility program like Who or Passwd, which the Shell calls from the computer's hard disk each time you use the program.

Whether the first item on a command line is the name of an intrinsic command (like Exit) or a program (like Who) matters only to the system. The distinction is mentioned to help you understand the Cromix-Plus documentation. For example, the User's Reference Manual summarizes the Shell commands and Cromix-Plus utility programs. "Shell command" means an intrinsic command such as Exit.

Privileged Access

If your Shell prompt is a percent sign, you are a nonprivileged user. As such, there are certain commands you cannot give. For example, you cannot use the Passwd utility to change someone else's password--that requires privileged access.

```
% passwd
```

```
Name: cindy
```

```
Can only change your own password.
```

```
%
```

You can, of course, change your own password.

As a nonprivileged user, you cannot give a command that might harm the system or inconvenience other users. So, as you use the Cromix-Plus Operating System, feel free to experiment with new commands.

If you accidentally give a command requiring privileged access, the Shell will display an error message, as in the following example:

```
% passwd -n
```

```
Must be privileged user.
```

```
%
```

Most commands can be given by all users--privileged and nonprivileged.

Chapter 3

WORKING WITH TEXT FILES

In this chapter you'll learn the basic utilities and Shell commands that manipulate files. First, you'll create a sample file using the Cromemco Screen program-entry editor. (The Screen editor is discussed in detail in appendix B.) Using this sample file, you can experiment with some of the Cromix-Plus system's most used commands--such as Rename (the command that gives a file a new filename) and Delete (the command that deletes a file from the hard disk).

Also covered is the Spool utility, which prints one or more files. Instructions for stopping printing are included.

Naming Files

In the next section, you'll create your first file. You can name this file **sample**, as suggested in the text, or select another filename. (In the Cromix-Plus documentation, filenames are always printed in **boldfaced**, lowercase type.) Filenames consist of one to twenty-four consecutive characters (in other words, no SPACES) from the following set:

a-z, A-Z, 0-9, _, \$, .

Because the Cromix-Plus system is not sensitive to letter case, the following filenames are equivalent:

letters
Letters
LETTERS

You can use the underscore character (_) to make filenames easier to read, as in the following examples:

form_letter
chapter_1

A period in a filename is the first character of a filename extension, as in the following examples:

```
file.text  
sort.out
```

In these filenames, the extensions are **.text** and **.out**. Filename extensions are useful for identifying similar kinds of files:

```
report.save  
chapter2.save  
memo.save
```

To illustrate, you would think twice before deleting a file with a **.save** extension. You might reserve this filename extension for all your important files.

If a filename contains several periods, only the final period (and the characters that follow) is a filename extension.

For example:

```
file.text.out  
sort.out.save
```

In these filenames, the extensions are **.out** and **.save**. As explained at the end of this chapter, the files you create may not have a **.bin** or **.com** filename extension.

Although you can use a **.bak** (for backup) extension when naming files, its use may prove confusing. The Screen program, introduced in this chapter, automatically creates a backup file with a **.bak** filename extension whenever you update an existing file. For details, refer to appendix B.

An Additional Consideration When Naming Files

Any Cromix-Plus utility that uses filenames will accept a 24-character filename. Remember, however, that not all the programs you may use are Cromix-Plus utilities.

There are a number of programs that can be installed on the Cromix-Plus system that may have their own file-naming conventions. An example is the Cromemco **Formatter-II** text-formatting program. **Formatter-II** is not a Cromix-Plus utility program and will not accept a 24-character filename. (Its limit is 12 characters, including a 3-character filename extension.)

When in doubt, consult the documentation for the program you're using.

Special Characters

Each of the following characters means something special to the Shell, and should not be used when creating files:

* ? [] > < | " ' { } - & ^ \

In choosing a filename, restrict yourself to those characters shown earlier in the chapter.

Creating a Sample File

In this section, you'll create a sample file. This manual assumes you'll be using the Screen program-entry editor to create the file. Screen is another of the Cromix-Plus utility programs, available to all Cromix-Plus users.

In preparation, read the following sections in appendix B, where the Screen editor is discussed in detail:

- "Selecting the Editor to Create a New File"
- "Giving Screen Commands"
- "The ESCAPE Key"
- "Entering Text in the Insert Mode"
- "Saving a File on Disk--the Exit Command"

Using the Screen editor, create a file named **sample**, and type the following text in the file:

```
Help
Passwd
Query
Who
Exit
```

Then save the file.

Because the Screen editor has its own set of commands, pressing CONTROL-C does not exit you from the Screen program. If you press CONTROL-C with the editor in the Command mode, your terminal beeps. Pressing CONTROL-C with the editor in the Insert mode adds an unwanted character (^C) to your text.

Displaying a List of Files

The Ls (for list) utility displays useful information about files. Without an argument, Ls displays a simple list of files. To better illustrate the display, the following example shows some typical filenames; obviously, you'll see a different display when you give the command:

```
% ls
letter      memo      output    plan6_10
```

Command Options

Command options modify what commands do. On a command line, options follow the command name. They consist of a single alphabetic character, preceded by a hyphen. In the following example, `-m` (a hyphen, followed by the letter "m") is a command option:

```
% ls -m
239  1 letter
241  1 memo
 50  1 output
240  1 plan6_10
```

The `-m` (for medium) option instructs Ls to display more detailed information about files. From left to right, the display shows you the size of the file (in characters), the number of links to the file, and the filename. (Links are discussed later in this chapter.)

Another option, `-l` (for long), displays even more information:

```
% ls -l
239  1 rewa re-- re-- betty      Nov-14 10:18 letter
241  1 rewa re-- re-- betty      Nov-14 10:18 memo
 50  1 rewa re-- re-- betty      Nov-14 10:22 output
240  1 rewa re-- re-- betty      Nov-14 10:21 plan6_10
```

Now you can check the time you last worked on a file. You can also see the access privileges associated with each file. (Access privileges and file ownership are discussed in the next section.)

To display still more detailed information about files, give the Ls command with the `-e` (for everything) option. A typical display is shown here:

```
% ls -e
letter                239
    created:  Nov-14-1984 10:17:16  rewa re-- re--
    modified: Nov-14-1984 10:18:08  betty          group: 1
    accessed:  Nov-14-1984 10:17:23  links: 1
    dumped:    000-00-1900 00:00:00  inode: 423

memo                  241
    created:  Nov-14-1984 10:18:18  rewa re-- re--
    modified: Nov-14-1984 10:18:55  betty          group: 1
    accessed:  Nov-14-1984 10:18:18  links: 1
    dumped:    000-00-1900 00:00:00  inode: 421

output                50
    created:  Nov-14-1984 10:21:55  rewa re-- re--
    modified: Nov-14-1984 10:22:14  betty          group: 1
    accessed:  Nov-14-1984 10:22:16  links: 1
    dumped:    000-00-1900 00:00:00  inode: 402

plan6_10              240
    created:  Nov-14-1984 10:19:58  rewa re-- re--
    modified: Nov-14-1984 10:21:04  betty          group: 1
    accessed:  Nov-14-1984 10:20:30  links: 1
    dumped:    000-00-1900 00:00:00  inode: 422
```

Included in the display is the inode number assigned to each file. (The system uses inode numbers to locate files on the computer's hard disk.) Ls with the `-i` (for inode) option displays only filenames and inode numbers. A System Administrator might give this command to determine the inode number of a particular file:

```
% ls -i
423 letter
421 memo
402 output
422 plan6_10
```

By adding options to a basic command ("Ls"), one utility serves the needs of many--ordinary users, who need to verify filenames, or System Administrators, who need to check inode numbers. Ls is one of many Cromix-Plus utilities that use options.

In command syntax, options are written as in the following example:

```
ls [-eilm]
```

Cromemco Introduction to Cromix-Plus
3. Working with Text Files

Ls, like a number of Cromix-Plus utilities, will accept multiple options:

```
% ls -e -m
letter                               239
  created: Nov-14-1984 10:17:16 rewa re-- re--
  modified: Nov-14-1984 10:18:08 betty          group: 1
  accessed: Nov-14-1984 10:17:23 links: 1
  dumped:   000-00-1900 00:00:00 inode: 423

memo                                  241
  created: Nov-14-1984 10:18:18 rewa re-- re--
  modified: Nov-14-1984 10:18:55 betty          group: 1
  accessed: Nov-14-1984 10:18:18 links: 1
  dumped:   000-00-1900 00:00:00 inode: 421

output                                50
  created: Nov-14-1984 10:21:55 rewa re-- re--
  modified: Nov-14-1984 10:22:14 betty          group: 1
  accessed: Nov-14-1984 10:22:16 links: 1
  dumped:   000-00-1900 00:00:00 inode: 402

plan6_10                              240
  created: Nov-14-1984 10:19:58 rewa re-- re--
  modified: Nov-14-1984 10:21:04 betty          group: 1
  accessed: Nov-14-1984 10:20:30 links: 1
  dumped:   000-00-1900 00:00:00 inode: 422
```

When you give Ls multiple options, the one producing the most extensive list (in this case, **-e**) prevails. If Ls cannot recognize one or more options, a command-syntax summary is displayed so you can see what options are available.

When using a utility that accepts multiple options, you can often group options after a single hyphen, as in this example:

```
% ls -em
letter                               239
  created: Nov-14-1984 10:17:16 rewa re-- re--
  modified: Nov-14-1984 10:18:08 betty          group: 1
  accessed: Nov-14-1984 10:17:23 links: 1
  dumped:   000-00-1900 00:00:00 inode: 423

memo                                  241
  created: Nov-14-1984 10:18:18 rewa re-- re--
  modified: Nov-14-1984 10:18:55 betty          group: 1
  accessed: Nov-14-1984 10:18:18 links: 1
  dumped:   000-00-1900 00:00:00 inode: 421
```

```
output                                50
  created:  Nov-14-1984 10:21:55  rewa re-- re--
  modified: Nov-14-1984 10:22:14  betty          group: 1
  accessed:  Nov-14-1984 10:22:16  links: 1
  dumped:    000-00-1900 00:00:00  inode: 402

plan6_10                               240
  created:  Nov-14-1984 10:19:58  rewa re-- re--
  modified: Nov-14-1984 10:21:04  betty          group: 1
  accessed:  Nov-14-1984 10:20:30  links: 1
  dumped:    000-00-1900 00:00:00  inode: 422
```

As a general rule, you can group options that do not require arguments. All Ls command options fit this description because none requires additional information on the command line for Ls to act on that option.

General rules regarding command options are summarized at the end of this chapter.

Access Privileges for the Owner of a File

When you gave the "ls -l" command, you saw a string of lowercase letters and hyphens preceding your login name:

```
% ls -l
  32  1 rewa re-- re-- you          Nov-14 10:14 sample
```

Your login name means you are the owner of the file **sample**. You "own" it in the sense that the Cromix-Plus system created the file for you.

What you can do with the file depends on the access privileges associated with the file. Your privileges (as the owner of a file) are defined by the first four characters in the string of lowercase letters and hyphens. As supplied, the Cromix-Plus system automatically gives you the following access to your files:

rewa

This group of letters:

```
r for "read"
e for "execute"
w for "write"
a for "append"
```

means you have all possible access privileges. You can read from the file, write to the file, append (add to the end) of the file, or execute the file. Read, write, and append access ensure you can display, change, and add text to files at will. Execute access becomes important when you create command files, as explained in chapter 7.

Having these privileges ensures you can work with your own files. For more information about access privileges, refer to the discussion of the Access utility in the Cromix-Plus User's Reference Manual.

Displaying the Contents of a File

The Type command displays the contents of text files on the terminal screen. (Any file you can edit using the Screen program is a text file.)

To display a file, the command syntax is:

```
ty filename
```

For example, you can display the contents of the **sample** file by giving this command:

```
% ty sample  
Help  
Passwd  
Query  
Who  
Exit  
%
```

Type shows you the text in a file. When files are long, pressing CONTROL-Q brings more text onto the terminal screen. Type only displays files--to change the contents of a text file, you must use an editor, such as the Screen program.

Displaying a File with Page Headings and Line Numbers

The Clist utility, like Type, displays the contents of text files. Clist, however, adds some information of its own to the display. In Clist's display, each line of the file is numbered for easy reference. At the top of the display is a heading, showing the name of the file and the last time you worked on it:

```
% clist sample  
File  SAMPLE                               Wednesday, November 14, 1984   10:14:29  
  
1      Help  
2      Passwd  
3      Query  
4      Who  
5      Exit
```

3. Working with Text Files

If you create text files that contain your own computer programs, you may wish to display those files with Clist. For text files containing letters, memos, and so on, use Type instead.

Neither Type nor Clist change the files they display in any way.

Making a Copy of a File

When you need another copy of an existing file, use the Copy utility. After a copy operation, you have two files--the original and the copy.

The two files are identical in every respect--except for their filenames. The Cromix-Plus system will not allow two files to have the same name.

When you wish to make a copy of a file, the command syntax is:

```
copy source-file destination-file
```

The term "source file" refers to the original file (the source of Copy's input). The term "destination file" refers to the new file. In the Cromix-Plus documentation, and in the on-line help files, you will see these terms often.

Make a copy of the **sample** file now. Call the new file **alpha**.

```
% copy sample alpha
```

The Ls command verifies the presence of the new file.

```
% ls  
alpha    sample
```

For practice, make a copy of the **alpha** file named **beta**.

```
% copy alpha beta
```

Cromemco Introduction to Cromix-Plus
3. Working with Text Files

Using Copy, you created two files that (except for their names) are identical to the file **sample**:

```
% ty alpha
Help
Passwd
Query
Who
Exit
% ty beta
Help
Passwd
Query
Who
Exit
%
```

Giving the "ls -l" command will show that even the creation times associated with the files are the same. If the system created the **sample** file on December 12, at 12:02, the **alpha** and **beta** files will show "Dec-12 12:02" as their creation times, too. Only the inode numbers associated with each file will be different.

File Links

The Cromix-Plus system identifies files by associating each filename with an inode number. Each inode number corresponds to a physical location on the computer's hard disk. When you create a file, you create both the file itself (under an inode number) and an identifying filename. The filename is also called a link, and every inode must have at least one link. The "ls -m" command

```
% ls -m
32      1 sample
```

tells you how many links an inode has. In the above example, the "1" indicates that the filename "sample" is the only link to that inode (use the **ls -li** command to get the inode number).

Multiple links are particularly useful for files that have long or awkward pathnames (pathnames are discussed in the next chapter). With the Maklink utility (refer to the Cromix-Plus User's Manual), you can make a link from the original file to a simpler filename in the current directory, and delete it later when you no longer need it. As long as an inode has more than one link, deleting one of the links does not delete the file.

Renaming a File

The Rename command gives an existing file a new filename.

Command syntax:

```
ren old-filename new-filename
```

To demonstrate, give the **sample** file a new filename:

```
% ren sample practice
```

The Ls command verifies the file has been renamed:

```
% ls  
alpha beta practice
```

Deleting a File

The Delete command deletes files. Once a file is deleted, it no longer appears in the list of files that Ls displays. Nor can you access the information in the file (unless that particular inode had more than one link).

Command syntax:

```
del file-list
```

For "file-list," you can substitute a series of filenames. Or, you can delete just one file by giving the command with a single argument (one filename). To demonstrate, delete the file **beta** by giving the following command:

```
% del beta
```

You know the file is gone when you see the new Shell prompt.

You can delete several files by giving the command with several arguments (a series of filenames). For example, if you had files named **text**, **demo**, and **chapter**, the following command would delete those files:

```
% del text demo chapter
```

Once you delete a file, its contents are irretrievably gone. For this reason, it is a good idea to check the contents of a file using Type before deleting that file.

Some Common Error Messages

When using Copy, Delete, or Rename, a new Shell prompt means the system carried out your commands.

```
% copy myfile yourfile  
% del myfile  
%
```

You know the system did not execute a particular command if an error message precedes the new prompt.

```
% ren myfile ourfile  
File not found: "myfile"  
%
```

This particular message means Rename could not locate the old file (the file to be renamed). Rename returned control to the Shell without renaming a file. To produce a similar error message, you might try renaming the **beta** file--the file you deleted in the previous section.

Some common error messages associated with Rename and Copy are:

```
% ren today  
Wrong number of arguments  
%
```

This is an example of a syntax error--Rename requires two arguments. Thus, Rename returned control to the Shell without even looking for a file named **today**.

```
% copy letter letter.save  
File already exists: "letter.save"  
%
```

As a safeguard, Copy and Rename normally require a new filename as their final command argument. Without this protection, you could easily destroy the contents of an existing file.

If you do want Copy to overwrite an existing file, use the **-f** (for force) option:

```
% copy -f letter letter.save  
%
```

Printing a File

When you want to print a copy of a file, use the Spool utility.

To print files, the command syntax is:

```
spool file-list
```

When you give Spool a filename as an argument, Spool sends a copy of that file to the system printer. To demonstrate, print a copy of the **practice** file by giving this command:

```
% spool practice  
%
```

The new Shell prompt tells you Spool found the **practice** file and sent a copy of the file to the system printer. If your printer is turned on, and properly loaded with paper, your file will be printed.

Like Delete and Rename, Spool displays an error message when it cannot locate a particular file.

```
% spool pactice  
File not found: "pactice"  
%
```

Sometimes, when you give the Spool command, another user's job will be printing. In this case, Spool will add your job to a queue--to be printed in its turn.

To print more than one copy of a file, give the Spool command with the **-m** (multiple-copy) option. Using this option, the following command instructs Spool to print three copies of the **practice** file.

```
% spool -m 3 practice
```

The **-m** option requires an argument--the number of copies to print. The option and its argument must be separated by spaces, as shown. When you do not use the **-m** option to specify multiple copies, Spool, by default, prints one copy of a file.

When called without a filename as an argument, Spool can be used with still more options. Two of them, **-l** and **-k**, are especially useful. The **-l** (for list) option displays a list of your own print jobs, whether a job is currently printing or in the queue.

```
% spool -l  
Filename           User      Seq  Dev  Pri  Pages  Lines  Copies  Form  
-> memo            betty     3    5:5  5    1      12     1       0
```

Cromemco Introduction to Cromix-Plus

3. Working with Text Files

By adding an option, **-a** (for all), to the command, you can display a list of all print jobs (other users' jobs, as well as your own):

```
% spool -la
  Filename          User      Seq  Dev  Pri  Pages  Lines  Copies  Form
-> memo             betty     3   5:5  5    1     12     1      0
  first_draft      jim       4   5:5  5   19    1498     1      0
  budget           jim       5   5:5  5    1     47     2      0
```

Spool displays an arrow, pointing to the current job (the one that is printing). Your job's Sequence number is a reference if you need to cancel that particular job. Pressing CONTROL-C after giving the Spool command does not stop printing.

You can use Spool with the **-k** (for kill) option to cancel any of your own print jobs. To kill a job, while it prints or as it waits in the queue, the Spool command syntax is:

```
spool -k sequence-#-or-filename
```

To illustrate, consider the following list of jobs:

```
% spool -la
  Filename          User      Seq  Dev  Pri  Pages  Lines  Copies  Form
-> memo             betty     3   5:5  5    1     12     1      0
  first_draft      jim       4   5:5  5   19    1498     1      0
  budget           jim       5   5:5  5    1     47     2      0
```

If Jim wanted to cancel the two jobs he has in the queue, he might give the following commands:

```
% spool -k 4
% spool -k budget
```

The first command kills a job by referring to its sequence number, the second, by referring to a filename. Use whichever method you prefer. Be aware, however, that killing a job by sequence number is more precise than killing a job by filename. To illustrate, if Jim had two files named **budget** in the queue, the command "spool -k budget" would kill both of them.

As a nonprivileged user, you can only kill your own print jobs. If Jim tries to kill Betty's job, an error message results:

```
% spool -k 3
3          not found
```

The Printer Daemon -- Although it's easiest to call Spool a utility that "prints a file" or "sends a copy of a file to the printer," Spool is a little more complex than this. Printing is a multi-step process.

The Spool utility is the process that initiates printing, but it is another process--called the printer daemon--that actually prints your files. Spool activates the daemon before returning control to the Shell.

At your terminal, you see a new Shell prompt, while (behind the scenes) the daemon prints your file.

General Rules for Command Options

As you use the Cromix-Plus system, you'll encounter numerous programs that can be used with options. The following information will help you use these programs effectively.

1. Command options consist of a single character, immediately preceded by a hyphen:

```
% ls -m
```

As shown, a space separates the command name and the command option.

2. You can specify more than one option on a command line:

```
% spool -l -a
```

3. You can group options that do not require arguments after a single hyphen:

```
% spool -la
```

4. You cannot group options that require arguments:

```
% spool -m 3 practice
```

In the preceding command, **-m** requires an argument (the number of copies to print). As shown, a space must separate the option and its argument.

5. On a command line, options precede other arguments:

```
% spool -m 3 practice
```

Not

```
% spool practice -m 3
```

When the Shell calls a program, it passes all arguments (including command options) to the program--without checking their validity. Cromix-Plus programs do not ignore invalid command options. If you specify one or more invalid options, your command will not be executed. Instead, the program will display a command-syntax summary and return control to the Shell.

The .bin, .com, and .cmd Filename Extensions

Three filename extensions have special significance to the Shell--**.bin**, **.cmd**, and **.com**.

.bin and **.com** mean the file is an executable binary file. For example, the Cromix-Plus utility programs are stored in files with **.bin** extensions. (Files with **.com** extensions were originally designed to work with Cromemco's CDOS Operating System.)

You cannot use the Screen program--entry editor (or another text editor) to edit the contents of a binary file.

```
% screen sample.bin
Cromix Screen Editor version xx.xx
Illegal filename: "sample.bin"
```

```
% screen sample.com
Cromix Screen Editor version xx.xx
Illegal filename: "sample.com"
```

Nor should you display the contents of a binary file using Type or Clist. If you do, your terminal may not work properly until you turn it OFF and back ON.

.cmd means the file is a command file. Command files are text files, like the file **practice**. (A text file is any file you can edit with the Screen program or display with Type or Clist.) The text in a command file is a series of ordinary commands (like the commands you gave in this chapter).

For example, the following command file contains the "spool -la" command.

```
% ty sla.cmd
spool -la
```

By giving the Shell the name of a command file--minus the filename extension--you tell the Shell to execute that file.

```
% sla
```

Executing a command file executes all the commands in the file.

3. Working with Text Files

If you're interested in command files, you may wish to duplicate this sample command file. Use the Screen editor to create a file named **sla.cmd**, and proof the file carefully before you save it.

When the Shell executes the file, a list of all print jobs will be displayed--as if you'd given the command "spool -la" in response to the Shell prompt:

```
% sla
  Filename      User      Seq  Dev  Pri  Pages  Lines  Copies  Form
-> chapter1    betty     539  5:5  5    42     6781   1       0
  policy       fred      540  5:5  5     3     132    1       0
```

The system executes the commands in a command file as though you'd given the commands from the terminal keyboard. (To the Shell, you did.) You can write command files to perform many repetitive tasks. Refer to chapter 7 for details.

Before continuing, you may wish to delete some of the sample files you created in this chapter. If so, keep the **practice** file (it is used again in chapter 6).

Chapter 4

CROMIX FILE STRUCTURE

The Cromix-Plus system must keep track of hundreds of files--including executable binary files (such as **who.bin**) and text files (such as **practice**).

In many respects, the Cromix-Plus file system is similar to an ordinary filing system. But, instead of file folders, the Cromix-Plus system organizes information in directories. On a computer, information is always stored in files. Thus, the information in a directory is one or more files.

For example, on every Cromix-Plus system, there is a directory called the **bin** directory. In this directory are most files with **.bin** filename extensions (**screen.bin**, **ls.bin**, etc.). The system stores text files (such as **practice**) in other directories. Using the commands in this chapter, you can create some directories and organize your information--the files you create with the Cromix-Plus system.

The Home Directory

The System Administrator created one directory for you--your user (or home) directory. Normally, the name of your home directory is your login name. Whenever you log in to the system, you are working in your home directory. Unless you change directories (as explained later in the chapter), the system stores your files in your home directory.

In chapter 3, you created a simple file system, or structure. It consists of your home directory and a text file.

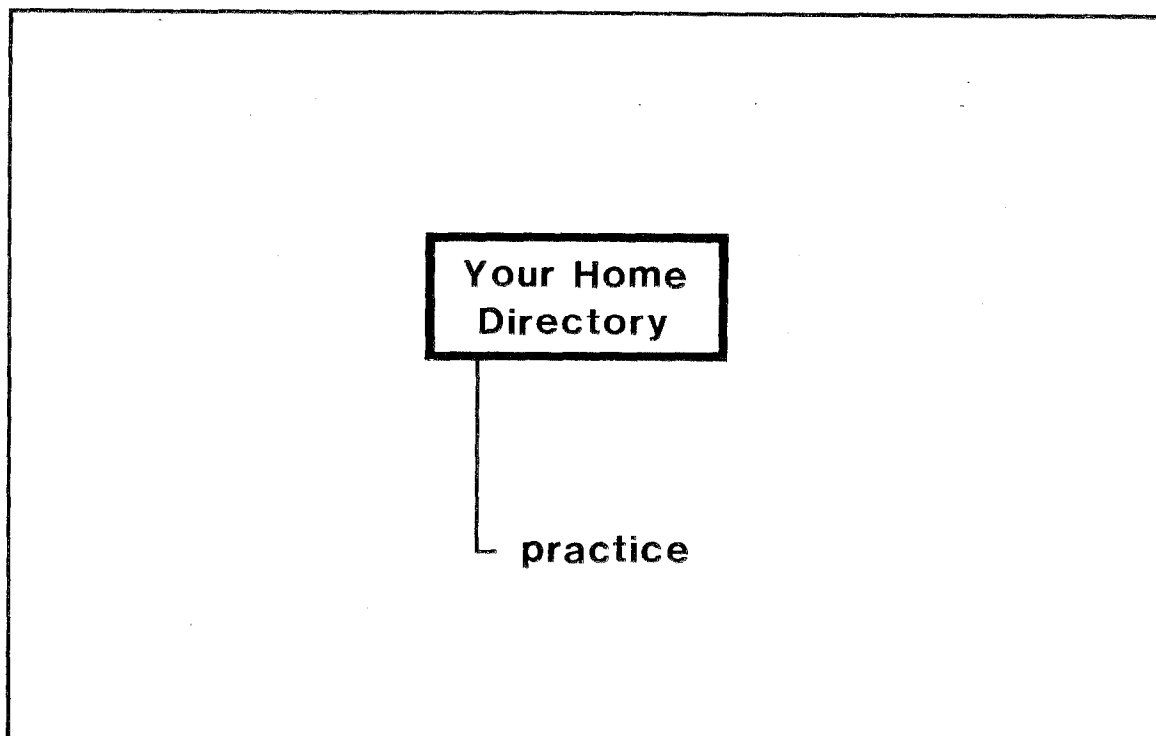


Figure 4-1: FILE STRUCTURE 1

The next section explains how your file structure fits into a much larger structure.

Visualizing the Cromix-Plus File Structure

The Cromix-Plus file system is hierarchical. In other words, its parts are ranked, so some are on higher levels than others. Figure 4-2 shows a typical system (because there are hundreds of files on any system, only directories are shown):

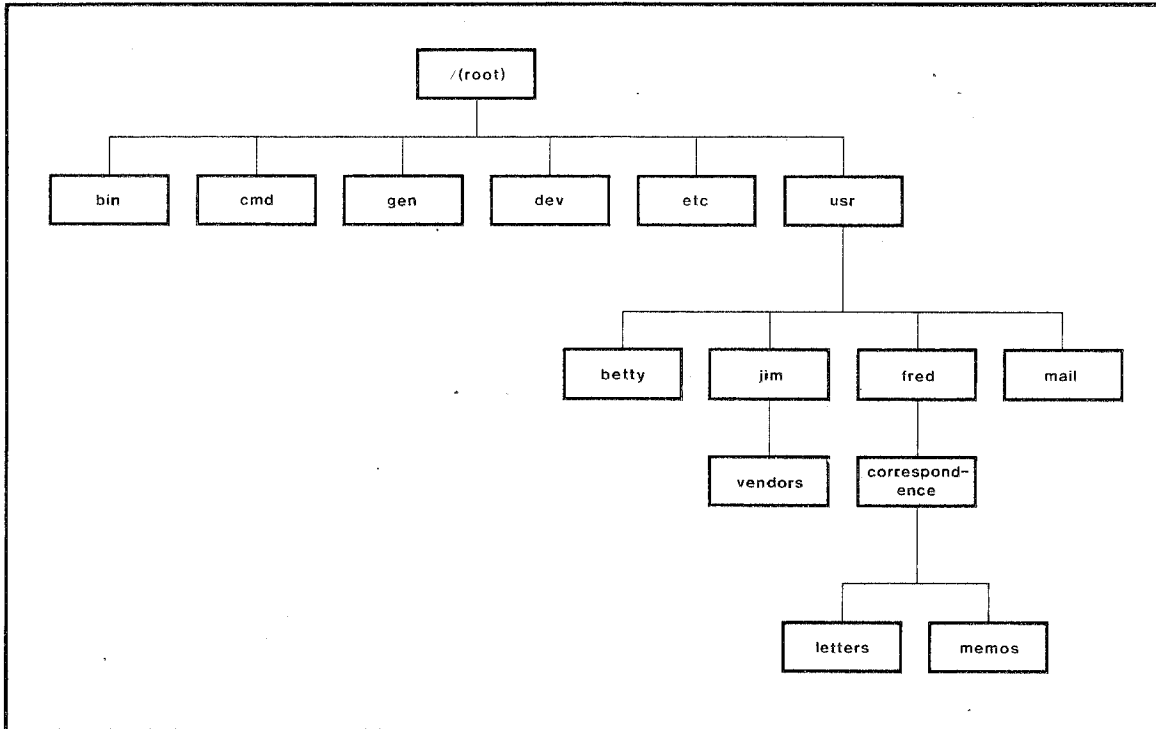


Figure 4-2: FILE STRUCTURE 2

As a family tree traces the ancestry of people, so the Cromix-Plus "tree" traces the ancestry of files. The ancestor of all files is the root directory. Its name is represented by a slash mark (/). Like your home directory, / (root) contains other files. But, most of the files in / are not ordinary files--they are directory files.

To the Cromix-Plus system, each directory (/ , your home directory, and so on) is a file. You cannot edit a directory file:

```
% screen /  
Cromix Screen Editor version xx.xx  
Directory file: "/"
```

Nor can you execute a directory file:

```
% /  
Command not found: "/"
```

But, directories are files nonetheless. A directory can contain ordinary files, other directory files (called subdirectories), or a combination of ordinary files and subdirectories.

Cromemco Introduction to Cromix-Plus
4. Cromix File Structure

For example, one of the files in / is a subdirectory called **bin**. This directory contains most of the executable files with **.bin** filename extensions. On the same level as **bin** is another subdirectory of / called **usr**. Figure 4-2 shows that **usr** contains subdirectories of its own. One of these subdirectories is your home directory.

Using the analogy of the family tree, / is the parent directory of **bin** and **usr**, while **usr** is the parent directory of every user's home directory.

You can determine the name of your current directory by giving the D (directory) command:

```
% d  
/usr/you
```

Without an argument, the D command prints the absolute pathname of the current directory. Until you change directories, as explained later in the chapter, the command displays the absolute pathname of your home directory. It traces a path to the directory file **you** (your login name) from the root directory, / (see figure 4-3).

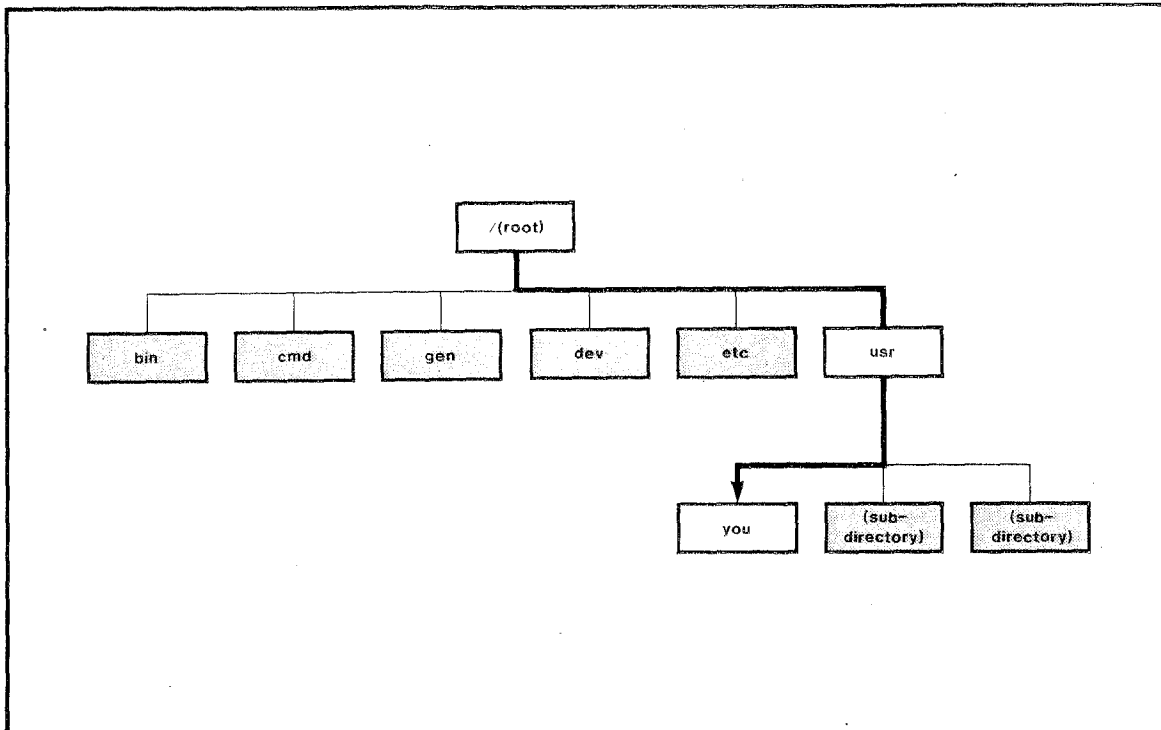


Figure 4-3: FILE STRUCTURE 3

Absolute Pathnames

The first item in an absolute pathname is the name of the root directory (/). The last item is a filename.

Using absolute pathnames, the system can locate any file in the Cromix file structure. The first "/" tells the system to look for the file from the root directory. Each successive "/" tells the system to look one level deeper in the file structure.

For example, the absolute pathname of the file **who.bin** is **/bin/who.bin**. This pathname gives the system the information it needs to trace a path from the root directory to the file **who.bin** in the **bin** directory. (For clarity, figure 4-4 shows only a few of the files in the **bin** directory.)

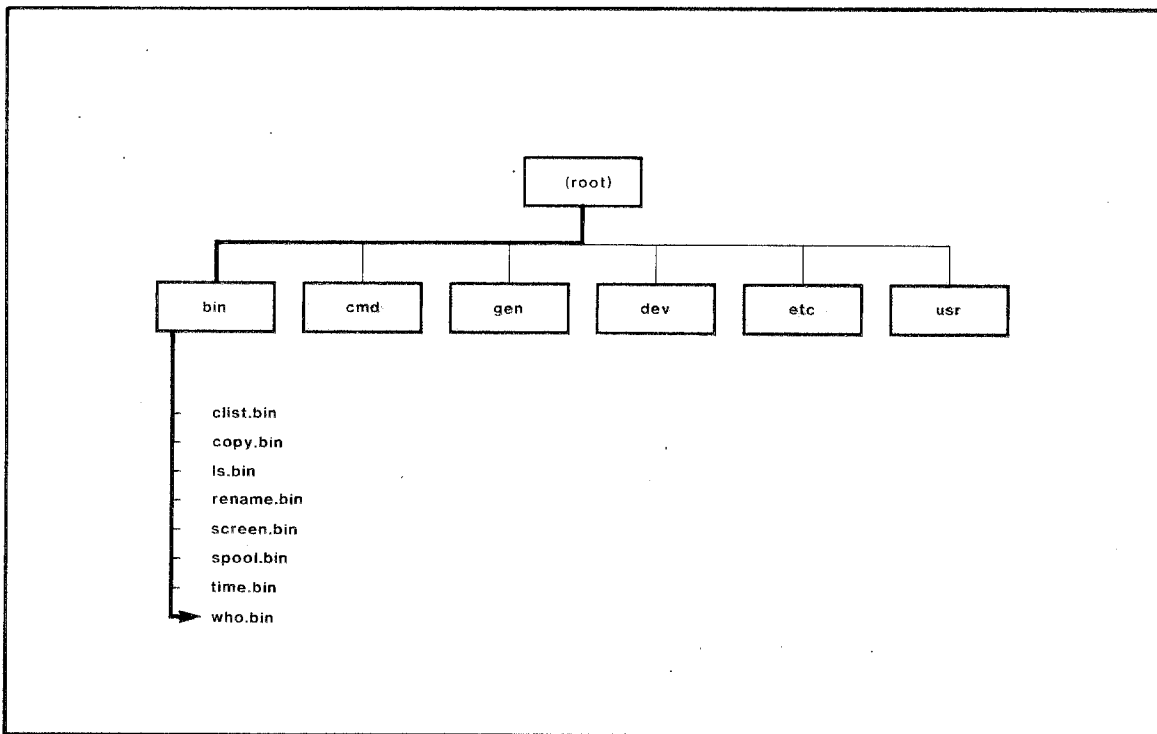


Figure 4-4: FILE STRUCTURE 4

Using the Ls command, you can display information about the files in many directories. Without an argument, Ls displays information for the current directory. With an argument, the pathname of a directory, Ls displays information for that directory.

Cromemco Introduction to Cromix-Plus
4. Cromix File Structure

To illustrate, give the following command:

```
% ls -m /usr
```

Without changing the current directory (you're still working in your home directory), Ls displays information about the files in the **usr** directory. An uppercase "D" means the file is another directory file, a subdirectory of **usr**. Preceding each "D" is the number of files in that directory.

The file corresponding to your login name is your home directory. In the following sample display, **bill** is the name of a user's home directory, as is **alan**, **jimmy**, **mark**, and **sue**.

```
% ls -m /usr
```

```
Directory: /usr
 6 D      1 alan
 3 D      1 bill
98 D      1 help
35 D      1 jimmy
 4 D      1 mail
27 D      1 mark
 2 D      1 pkg
 5 D      1 query
 4 D      1 spool
83 D      1 sue
```

Subdirectories such as **help**, **mail**, and **spool** serve other purposes. They are not user directories, as are **mark** or **sue**.

The Ls command with the **-l** or **-e** options displays even more information about the files in **/usr**. (Because the "ls -e" command displays so much information, only a portion of the display is shown.)

```
% ls -e /usr
```

```
Directory: /usr
alan                                     6 directory
  created:   Oct-16-1984 11:13:25  rewa re-- re--
  modified:  Oct-16-1984 11:13:25  alan          group: 1
  accessed:  Nov-14-1984 08:41:14  links: 1
  dumped:    000-00-1900 00:00:00  inode: 393

bill                                     1 directory
  created:   Nov-06-1984 09:59:41  rewa re-- re--
  modified:  Nov-06-1984 09:59:41  bill          group: 1
  accessed:  Nov-06-1984 10:03:49  links: 1
  dumped:    000-00-1900 00:00:00  inode: 157
```

Cromemco Introduction to Cromix-Plus
4. Cromix File Structure

```
help                               98  directory
created:  Oct-12-1984 12:20:36  rewa re-- re--
modified:  Oct-12-1984 12:20:36  bin           group: 32767
accessed:  Nov-06-1984 13:27:34  links: 1
dumped:    000-00-1900 00:00:00  inode: 146
.
.
.
```

When you give the commands, both displays will show the date and the time the System Administrator created your home directory (**/usr/you**) and the home directories of the other system users.

Ordinary files have absolute pathnames, too. For example, the following command displays the contents of the **practice** file:

```
% ty /usr/you/practice
Help
Passwd
Query
Who
Exit
%
```

The first "/" told the system to look for the file from the root directory. Each successive "/" told the system to look one level deeper in the file structure.

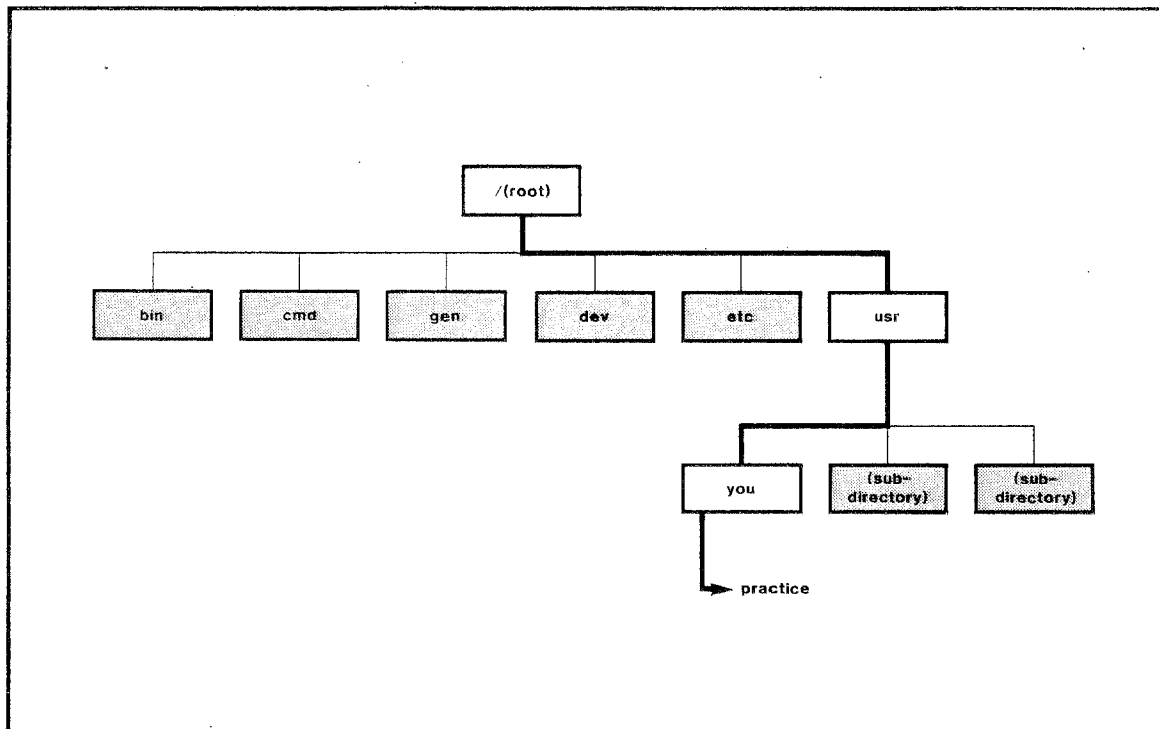


Figure 4-5: FILE STRUCTURE 5

Cromenco Introduction to Cromix-Plus

4. Cromix File Structure

First, the system looked for a subdirectory of / named **usr**. Finding **usr**, the system looked for a subdirectory of **usr** named **you**. Finding **you**, the system looked for (and found) the **practice** file.

Each directory in a pathname must be the parent of the next. For this reason, the following command produces an error message:

```
% ls /etc/usr
File not found: "/etc/usr"
```

The system cannot trace a path from the root directory through the **etc** directory to the **usr** directory because **etc** and **usr** are on the same level in the file structure.

As supplied, the Cromix-Plus system makes a few directories inaccessible to nonprivileged users. For example, in the **usr** directory, there is a subdirectory named **mail**. (Chapter 5 explains this directory's purpose.) The access privileges (chapter 3) associated with the file **/usr/mail** do not normally include "execute" access for nonprivileged users. Without execute access for a directory, you cannot use the name of that directory in a pathname.

The following command attempts to make **mail** the current directory using an absolute pathname:

```
% d /usr/mail
Directory not accessible: "/usr/mail"
```

Because you do have access to the directories you really **need** to work with (such as your home directory), you may never see a similar message. If you do see "Directory not accessible," however, its meaning is always the same. Somewhere in the pathname you gave the Shell is a directory for which you lack execute access.

The System Administrator can explain the situation. If need be, the System Administrator can also change the access privileges for the directory. Refer to the discussion of the Access utility in the Cromix-Plus User's Reference Manual.

A privileged user, such as the System Administrator, automatically has access to all the system's directories.

How to Make Sure You Have Execute Access for a Directory

You have execute access for many files you do not "own" simply because you can log in to the Cromix-Plus system. When discussing access privileges, anyone who can log in is called a member of the "public".

When you give the "ls -l" command, public access privileges are defined by the last four characters preceding each filename. A lowercase "e" means you, as a member of the public, have execute access for that file.

To illustrate, consider the following display:

```
% ls -l /usr
```

```
Directory: /usr
```

```
  6 D 1 rewa re-- re-- alan      Oct-16  11:13 alan
  3 D 1 rewa re-- re-- bill      Nov-06   9:59 bill
 98 D 1 rewa re-- re-- bin       Oct-12  12:20 help
 35 D 1 rewa re-- re-- jimmy     Oct-12  14:40 jimmy
  4 D 1 rewa ---- ---- bin      Oct-12  12:20 mail
 27 D 1 rewa re-- re-- mark      Oct-16  11:20 mark
  2 D 1 rewa re-- re-- bin       Oct-12  12:20 pkg
  5 D 1 rewa re-- re-- bin       Oct-12  12:20 query
  4 D 1 rewa ---- ---- bin      May-04  12:20 spool
 83 D 1 rewa re-- re-- system    Oct-12  15:38 sue
```

Members of the public (Alan, Bill, Jimmy, Mark, and Sue) have execute access for all subdirectories of **/usr** except **mail** and **spool**. The four characters preceding these filenames do not include a lowercase "e" (for execute). Thus, none of this system's nonprivileged users may use the name of either directory in a pathname.

Access privileges affect nonprivileged users only. The System Administrator, and any other privileged user, has access to all files. (For more information about access privileges, consult the discussion of the Access utility in the Cromix-Plus User's Reference Manual.)

Displaying the Absolute Pathname of an Executable File

Using the Path command you can display the absolute pathname of an executable file.

For example:

```
% path who
/bin/who.bin
```

Or:

```
% path screen
/bin/screen.bin
```

If Path's argument is the name of an intrinsic command, Path displays the message "Shell command," as in the following examples:

```
% path ex
ex: Shell command
% path del
del: Shell command
```

You cannot use Path to display the absolute pathnames of ordinary files, such as **practice**:

```
% path practice
Command not found: "practice"
```

There is additional information about the Path command in chapter 7.

Relative Pathnames

A pathname that does not begin with a "/" is a relative pathname. The path traced by a relative pathname starts at (or is relative to) the current directory.

A relative pathname can be as simple as the name of an ordinary file. For example, with your home directory the current directory, the following command displays the **practice** file.

```
% ty practice
```

Using a relative pathname, the system looks for (and finds) the file within the current directory.

Like absolute pathnames, relative pathnames can trace a path through many levels of the file structure. Consider figure 4-6, a typical user's file structure. Like your own file structure, it begins with the user's home directory (**fred**). The home directory is the parent of another directory, **correspondence**, and that directory is the parent of other directories (**letters** and **memos**). The contents of the **letters** and **memos** subdirectories are ordinary files.

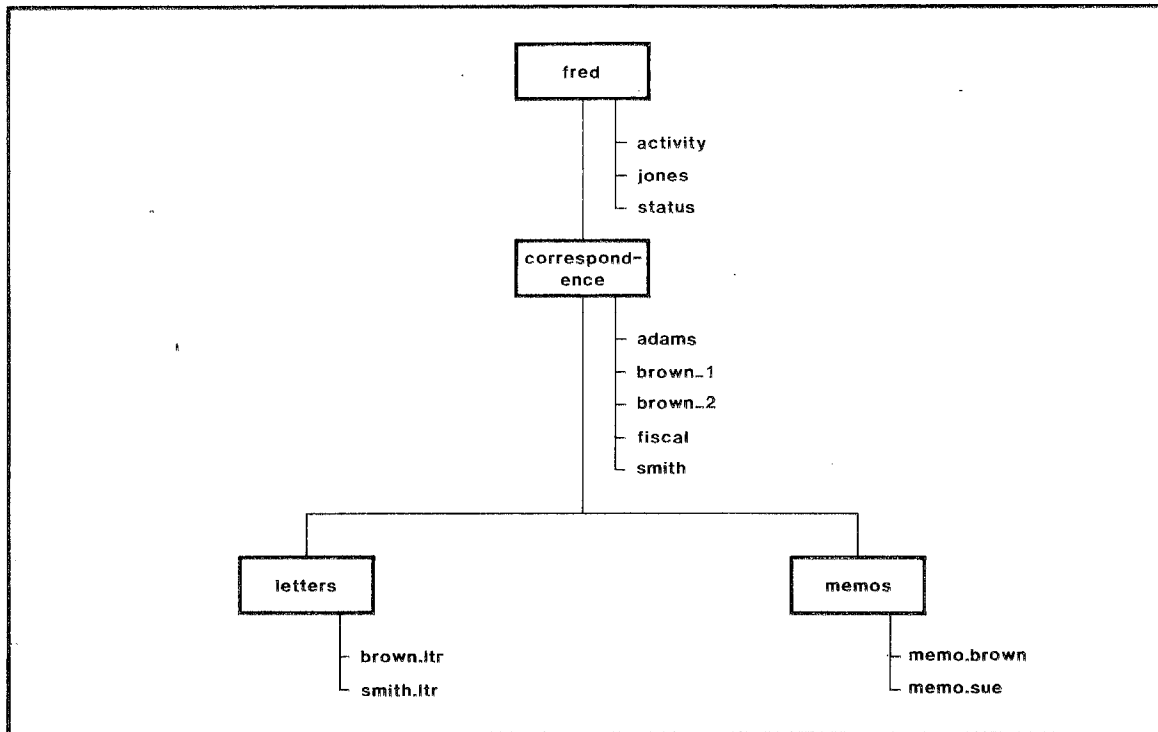


Figure 4-6: FILE STRUCTURE 6

While the home directory is the current directory, the following command displays information about the files in the **memos** directory:

```
% ls -m correspondence/memos
```

```
Directory: correspondence/memos
576 1 memo.brown
320 1 memo.sue
```

```
%
```

In a similar manner, the next command displays the contents of the file **memo.sue**.

```
% ty correspondence/memos/memo.sue
```

Figure 4-7 shows the path the system traced to locate the file.

Changing Directories

The D command, which you've used to display the absolute pathname of the current directory, can also be used to change directories. To make another directory the current directory, the command syntax is:

```
d directory-pathname
```

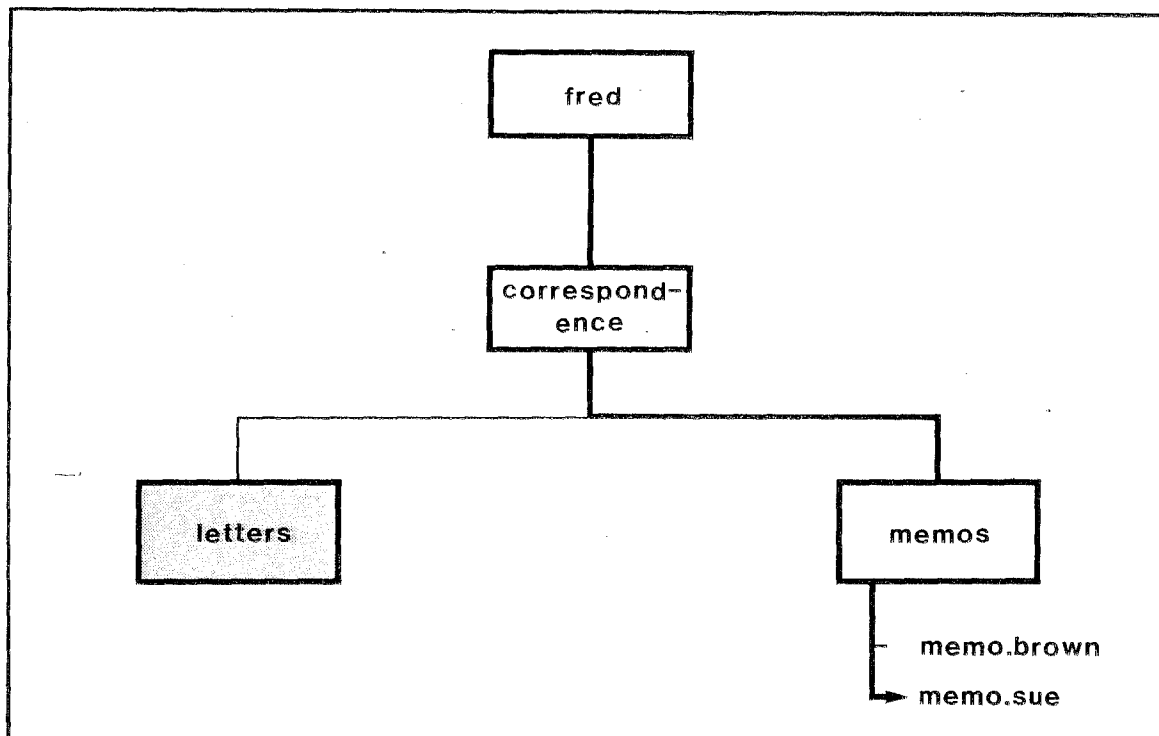


Figure 4-7: FILE STRUCTURE 7

For "directory-pathname," substitute the relative or absolute pathname of the new directory.

To move to any directory that is a descendant of the current directory, it's easiest to use a relative pathname. For example, while **fred** (figure 4-6) is the current directory, the following command makes **correspondence** the current directory:

```
% d correspondence
```

To move to any directory that is an ancestor of the current directory, use an absolute pathname. When you use absolute pathnames, the current directory is immaterial. From **correspondence** (or any other directory), the following command makes **fred** the current directory:

```
% d /usr/fred
```

In a similar way, while **memos** is the current directory, the following command makes **letters** the current directory:

```
% d /usr/fred/correspondence/letters
```

Because the system can locate any file from the root directory, the current directory is immaterial.

Unless you take advantage of the "shortcuts" discussed later in the chapter, you must use absolute pathnames to move up in the Cromix-Plus file structure.

Creating a Directory

The Makd (Make Directory) command creates directory files:

```
makd directory-pathname
```

The command argument can be a relative or absolute pathname. Because creating a directory creates a new file, the last item in the pathname must be a new filename.

To illustrate, if **fred** (figure 4-6) is the current directory, the following command creates a subdirectory of **letters** named **jan_june**.

```
% makd correspondence/letters/jan_june
```

If **letters** (figure 4-8) is the current directory, the pathname is simply a new filename.

```
% makd jan_june
```

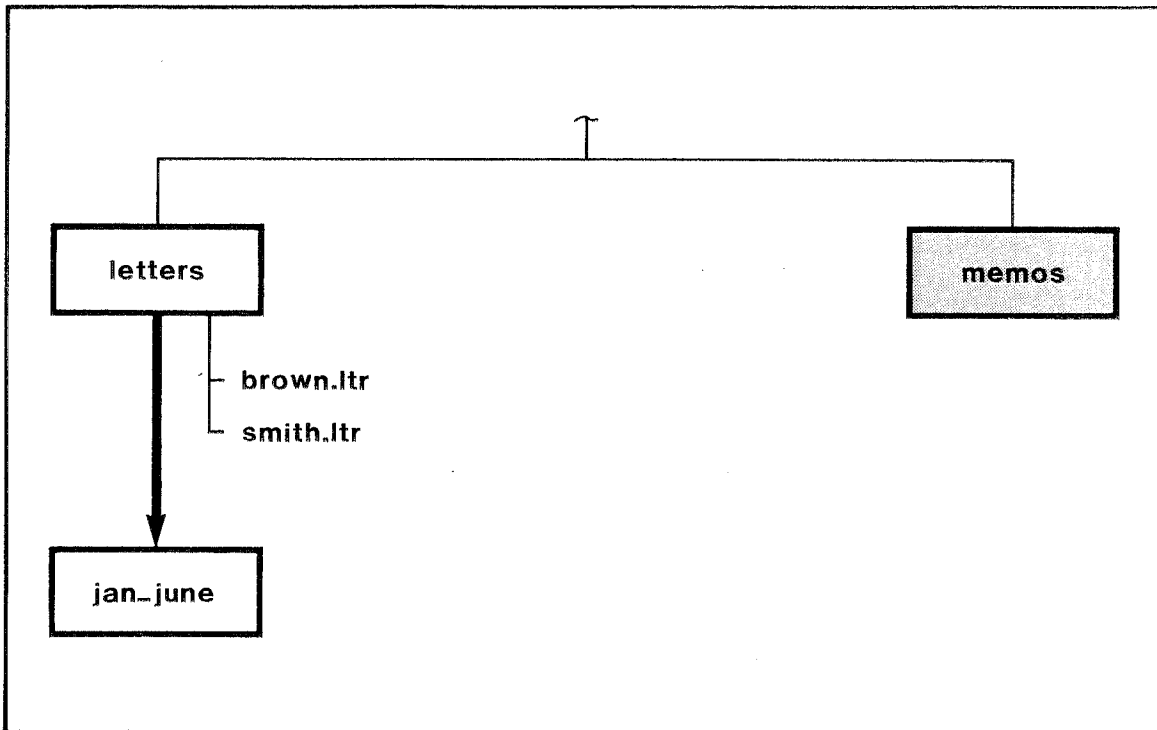


Figure 4-8: FILE STRUCTURE 8

Moving Files to a Directory

The Move utility moves files from one directory to another:

```
move file-pathname(s) directory-pathname
```

When moving a group of files, it is usually easiest to give the command from the directory that contains the files. In this way, you reduce the need for typing long pathnames.

To illustrate, if **correspondence** (figure 4-6) is the current directory, the following command moves two files from **correspondence** to the **jan_june** subdirectory of **letters**:

```
% move brown_1 brown_2 letters/jan_june
```

Each filename is the shortest kind of relative pathname.

How Move Works

In "moving" files within a Cromix file structure, Move does not physically move files. A file you "move" is still stored at the same location on the computer's hard disk (the file's inode number is unchanged). The Move command creates a new link to the inode and deletes the old link.

For example, in "moving" **brown_1** to **jan_june**, all Move did was change the absolute pathname of the file from

```
/usr/fred/correspondence/letters/brown_1
```

to

```
/usr/fred/correspondence/letters/jan_june/brown_1
```

Copying Files to Another Directory

The Copy command (chapter 3) can also copy one or more files to another directory:

```
copy file-pathname(s) directory-pathname
```

4. Cromix File Structure

For example, while **correspondence** is the current directory, the following command makes a copy of the file **smith** in the **letters** directory:

```
% copy smith letters
```

There are now two files named **smith**, one in the **correspondence** directory, another in **letters**:

```
% ls /usr/fred/correspondence
```

```
Directory: /usr/fred/correspondence  
adams    fiscal  letters memos    smith
```

```
% ls /usr/fred/correspondence/letters
```

```
Directory: /usr/fred/correspondence/letters  
brown.ltr  jan_june  smith     smith.ltr
```

Files in different directories can have the same name. Because their pathnames are different, they are different files to the Cromix-Plus system.

Renaming Files With Move and Copy

By using another form of these commands, you can rename a file as you move or copy it to another directory:

```
move file-pathname file-pathname  
copy file-pathname file-pathname
```

To rename a file, the final command argument is the pathname of an ordinary file (not a directory pathname).

For example, with **correspondence** the current directory, the following command moves a file named **fiscal** to the **memos** directory.

```
% move fiscal memos/budget
```

Because **budget** is an ordinary filename, the file is renamed **budget** in its new directory, as shown in figure 4-9.

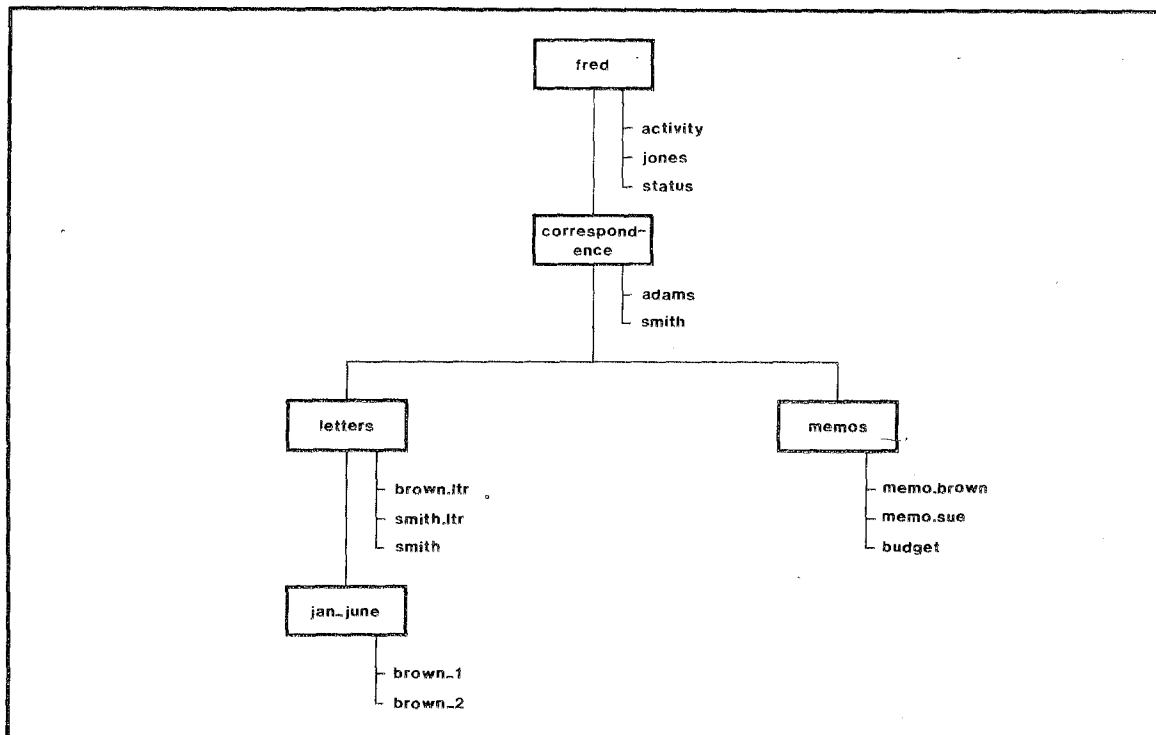


Figure 4-9: FILE STRUCTURE 9

It is possible to accidentally rename files using Move. To demonstrate, suppose that Fred's **correspondence** directory is the current directory, and he gives the following command to move the file **adams** from **correspondence** to **letters**:

```
% move adams lettrs
```

Because there is no subdirectory of the current directory named **lettrs**, Move renames **adams** to **lettrs**--within the current directory.

```
% ls /usr/fred/correspondence
```

```
Directory: /usr/fred/correspondence
letters lettrs memos smith
%
```

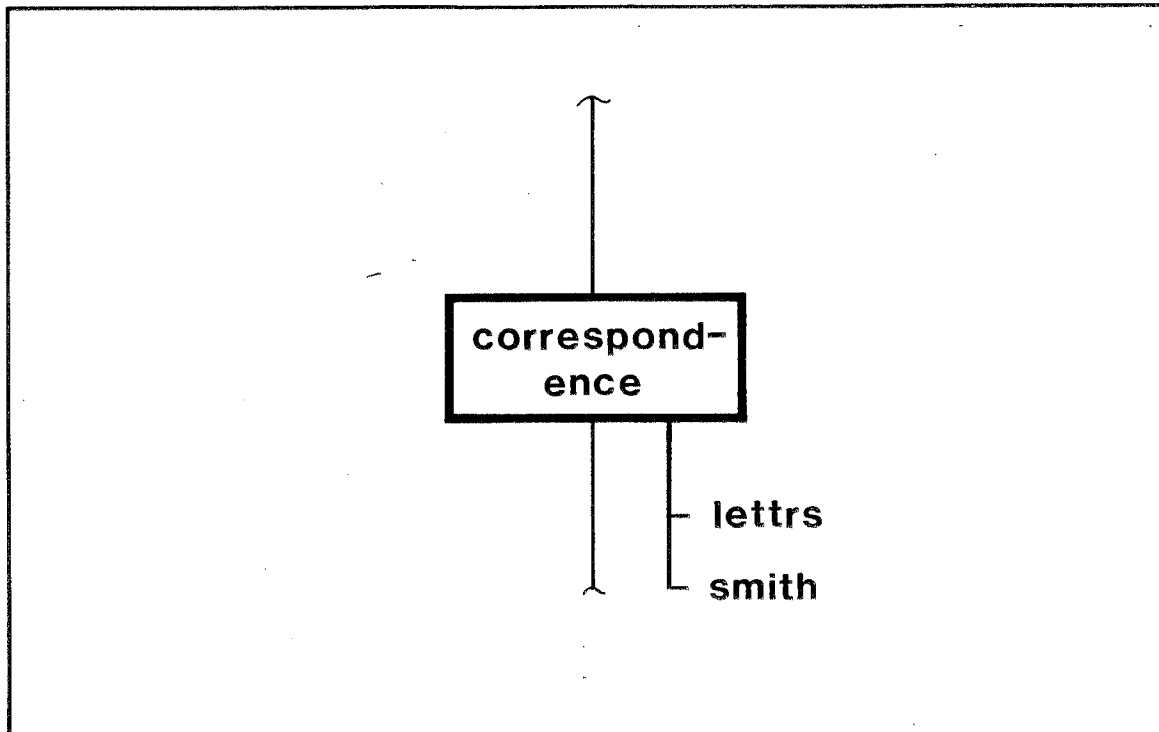



Figure 4-10: FILE STRUCTURE 10

Shortcuts for Working Within a Directory Structure

To reduce the need for typing long pathnames, the Cromix-Plus system provides some useful notations:

1. A single period (.) represents the current directory.
2. A double period (..) represents the home directory.
3. A caret (^) represents the parent directory.

Cromemco Introduction to Cromix-Plus
4. Cromix File Structure

Using the file structure shown in figure 4-11, here are some things you can do with these notations.

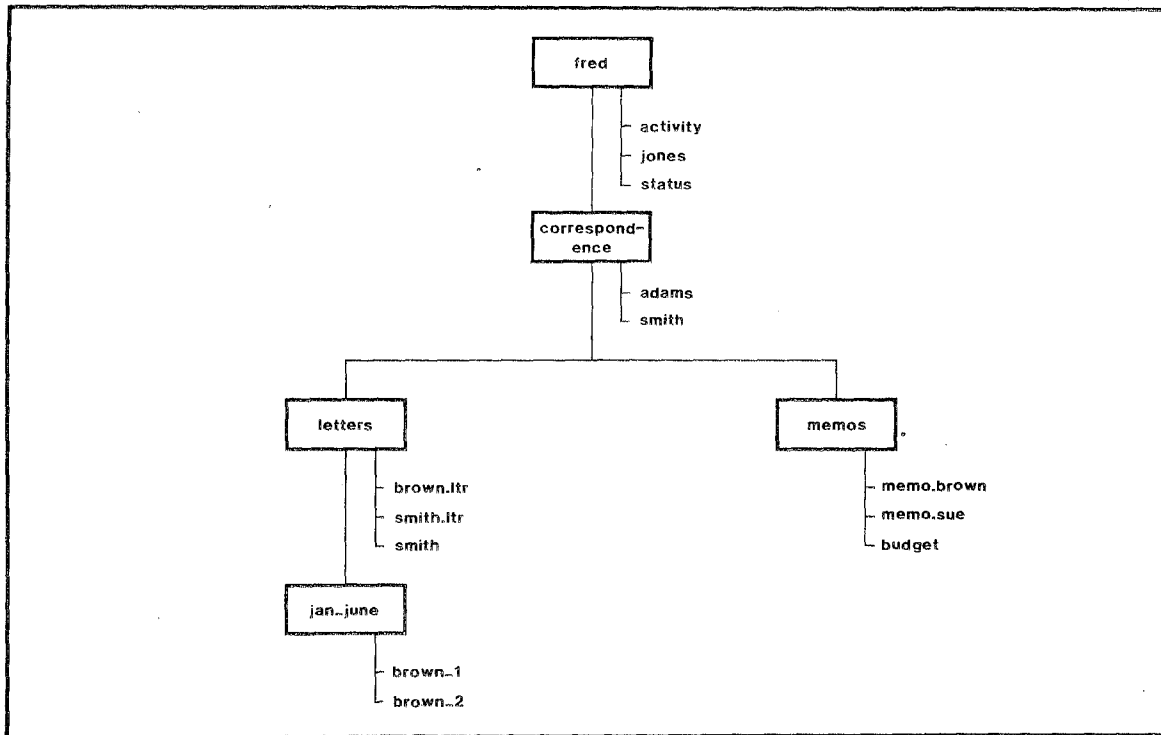


Figure 4-11: FILE STRUCTURE 11

With **jan..june** the current directory, the following command makes **letters** the current directory:

```
% d ^
```

With **letters** the current directory, the next command moves a file named **jones** from the home directory (..) to **letters** (.):

```
% move ../jones .
```

Without these notations, you would have to use absolute pathnames to move the file:

```
% move /usr/fred/jones /usr/fred/correspondence/letters
```

The double period (..) is often used to make the home directory the current directory:

```
% d ..  
% d  
/usr/fred
```

The command "d .." makes the home directory the current directory from any directory in the Cromix file structure.

If **jan_june** is the current directory, this command makes its parent directory, **letters**, the current directory:

```
% d ^
```

Or, from **jan_june**, this command makes the "grandparent" of **jan_june** the current directory:

```
% d ^^  
% d  
/usr/fred/correspondence
```

Each ^ moves one level higher in the Cromix file structure.

One or more carets (^) can also be used as part of a pathname. To illustrate, if **letters** is the current directory, the following command displays the file **memo.brown** in the **memos** directory:

```
% ty ^memos/memo.brown
```

The system started looking for the file from the parent of the current directory.

Deleting a Directory Structure

You can delete a directory using the Del command (chapter 3), as you would any other file. You must, of course, delete all the directory's descendants (files and subdirectories) before deleting the directory itself. The following series of commands would delete the **memos** directory from its parent directory, **correspondence**.

```
% del memos/budget  
% del memos/memo.brown  
% del memos/memo.sue  
% del memos  
%
```

Cromemco Introduction to Cromix-Plus
4. Cromix File Structure

To simplify the process, the Cromix-Plus system provides the Deltree (Delete Tree) utility, which deletes an entire directory structure (or "tree"). Deltree deletes the directory and all its descendants.

The Deltree command syntax is:

```
deltree directory-pathname
```

Deltree asks for confirmation before deleting each file. To answer, type **y** or **n** without pressing RETURN.

```
% deltree memos  
Delete memos/budget? y  
Delete memos/memo.brown? y  
Delete memos/memo.sue?
```

Deltree's final prompt asks if the directory itself should be deleted.

```
% deltree memos  
Delete memos/budget? y  
Delete memos/memo.brown? y  
Delete memos/memo.sue? y  
Do you really want to delete all of memos? y  
%
```

Because Deltree asks for confirmation before deleting each file, it can be used to "prune" a directory structure. In other words, without intending to delete an entire directory structure, you can use Deltree to selectively delete a series of files.

When you really want to delete all files, you may wish to use the **-a** (for all) option.

```
% deltree -a memos  
Delete memos? y  
%
```

When you select the **-a** option, Deltree asks if you wish to delete the entire directory structure. If you type **y**, Deltree deletes all files--without requesting further confirmation.

Copying a Directory Structure

The Cptree (Copy Tree) utility makes a copy of all or parts of a directory structure. Before using Cptree, you must first create a destination directory with Makd. This example creates a destination directory named **oldletters** (the command is given from the **correspondence** directory):

```
% makd oldletters
```

To copy an entire "tree" (a directory and all its descendants), the command syntax is:

```
cptree source destination
```

"Source" refers to the existing directory and its descendants--the source of the new directory structure.

For example, with **correspondence** the current directory, the following command copies **letters** and all its descendants to the **oldletters** directory.

```
% cptree letters oldletters
```

Cptree copies **letters** and all its descendant files, as shown in figure 4-12.

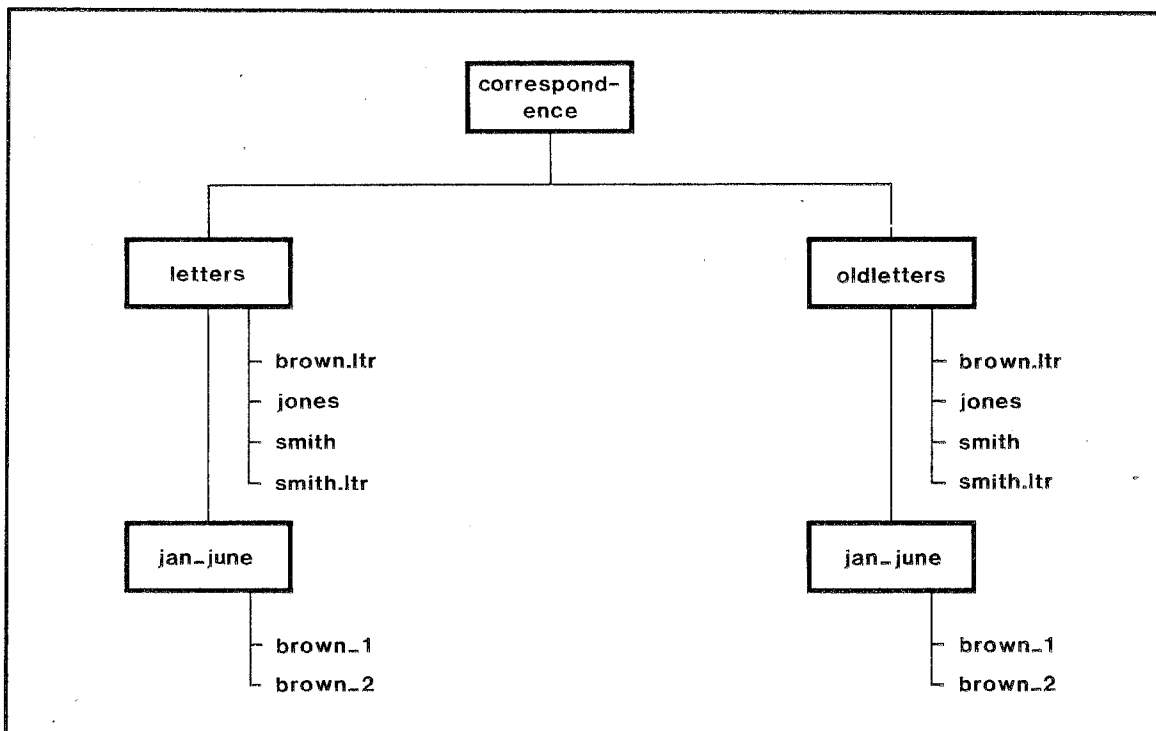


Figure 4-12: FILE STRUCTURE 12

Because it can take a long time to copy an entire "tree," you may wish to use the **-v** (for verbose) option when you give the command. Cptree with the **-v** option displays each filename as it is copied.

```
% cptree -v letters oldletters
oldletters/brown.ltr
oldletters/smith.ltr
oldletters/jan_june
oldletters/jan_june/brown_1
oldletters/jan_june/brown_2
oldletters/jones
oldletters/smith
```

In using the Cromix-Plus system, you'll discover other commands you can give with a **-v** (for verbose) option. Selecting this option displays useful information on the terminal screen while the program executes. The information displayed depends on the program.

How the Shell Looks for Executable Files

When you give a command, the Shell interprets the first item on the command line as the name of an executable file with a **.bin**, **.com**, or **.cmd** filename extension.

For example:

```
% time
```

causes the Shell to look for an executable file named **time**, with a **.bin**, **.com**, or **.cmd** filename extension. Finding the file, the Shell calls the corresponding program.

You do not need to supply the pathnames of programs or commands because the Shell knows where to look for these files. The Shell searches the following directories, in the order shown:

1. The current directory
2. The **/ram** directory (if present)
3. The **/bin** directory
4. The **/cmd** directory

4. Cromix File Structure

Note: The **/ram** directory will be present if your system has a RAM disk containing executable files.

Following this "search path," the Shell looks for the file in the following way:

1. Within the current directory, the Shell looks for the file with a **.bin** extension. If **file.bin** is not found, the Shell looks for the same file with a **.com** extension. If **file.com** is not found, the Shell looks for the same file with a **.cmd** extension.
2. If **file.cmd** is not found within the current directory, the search proceeds in the **/ram** directory (if present). Within **/ram**, the Shell looks for **file.bin**, **file.com**, and **file.cmd**.
3. If **file.cmd** is not found within the **/ram** directory, the search proceeds in the **/bin** directory. Within **/bin**, the Shell looks for **file.bin** and **file.com**.
4. If **file.bin** and **file.com** are not found within the **/bin** directory, the search proceeds in the **/cmd** directory. Within **/cmd**, the Shell looks for **file.cmd**.
5. If **file.cmd** is not found, the Shell displays the error message:

Command not found

Special Files in the Home Directory

If you create a file in your home directory named **.reminder**, the Shell will display the contents of that file on the terminal screen whenever you log in to the system.

The contents of a typical **.reminder** file is shown here:

```
% ty .reminder
Don't forget to fill out your timesheet.
%
```

Logging in with this **.reminder** file in the home directory might look like this:

```
LOGIN: jane
Password:
```

```
Logged in jane Nov-15-1984 12:18:40 on qtty1
Message of the day: Welcome to the Cromix-Plus Operating System
Don't forget to fill out your timesheet.
%
```

Cromemco Introduction to Cromix-Plus

4. Cromix File Structure

You may also wish to create a file in your home directory named **.startup.cmd**. The Shell will execute this command file whenever you log in.

A simple **.startup.cmd** file might contain these commands:

```
% ty .startup.cmd
who
ls
%
```

With this file in the home directory, logging in might produce the following result:

```
LOGIN: fred
Password:

Logged in fred  Nov-14-1984  09:53:47 on tty3
Message of the day: Welcome to the Cromix-Plus Operating System
betty      tty1      Nov-14-1984 07:16:22      0      0
jim        tty2      Nov-14-1984 09:42:29      0      0
fred       tty3      Nov-14-1984 09:53:47      0      0
activity   correspondence jones      status
%
```

Because they begin with periods, **.reminder** and **.startup.cmd** are "invisible" filenames. The **Ls** command will not display information about the files unless you use the **-a** (for all) option. In the following example, **-m** and **-a** combine to produce a medium-detailed list of all files.

```
% ls -ma
  9      1 .startup.cmd
 950     1 activity
  5 D    1 correspondence
 505     1 jones
1,216   1 status
```

Device Files

The System Administrator often works with files that are neither ordinary files nor directory files. This third kind of file is the device file.

Most device files represent the system's peripheral devices. A peripheral device is any piece of hardware attached to the computer that is not a part of the computer. The system printer is a peripheral device. So is every terminal connected to the computer. Each has a corresponding device file in the **/dev** (device) directory.

4. Cromix File Structure

You see your terminal's device filename whenever you give the Who command:

```
% who am i
john      qtty7      Nov-15-1984 11:35:51      0      0
```

In the sample display, it is **qtty7**, or **/dev/qtty7** (the file's absolute pathname).

The System Administrator uses the Makdev utility to add devices to the system. Makdev creates device files. Although you might never work with device files as the System Administrator does, knowing they exist may help you better understand the Cromix-Plus system.

To the Cromix-Plus system, devices and disk files (such as **practice**) appear to be files. This compatibility allows the Shell to redirect input and output as described in chapter 6.

Chapter 5

THE MAIL UTILITY

This chapter discusses the Mail utility--the program that handles the bulk of communication among Cromix-Plus system users. Such communication is from user to user, in the form of typed messages that Mail copies from one terminal for eventual display on another.

A typical mail "delivery" is shown here:

From sue, Nov-12-1984 15:57:01

Friday's meeting has been rescheduled. The new
time is Monday at 9:00 A.M.

Let me know if you can attend.

The Mail program automatically signs and dates all messages. Thus, whoever receives your message knows who sent it, when, and where to address a reply.

This chapter explains how to send mail and how to read the mail others send you. For more information about communications under the Cromix-Plus system, refer to the discussions of the Mail, Msg, and Ccall utilities in the Cromix-Plus User's Reference Manual.

Sending Mail

To send mail to another system user, use the following procedure:

1. Give the Mail command with that user's login name as an argument, as in the following example:

```
% mail john
```

2. Type your message, ending each line by pressing RETURN.

```
% mail john  
Friday's meeting has been rescheduled. The new  
time is Monday at 9:00 A.M.
```

3. When you're through, type CONTROL-Z on a line by itself:

```
% mail john
Friday's meeting has been rescheduled. The new
time is Monday at 9:00 A.M.
```

```
Let me know if you can attend.
(CONTROL-Z)
%
```

When Mail receives CONTROL-Z (end-of-file) from the terminal, the program knows there is no more input from the terminal. Mail sends the message and returns control to the Shell.

Correcting Mistakes While Using Mail

As you type your message, you can erase characters on the current line by pressing either the DELETE key or the LEFT ARROW key. After erasing one or more characters, you can then retype that portion of the line. This is the only way to make corrections when giving a program like Mail input from the terminal.

If you want to start over, press CONTROL-C, as you do to stop other programs as they execute. CONTROL-C displays a new Shell prompt--without sending mail.

Note: There is a way you can edit a message you send using Mail as you would an ordinary file. For details, refer to chapter 6, the section "Redirecting Input From a File."

What Happens to the Mail You Send

When you press CONTROL-Z, your message is not displayed immediately on some other terminal. Instead, it is delivered to a file in the **/usr/mail** directory, where it is stored until it is read.

Each system user has a place for mail within **/usr/mail**. As a nonprivileged user, you probably cannot "list" the files in **/usr/mail** (you lack execute access to the directory). If you're curious about its contents, a typical display is shown here:

```
# ls -m /usr/mail

Directory: /usr/mail
 0      1 alan
 0      1 bill
 0      1 jimmy
193     1 mark
 3 D    1 rpkg
 0      1 sue
```

Most of the filenames are user login names. In each of these files, the corresponding user's mail is stored until that user reads the mail from one of the system's terminals.

A "0" preceding the filename means the file is empty (at the moment, that user has no mail). In the sample display, only the user named Mark has mail. There are 193 characters stored in the file `/usr/mail/mark`.

How Do You Know When You Have Mail

When mail for you is received in `/usr/mail`, the Mail utility notifies you. You will see a message on whatever terminal you're using. Mail even displays the login name of the user who sent you mail, as in the following example:

```
You have mail from sue
```

If you are logged out when you receive mail, you'll see the message "You have mail" as soon as you log in.

Until you read your mail, the system will remind you about it each time you log in. Checking `/usr/mail` is part of the system's login procedure.

Reading Your Mail

To read mail, give the Mail command without an argument. Here, a user named John logs in to discover he has mail. He then gives the Mail command to read it:

```
LOGIN: john  
Password:
```

```
Logged in john Nov-13-1984 08:32:12 on qtty5
```

```
You have mail.
```

```
Message of the day: Welcome to the Cromix-Plus Operating System  
% mail
```

```
-----  
From sue, Nov-12-1984 15:57:01
```

```
Friday's meeting has been rescheduled. The new  
time is Monday at 9:00 A.M.
```

```
Let me know if you can attend.
```

```
Should mail be saved?
```

To answer Mail's prompt ("Should mail be saved?"), type **y** or **n** and press RETURN. If you type **n** for **no**, Mail discards that item of mail. If you type **y** for **yes**, Mail stores that item of mail in a file named **mbox**.

The mbox File

The first time you save mail, the Mail program creates a file named **mbox** in the current directory. The file contains the mail you saved.

% mail

From sue, Nov-12-1984 15:57:01

Friday's meeting has been rescheduled. The new
time is Monday at 9:00 A.M.

Let me know if you can attend.

Should mail be saved? y

% ty mbox

From sue, Nov-12-1984 15:57:01

Friday's meeting has been rescheduled. The new
time is Monday at 9:00 A.M.

Let me know if you can attend.

%

The **mbox** file is an ordinary text file. You can edit the file, rename it, print it, and so on. When you have an **mbox** file in the current directory, Mail adds new items of saved mail to the end of the file. If you delete the **mbox** file, Mail will create it again the next time you save mail.

Sending the Same Mail to Several Users

By giving the Mail command with several arguments (several login names), you can send the same mail to more than one user.

The following example sends the same mail to two users:

```
% mail betty jim
I need your estimates by Thursday
afternoon. No excuses!
(CONTROL-Z)
%
```

When Betty and Jim read their mail, each will receive the same message. By including your own login name on the command line, you can save copies of the mail you send others. The copy is created when you read your own mail and save it in the **mbox** file.

Chapter 6

THE CROMIX SHELL

This chapter explains some of the Cromix Shell's special features. By taking advantage of these features, you can make your work easier. For example, you can let the Shell generate the filenames it passes, as arguments, to a program. When the Shell generates filenames, you don't have to type a long list of filenames on the command line. In a sense, the Shell does it for you.

One of the Shell's most useful features is its ability to redirect a program's output to an ordinary file. When the Shell redirects output to a file, output you would normally see on the terminal is written to a file on the disk. Most of the sample output in this manual, such as the Ls command example shown below, was created by redirecting output to a file:

```
# ls -m /usr/mail  
  
Directory: /usr/mail  
0      1 alan  
0      1 bill  
0      1 jimmy  
193    1 mark  
3 D    1 rpkg  
0      1 sue
```

In the example, only the command itself was typed. The rest of the display was produced by redirecting Ls's output to an ordinary file, which was then added to the text of the manual. In this chapter, you will use the Cromix-Plus utilities to create some similar files.

The Shell can also redirect a program's input. Redirected input comes from an ordinary file instead of from the terminal keyboard.

Using another kind of redirection, called a "pipe," the Shell can connect one or more programs specified on the command line. The programs work together to produce a customized result.

To experiment with these, and other, features, you don't need to create a single new file with the Screen program. Nor do you need to learn to use additional utility programs. The two new utilities in this chapter (Sort and Create) are introduced only to aid your experiments. You already know enough Cromix-Plus commands to use all of the features discussed in the chapter.

The Standard Output

The Cromix-Plus system provides a standard file where programs can send their output. It is called STDOUT (for standard output). Although STDOUT is "visible" only to the Shell, you see its effects whenever you use the system.

For example, each time you've given the Who command, you've seen output (a list of system users) on your terminal screen:

```
% who
betty  qtty1  Nov-14-1984 07:16:22  0  0
jim    qtty2  Nov-14-1984 09:42:29  0  0
fred   qtty3  Nov-14-1984 09:53:47  0  0
```

It happened because the Who utility sends its output to STDOUT, and the Shell connects STDOUT with the terminal screen. Clist, Ls, and Date are some other programs that send output to STDOUT. Their output, too, appears on the terminal screen.

Unless you redirect output as explained in this chapter, STDOUT is always the terminal.

Note: Not all programs use STDOUT. For example, the Spool program sends its output to a system printer--not to the terminal screen.

The Standard Input

The Cromix-Plus system provides another standard file where programs can get input. It is called STDIN (for standard input). STDIN is connected to the terminal keyboard.

An example of a Cromix-Plus utility that uses STDIN is Mail. When you give Mail an argument (a login name), Mail takes its input from the terminal keyboard.

```
% mail john
Friday's meeting has been rescheduled. The new
time is Monday at 9:00 A.M.
Let me know if you can attend.
(CONTROL-Z)
%
```

CONTROL-Z (end-of-file) sends what you typed--via STDIN--to Mail.

Unless you redirect input as explained in this chapter, STDIN is always the terminal keyboard.

Note: Not all programs use STDIN. For example, Who takes its input from a file in the **etc** directory (**/etc/who**)--not from the terminal keyboard.

The Sort Utility

The Cromix-Plus system provides several programs that can use both STDOUT and STDIN. One of them is Sort, which sorts its input (normally, the lines in an ordinary file). In this chapter, you'll use this program to sort the lines in a file.

Your **practice** file (chapter 3), which contains a series of short, unalphabetized lines, is ideal for demonstrating Sort.

```
% sort practice
Exit
Help
Passwd
Query
Who
```

Sort takes its input--the lines in the **practice** file--and sends sorted output to STDOUT (the terminal screen). Often, that output is sorted alphabetically, as shown above.

Only Sort's output is sorted. The input file (**practice**) is unchanged.

```
% ty practice
Help
Passwd
Query
Who
Exit
%
```

Summarized, this is what happened when you gave the Sort command:

1. The Shell located and called the Sort program, passing on the command argument (the filename) to Sort. The Shell never checks the arguments and command options it passes to a program.
2. Sort located its input file and sent a sorted version of that file to STDOUT.
3. With STDOUT connected to the terminal, Sort's output appeared on the terminal screen.

A program like `Sort`, which can use both `STDOUT` and `STDIN`, is ideal for experimenting with the redirection techniques discussed in this chapter. The first of these techniques, redirecting output to an ordinary file, is explained in the next section.

`Sort` is one of the most versatile Cromix-Plus utility programs. For detailed information about this program, consult the Cromix-Plus User's Reference Manual.

Redirecting Output to a File

If you add a greater-than symbol (`>`) and a filename to the command line, the Shell will connect `STDOUT` to an ordinary file. Then, output that normally appears on the terminal screen is written to a file instead.

```
% who > who.out
%
```

The redirect-output symbol (`>`) instructs the Shell to redirect a program's output from its usual destination--the terminal screen--to a file. In this case, the Shell redirected `Who`'s output to the file `who.out`.

```
% ty who.out
betty  tty1  Nov-14-1984 07:16:22  0  0
jim    tty2  Nov-14-1984 09:42:29  0  0
fred   tty3  Nov-14-1984 09:53:47  0  0
```

Summarized, this is what happens when you give a command that contains a redirect-output symbol:

1. First, the Shell opens an output file (such as `who.out`) and connects `STDOUT` with that file. If the output file already exists, the Shell deletes its contents before it opens the file.
2. With the output file ready, the Shell calls the program (such as `Who`).
3. The program executes in response to the Shell. As always, the program sends its output to `STDOUT`. But, with `STDOUT` connected to an ordinary file, output that normally goes to the terminal goes to that file instead.

In the following example, `Ls`'s output is redirected to a file named `list.usr`.

```
% ls -m /usr > list.usr
```

The result is a file containing information about the files in the `usr` directory.

```
% ty list.usr
```

```
Directory: /usr  
 6 D 1 alan  
 3 D 1 bill  
98 D 1 help  
35 D 1 jimmy  
 4 D 1 mail  
27 D 1 mark  
 2 D 1 pkg  
 5 D 1 query  
 4 D 1 spool  
83 D 1 sue
```

```
%
```

Note: Redirecting output to a file works only with programs that send output to STDOUT.

Sort uses STDOUT. Thus, the following command creates an output file containing a sorted version of **practice** (Sort's input file).

```
% sort practice > practice.out
```

The output file **practice.out** is an ordinary file. You can print it using Spool or edit it using Screen:

```
% spool practice.out  
% screen practice.out
```

When you redirect a program's output to a file, the Shell will overwrite any file in the same directory with the same name. To demonstrate, create a file named **test** by giving the following command:

```
% who > test  
%
```

The file **test** contains the redirected output from Who.

```
% ty test  
betty qtty1 Nov-14-1984 07:16:22 0 0  
jim qtty2 Nov-14-1984 09:42:29 0 0  
fred qtty3 Nov-14-1984 09:53:47 0 0  
%
```

Now try redirecting the output from Time to the same file. Then display the contents of the file with Type.

```
% time > test
% ty test
Wednesday, November 14, 1984          9:58:12
%
```

The file **test** now contains only the output from Time. In opening a new output file named **test**, the Shell automatically deleted the previous contents of the **test** file.

When you don't want the Shell to overwrite a file with the same name, use the redirect-and-append symbol discussed in the next section.

Important: In opening an output file, the Shell automatically deletes the contents of an existing file with the same name. This happens BEFORE the Shell calls the program.

It is thus possible to accidentally delete a file while redirecting output. Consider the following command:

```
% clist prog.c > prog.c
```

The input file and the output file have the same name (**prog.c**). In executing this command, the Shell deletes the contents of **prog.c** before calling Clist. When Clist executes, the file **prog.c** contains nothing.

When redirecting output, make sure the input file (if any) and the output file have different filenames.

Appending Output to a File

The Shell will not overwrite a file with the same name if you use a redirect-and-append symbol (>>). Instead, the Shell adds (appends) the new output to the end of the existing file.

To demonstrate, give the following commands:

```
% time >> time.out
% ls >> time.out
```

The first command created a new file named **time.out**, a file containing Time's output. The second command appended Ls's output to the end of the file.

```
% ty time.out
          Friday, November 16, 1984      16:28:24
list.usr  practice    test      time.out  who.out
%
```

Redirecting Type's Output to a File

Unless you specify an input file, Type waits for input from the terminal (STDIN). By redirecting Type's output to a file, as you supply input from the terminal keyboard, you can create a text file using Type.

If you'd like to try it, give the following command:

```
% type > type.out
```

In response, the Shell calls Type--without passing a filename to the program. (The Shell opens the output file before calling the program.) Because the Shell called Type without an argument, Type expects input from STDIN (the terminal keyboard).

Enter two or three lines of text (as if you were using Mail). When you're through, type CONTROL-Z (for end-of-file).

```
% type > type.out
This is some sample text
I am typing into a file
named type.out.
(CONTROL-Z)
%
```

The new prompt means Type's output has been redirected. You now have a file named **type.out** in your directory.

```
% ls type.out
type.out
%
```

You can create a similar, sorted file using Sort:

```
% sort > sort.out
Music
Message
Case
Countess
(CONTROL-Z)
% ty sort.out
Case
Countess
Message
Music
%
```

Sort and Type can create files in this way because both programs use STDOUT, and both can take input from STDIN (the terminal keyboard). When called with an argument, a filename, they take their input from a file; when called without an argument, they take their input from STDIN. Some other programs that use STDIN when called without an argument are: Clist, Dump, Match, Scan, and Spool.

Redirecting Input From a File

If you add a less-than symbol (<) and a filename to the command line, the Shell will connect STDIN to an ordinary file. The redirect-input symbol (<) instructs the Shell to redirect a program's input so it comes from an ordinary file instead of from the terminal keyboard. In this example, the Shell redirects Mail's input from the file **info**.

```
% mail sue < info
%
```

Mail sends a copy of the text in the file--as if that text had been typed after giving the Mail command. Summarized, this is what happens when you give a command that contains a redirect-input symbol:

1. First, the Shell locates the input file (such as **info**) and connects STDIN with that file. If the Shell cannot locate the input file, the Shell displays the message "File not found" before calling the program.
2. The Shell calls the program (such as Mail).
3. The program executes in response to the Shell. As always, the program gets its input from STDIN. But, with STDIN connected to an ordinary file, input that normally comes from the terminal comes from that file instead.

4. The program returns control to the Shell when it encounters an end-of-file indicator. (At this point, the Shell displays a new prompt.)

When you redirect a program's input from a file, the Shell supplies the end-of-file indicator for you. You do not need to type CONTROL-Z (for end-of-file), as you do when a program's input comes from the terminal keyboard.

Note: Redirecting input from a file works only with programs that can take input from STDIN.

Running a Job in the Background

So far, your commands have executed in seconds. After each command, the Shell quickly displayed a new prompt so you could give another. Some commands, however, can take much longer to execute.

To illustrate, give the following command:

```
% ls -e /bin > bin.out
```

The Shell will not display a new prompt until Ls's output (information about the files in the **bin** directory) has been redirected. Because there are a lot of files in the **bin** directory, and the **-e** option produces so much information, the process will take some time.

When you have a new prompt, try a similar version of the same command. Add an ampersand (&) to the command line:

```
% ls -e /bin > bin.out &
```

An ampersand instructs the Shell to start a detached process--a process that is no longer connected to your terminal. In seconds, the Shell displays a PID (process identification) number and a new prompt. You can now give a new command while the Ls program executes in the background (as a detached process). You can even log out while the program executes.

Any program that takes a long time to execute can be run in the background. For example, if you are copying a long file, you may wish to run the job as a detached process:

```
% copy longfile longfile.save &  
PID=1578  
%
```

Using Cptree (chapter 4) to copy an entire directory structure is also time-consuming.

```
% cptree bigdirectory newdirectory &  
PID=5781  
%
```

While a program runs as a detached process, pressing CONTROL-C will not stop it. To stop a detached process, use the Kill command. Kill's argument is the PID number of the process. This example starts and stops a typical detached process:

```
% copy bigfile bigfile.copy &  
PID=4137  
% kill 4137  
%
```

As a non-privileged user, you can kill any process the Shell starts for you. You cannot kill a process the Shell starts for another user.

Important: When the Shell runs Spool as a background process, the PID # you see on the terminal is Spool's process number. The printer daemon (chapter 3) is a separate process, and the Kill utility will not stop it. To kill background printing, use the "spool -k" command.

Giving Sequential Commands

You can give the Shell more than one command on a single command line by separating the commands with a semicolon (;). To illustrate, give the following command sequence:

```
% time;who;ls
```

In response, Time executes, followed by Who, followed by Ls. The Shell processes each command in the order it appears on the command line. This is called sequential processing.

In the next example, the Shell deletes a file before Ls executes.

```
% del who.out;ls  
list.bin  list.usr  practice  sort.out  test      time.out  type.out  
%
```

When giving the Shell a series of commands, you can also use an ampersand to separate the commands. The Shell executes a command followed by an ampersand as a detached process (in the background).

This example redirects output to a file named **logged_in** in the background and, at the same time, calls the Screen editor:

```
% who > logged_in&screen practice
```

Or, this command starts two detached (or background) processes:

```
% time > time.out&ls /usr > l.out&
```

Parentheses on the Command Line

Parentheses can group commands so the output from two or more sequential processes can be redirected to the same file. For example, the following command redirects the output of Time and Who to a file named **group**:

```
% (time;who) > group  
%
```

This command does the same thing in the background:

```
% (time;who) > group &  
PID=892  
%
```

With either command, the result is the same as if you'd given these commands:

```
% time > group  
% who >> group
```

For more information about processes, refer to the discussion of the Pstat command in the Cromix-Plus User's Reference Manual.

Redirecting Error Messages

The Cromix-Plus system provides another standard file where programs can send error messages. It is called STDERR (for standard error). Like STDOUT, STDERR is normally connected to the terminal. Thus, error messages appear on the terminal screen.

Sometimes, error messages can be disconcerting. For example, if you are using the Screen editor while a job runs in the background (as a detached process), you might not want error messages from the background job interrupting your work. If so, you can redirect error messages to a file. An asterisk (*) immediately following a redirect symbol (>) or a redirect-and-append symbol (>>) instructs the Shell to redirect error messages that would normally appear on the terminal screen to an ordinary file.

For example, the following commands move a series of files--and redirect error messages to a file named **error**.

```
% move memol memo2 memo3 memos >* error
% move letter1 letter2 correspondence >>* error
%
```

Error messages, if any, are contained in the file **error**.

```
% ty error
File not found: "memo3"
File not found: "letter1"
%
```

Redirection with Pipes

A pipe symbol (|) on the command line instructs the Shell to connect the output from one program to the input of another. For example, the following command uses a pipe to connect Ls with Spool:

```
% ls /usr | spool
```

Spool's input is the output from Ls. The result is a printed list of the files in the **/usr** directory, as if you'd given the following commands:

```
% ls /usr > list.out
% spool list.out
% del list.out
```

A pipe does not create an intermediate file. Instead, Ls's output is channeled directly to Spool.

Try this similar command to connect Who and Sort:

```
% who | sort
```

The result is a sorted list of system users.

This command uses two pipes to produce a printed, sorted list of system users.

```
% who | sort | spool
```

Using pipes, the Shell can connect a variety of programs that were not specifically designed to work together. At the beginning of a pipe is a program that sends output to STDOUT. In the preceding command, both Who and Sort fill this requirement. At the end of a pipe is a program that can take its input from STDIN. In the sample command, both Sort and Spool fill this requirement.

When you use a pipe (|), the programs run concurrently. The Shell starts three processes (one for each program) when you give this command:

```
% who | sort | spool
```

Even though Spool must wait for Who and Sort to execute before it has output to print, it is running nevertheless. If system memory is at a premium, you may wish to use the redirection technique discussed in the next section instead of a pipe.

Redirecting Output to a Temporary File

Using a redirect-output symbol immediately followed by a redirect-input symbol, you can duplicate the effects of a pipe. For example, the following command prints a list of system users:

```
% who >< spool
```

A "><" symbol, called a sequential "pipe," instructs the Shell to create a temporary file. In the sample command, the Shell redirects Who's output to a temporary file, and then redirects Spool's input from the same file. The file is "temporary" because the Shell automatically deletes it before displaying a new prompt.

The Shell runs each process (for example, Who and Spool) sequentially. In other words, Who executes before the Shell calls Spool. This form of redirection, although slower than using a pipe, saves system memory.

The Tee Command

Using the Tee command, you can redirect output to a file and have that output appear on the terminal as well. For example, this command redirects Sort's output to a file (**practice.sort**) and displays that output, too.

```
% sort practice | tee practice.sort
```

The Tee command requires an argument--the pathname of an ordinary file. For more information about the Tee command, refer to the Cromix-Plus User's Reference Manual.

Filename Generation

A command argument that contains an asterisk (*) or question mark (?) is an ambiguous file reference:

1. An asterisk (*) matches a string of zero or more characters. An asterisk does not match a leading or embedded period.
2. A double asterisk (**) matches zero or more characters, including an embedded period. A double asterisk does not match a leading period.
3. A question mark (?) matches any single character other than a leading period.

In response to an ambiguous file reference, the Shell generates the names of specific files and passes those filenames to a program. For example, suppose you have the following files in your directory:

```
account
data_sheet
intro
memo
memos
memo.fred
memol
memo2
```

The following command displays information about all the "memo" files:

```
% l m**
memo memos memo.fred memol memo2
```

Because a double asterisk matches zero or more characters (including an embedded period), the Shell generates the names of all files that begin with "m".

This command displays information about the file **memo.fred** only:

```
% l m*.f*
memo.fred
```

Only this file has a name that consists of "m", followed by zero or more characters, followed by a period, followed by zero or more characters.

This command uses a question mark to display information about the files that begin with "memo", followed by any single character:

```
% l memo?  
memos memo1 memo2
```

The Shell does not generate the filename **memo** because it does not consist of "memo" followed by another character.

You can usually use an ambiguous file reference wherever you'd use an ordinary filename. In this example, an ambiguous file reference is used to print the file **memo1**:

```
% spool m*1
```

Or, in this example, an ambiguous file reference is used to move the files **memos**, **memo1**, and **memo2** to another directory.

```
% move memo? correspondence
```

Because ambiguous file references can produce some unforeseen matches, be careful when you use them. When in doubt, test them with the Ls command:

```
% ls ?.txt  
1.txt 2.txt 3.txt 4.txt 5.txt
```

If none of the filenames in a directory begins with a period, a double asterisk--by itself--matches every filename in the directory. Thus, the following command will usually delete all the files in the current directory:

```
% del **
```

This command is mentioned because, sooner or later, almost everyone tries it. Resist temptation, and use the Deltree utility instead. Deltree requires confirmation from you before deleting files--Delete does not.

It is safest not to use ambiguous file references with the Delete command. If you feel you must, always check the current directory before giving the Delete command.

Specifying a Range of Characters

Using another kind of ambiguous file reference, you can instruct the Shell to look for a range of characters. The Shell will substitute characters in square brackets for the corresponding character in a filename.

To illustrate, suppose your directory contains the following files:

```
a_letter  
b_letter  
c_letter  
a_memo  
b_memo  
c_memo  
d_memo
```

The following command would display information about the files **a_letter** and **b_letter**:

```
% ls [ab]_letter  
a_letter  b_letter
```

Without the brackets, the Shell would pass the argument "ab_letter" to Ls, and Ls would look for an exact match:

```
% ls ab_letter  
file not found: "ab_letter"
```

The letters "a" and "b", in square brackets, define a range of characters the Shell may substitute for the first character in the filename.

In a similar manner, the next command displays information about all files that begin with "a" or "d":

```
% ls [ad]*  
a_letter  a_memo  d_memo
```

To specify a wider range of characters, you can use a hyphen within the brackets, as in this example:

```
% ls [b-d]_memo  
b_memo  c_memo  d_memo
```

The hyphen means "through," as it does in normal usage.

An Important Consideration Regarding Filename Generation

The number of filenames the Shell can generate in response to an ambiguous file reference is limited by the number of characters the Shell can pass, as command arguments, to any program. The maximum is 512 characters. When asked to pass more characters than this to any Cromix-Plus program, the Shell displays the message "Arg list too big," as shown below:

```
% ls /bin/**  
Arg list too big
```

If you give this command, you'll see the same message, followed by a partial list of files in the `/bin` directory. This message comes from the Shell. The program (in this case, `Ls`) has no way of knowing it received a partial list of filenames from the Shell.

If, as you use the Cromix-Plus system, you see an "Arg list too big" message, try breaking the list into two commands, as in the following examples:

```
% move [a-k]** /usr/jerry/sales_leads  
% move [l-z]** /usr/jerry/sales_leads
```

Better still, avoid the problem by limiting the size of your directories. If the Shell cannot generate the names of all the files in a user's directory, that directory contains too many files. Create some subdirectories, and reduce the clutter.

Experimenting with Filename Generation

Without a group of files with similar filenames, it's difficult to appreciate the Shell's ability to generate filenames. If you would like to experiment with filename generation, there is an easy, quick way to create a series of files.

The `Create` command with a filename as an argument creates a file containing zero bytes. The file has a name, but there is no information in the file. By giving the command with a list of filenames, you can create a series of zero-byte files:

```
% create big bigger biggest 2big really_big
```

The preceding command will create five new files in your current directory. Although the files contain no characters, they will show up in the list `Ls` displays. Because their filenames are similar, they are ideal for demonstrating filename generation.

Chapter 7

WRITING COMMAND FILES

In addition to general information about command files, this chapter discusses some commands that are often used in command files. A few of these commands (such as Shift and Rewind) are used nowhere else.

Most likely, when you've used the Cromix-Plus system for a while, you'll discover some repetitive task you could do more easily with a command file. By taking advantage of the Shell's ability to interpret command files (the programs you write for the Shell), you can make your work a lot easier.

Command-File Description

Command files are ordinary files, which contain a series of Cromix-Plus commands. A command file must have the filename extension **.cmd** (for command). The following command file (**list.cmd**) contains the "ls -m" command:

```
% ty list.cmd  
ls -m
```

Giving the command filename (minus the filename extension) from the command line executes the file:

```
% list  
239    1 letter  
241    1 memo  
50     1 output  
240    1 plan6_10
```

Command files can be this simple. Command files can also perform more complex tasks, such as copying files from the computer's hard disk onto floppy diskettes for long-term storage.

When you give a command that executes a command file, that file should be located in the current directory or the **/cmd** directory. (If **/ram** is present, command files may also reside in that directory.) To execute a command file in some other directory, supply a complete pathname:

```
% /usr/tes/list
```

OR

```
% ../list
```

Note: Only privileged users may create files in the **/cmd** directory. Even if you are privileged, you should check with the System Administrator before adding command files to **/cmd**.

As supplied, the Cromix-Plus system has some standard command files in **/cmd**. For example, the file named **/cmd/bak.cmd** deletes all files in the current directory with **.bak** filename extensions. (For information about the Bak command, consult the Cromix-Plus User's Reference Manual.)

A Practical Use of the Path Command

Because the Shell looks first in the current directory for executable files (refer to chapter 4), you have complete freedom over the command files you create within your own directory structure. For example, if you have a file named **sort.cmd** (a personalized version of Sort) in your current directory, the following command executes your file--not the file **sort.bin** in the **/bin** directory:

```
% sort infile > outfile
```

To find out what file (if any) the Shell will execute in response to a particular command, use the Path command:

```
% path sort  
/usr/you/sort.cmd
```

In this example, the Shell located the file in the current directory. Giving the Sort command now will execute the file **sort.cmd**.

Here, the Shell locates the file in the **/bin** directory:

```
% path sort  
/bin/sort.bin
```

If you do have a file named **sort.cmd** it is not in the current directory. Giving the Sort command now will execute the file **sort.bin**.

To avoid confusion, it is a good idea to give your command files distinctive filenames, such as **mysort.cmd**. Once again, the Path command will tell you if the Shell can execute the file:

```
% path mysort  
Command not found: "my sort"
```

"Command not found" means the file, if it exists, is not in any of the directories the Shell searches for executable files (refer to chapter 4). To execute the file, you must either change directories or supply a more complete pathname, as in the following examples:

```
% ls ../my sort
../mysort.cmd
% ../my sort infile > outline
%
```

The first command verifies the existence of the file in a user's home directory. The second command executes the file.

Redirection Within a Command File

Any command you can give from the command line can be put in a command file. This means you can use all the redirection techniques discussed in chapter 6.

Because command files can execute many commands, redirecting error messages is often desirable. You can even discard error messages (so they are neither displayed nor saved in a file) by redirecting them to a file named **null** in the **/dev** directory:

```
del *.temp >* /dev/null
```

/dev/null is a null device. Redirecting any kind of output to this file discards that output.

The next section introduces the Echo command, which is especially useful when writing command files. Using Echo, you can send informative messages to the terminal as a command file executes.

The Echo Command

Echo "echoes" its arguments to the terminal screen, as in the following example:

```
% echo Hello there.
Hello there.
%
```

You can echo "special" characters (such as an asterisk or pipe symbol) by quoting that character on the command line, as in the following example:

```
% echo This is a special character: ">" right  
% This is a special character: > right
```

As an alternative, you may wish to quote the entire string of arguments, as in this example:

```
% echo 'This is a special character: > right'  
This is a special character: > right
```

The Shell ignores special characters anywhere within single or double quotation marks. This is true when using Echo or any other utility. Without the quotation marks, the sample command redirects Echo's output to a file named **right**:

```
% echo This is a special character: > right  
% ty right  
This is a special character:  
%
```

Chapter 3 contains a list of special characters you must quote if the Shell is to interpret them literally on a command line.

Command File Structure

Within a command file:

1. You can instruct the Shell to substitute arguments from the command line for up to nine numbered parameters in the command file.
2. You can specify labels and comments.
3. You can jump to labels using the Goto command, or do conditional jumps using Goto and If.

Items 1 and 2 are discussed more fully in the following subsections. After these subsections, the If and Goto commands are discussed.

Argument Substitution -- For #1 in a command file, the Shell substitutes the first argument from the command line, for #2, the second, and so on, through the ninth command-line argument (#9).

Cromemco Introduction to Cromix-Plus

7. Writing Command Files

For example, the following command file displays two files--#2 (the second command argument) and #1 (the first command argument).

```
% ty display.cmd
ty #2
ty #1
```

The following command executes the file and displays the files **letter1** and **letter2**.

```
% display letter1 letter2
This is letter 2
.
.
.
This is letter 1
.
.
.
%
```

For **#*** in a command file, the Shell substitutes all command-line arguments:

```
% ty display_all.cmd
ty #*
% display_all letter1 letter2
This is letter 1
.
.
.
This is letter 2
.
.
.
%
```

For the asterisk, the Shell substitutes arguments in order--first, second, and so on, until there are no more arguments.

You can pass even more arguments to a command file using ambiguous file references on the command line. This example (again using the **display_all** command file) displays all the files in the current directory with **.txt** filename extensions:

7. Writing Command Files

```
% display_all *.txt
This is text-file 1
.
.
.
This is text-file 2
.
.
.
This is text-file 3
.
.
.
```

Specifying Labels and Comments -- A percent sign anywhere on a line means the rest of the line is a comment. In the following example, "display all files" is a comment:

```
% ty display_all.cmd
ty #*          % display all files
```

The Shell executes the command (ty #*) and ignores the comment. Comments are always ignored. Their only purpose is to make a command file's operation more understandable.

A comment at the beginning of a line--with no space after the percent sign--is a label. In the following example, "%start" and "%end" are labels:

```
% ty display_all.cmd
%start
ty #*          % display all files
%end
```

As described in the next section, labels can affect the operation of a command file.

The Goto Command

The Goto command transfers control within a command file. Goto's argument is always a line label.

To illustrate, consider this sample command file:


```
% ty echo_args.cmd
%start
echo #*
%end
```

All this file does is echo any command arguments.

Adding a Goto command causes the file to execute again and again.

```
% ty echo_args.cmd
%start
echo #*
goto start
%end
```

The command "goto start" starts the file executing once more from the beginning, the label **start**. Only CONTROL-C stops execution.

A Goto command always interrupts the sequential, line-by-line execution of a command file. After a Goto command, execution continues at another point in the command file, as determined by Goto's argument (a label). The process is commonly called "jumping" to a label.

Using Goto in combination with the If command (the next section), you can do "conditional" jumps. In other words, you can jump to a label if--and only if--some condition is satisfied. If the condition is not satisfied, the Goto command is ignored.

If you give the Goto command with a nonexistent line label, any commands in the file after Goto are not executed.

The If Command

The If command conditionally executes another command (often, the Goto command).

The most common forms of the If command are:

1. Execute the command if the previous command returned an error:

```
if -err command
```

All commands return a value to the Shell when they are through executing. The Shell, in turn, passes that value to the next command. A non-zero value indicates an error.

The "if -err" command tests for a non-zero return value. If the preceding command returns an error, the command following "if -err" executes, as in this example:

```
if -err goto end
```

If the preceding command returns a zero value, the command following "if -err" (in this case, Goto) does not execute.

There is additional information about return values in this chapter and in the Cromix-Plus User's Reference Manual.

2. Execute the command if a specified condition is true or false:

```
if string-1 = string-2 command  
if string-1 != string-2 command
```

The relational operator (!=) means "not equal." Spaces must bracket both relational operators (= and !=).

For example, this line tests for an argument:

```
if #1hi = hi goto done
```

For #1, the Shell substitutes the first argument on the command line. In the absence of an argument, the Shell substitutes a null string (comparable to nothing) for #1. In this case, "#1hi" and "hi" are equal, and the Goto command is executed. If they are not equal (the condition specified by If is false), the Goto command is not executed.

Comparing #1 to a single character, such as a period (.), is the quickest way to test for an argument:

```
if #1. = .
```

You must compare #1 to something because the following command is syntactic nonsense:

```
if #1 =
```

You may be used to combining If with Else and Endif. Under the Cromix-Plus system, If-Else-Endif constructions are not possible. However, you can usually emulate their effects with consecutive If-Goto constructions.

The Shift Command

The Shift command shifts arguments in a command file. After a Shift command, #1 matches the second argument on the command line, #2, the third, and so on. Using Shift commands, you can cycle through a series of arguments.

If you want to return all command-file parameters to their original values, use the Rewind command.

The Rewind Command

The Rewind command cancels all preceding Shift commands in a command file. After a Rewind command #1 matches the first argument on the command line, #2, the second, and so on.

The Exit Command

Within a command file, Exit returns control to the Shell. Exit logs you out only when you give the command in response to the Shell prompt.

By giving the Exit command with an argument (a number), you can control the value Exit returns to the Shell. Any nonzero value will return an error. Without an argument, Exit returns whatever value it receives from the Shell.

The following command file, which sorts a series of files to the file **sortout**, contains several Exit commands. Clist displays the file, providing line numbers for reference (the line numbers and heading are not a part of the file):

```
% clist sort_all.cmd
File      SORT_ALL.CMD      Wednesday, November 14, 1984    10:14:29
  1      if #1. = . goto error % if no arguments
  2      sort #* >> sortout % sort input files
  3      exit
  4
  5      %error
  6      echo "Give me an argument!"
  7      exit 1
```

The first Exit command returns the value returned by the preceding command (Sort). The second Exit command returns an error value, regardless of the value returned by the preceding command (Echo).

Which Exit command executes depends on the outcome of the argument test (line 1). If true (#1. = .), execution jumps to **error**, and lines 6 and 7 execute.

```
% sort_all
Give me an argument!
%
```

If the argument test (line 1) is false (#1. != .), lines 2 and 3 execute. The rest of the file (lines 4 through 7) does not execute.

The Input and Testinp Commands

The Input utility reads one line from STDIN and writes that line to STDOUT. The line is written when you press RETURN to end that line.

```
% input
This line is displayed when I press RETURN.
This line is displayed when I press RETURN.
%
```

By redirecting output, you can use Input to create a file containing one line of text.

```
% input > oneline
This line is written to a file when I press RETURN.
% ty oneline
This line is written to a file when I press RETURN.
%
```

Input, when used with Testinp (the next command discussed), lets you communicate from the terminal with an executing command file.

The Testinp Command

Testinp tests for equality between an input file and one or more text strings--ignoring letter case. For example, the following command tests the file **temp** for equality with the string **yes** or **please**.

```
testinp temp yes please
```

If the contents of the file is not identical to either of the two strings, Testinp returns an error value. (Like all commands, Testinp returns a value to the Shell when it is through executing.)

With the **-f** (for first) option, Testinp tests the first character of the file for equality with the first character of one or more strings. With the **-d** (for delete) option, Testinp deletes the file after the test.

```
% testinp -d temp yes please
% ty temp
File not found: "temp"
%
```

Cromemco Introduction to Cromix-Plus
7. Writing Command Files

The display does not reflect the results of the test. It isn't supposed to. All Testinp does is test a file and return a value based on the test.

To test for a multiword string, enclose the string in quotation marks, as in this example:

```
testinp temp "yes please"
```

In response to this command, Testinp returns an error if the contents of the file does not match the entire string "yes please."

Important: Do not conclude that Testinp tests for a string embedded in the text of a file. It does not. Testinp tests for equality between the file and the string. In other words, whether the file contains the string is irrelevant. The contents of the file must be the same as the string.

To test for embedded text strings, use the Match utility (refer to the Cromix-Plus User's Reference Manual).

If no use for Testinp springs to mind, consider the following command file. This command file runs the Shutdown utility if confirmation from the terminal is received:

```
% clist crash.cmd
File CRASH.CMD          Thursday, October 25, 1984    14:30:16
  1  echo "Do you want to shut down the system?"
  2  input > temp
  3  testinp -fd temp y
  4  if -err goto noshutdown
  5  shutdown
  6  %noshutdown
  7  exit
%
```

Line 1 echoes the phrase "Do you want to shut down the system?"

```
% crash
Do you want to shut down the system?
```

Input now waits for input from the terminal. Any typed response will be redirected to the file **temp**.

In this example, the user types **yes** and presses RETURN.

```
% crash  
Do you want to shut down the system?  
yes
```

Testinp now compares the file **temp** to the string "yes". The **-f** option causes only the first character of the file and the string to be compared. The **-d** option causes the file **temp** to be deleted after the comparison.

The first character of the file and the first character of the string are the same. Thus, Testinp returns a non-error value to the Shell.

The "if -err" command receives the value returned by Testinp. Because Testinp did not return an error value, the Goto command does not execute. Instead, execution proceeds with Shutdown (line 5).

```
% ty crash.cmd  
echo "Do you want to shut down the system?"  
input > temp  
testinp -fd temp y  
if -err goto noshutdown  
shutdown  
%noshutdown  
exit  
%
```

By combining Input with Testinp, you can communicate interactively with any command file.

The Repeat Command

The Repeat command repeats a command a specified number of times, as in the following example:

```
% repeat 3 echo "this line is displayed three times"  
this line is displayed three times  
this line is displayed three times  
this line is displayed three times  
%
```

Other commands on a command line are not repeated:

```
% repeat 3 echo "Get to work!"; time  
Get to work!  
Get to work!  
Get to work!  
Monday, November 4, 1984 12:57:32
```

The semicolon (;) as a command separator is discussed in chapter 6.

In a command file, Repeat might be used to do multiple "shifts":

```
repeat 3 shift
```

After this command, the fourth command-line argument is equivalent to #1 in the command file.

The Find Command

Note: If you are familiar with the C programming language, the Scan utility is a powerful alternative to Find. Scan does everything Find does, and much more. Consult the Cromix-Plus User's Reference Manual for details.

The Find utility locates files within a file structure. In its simplest form, the Find command syntax is:

```
find directory-pathname -name file-list
```

Find searches the specified directory and all its descendants for one or more files. The keyword "-name" must precede the list of filenames. Ambiguous file references (if applicable) must be enclosed in quotation marks.

This form of the command simply finds files. To do something each time a file is found, you must add some information to the Find command line:

```
-a -exec command
```

"-a -exec" means "and execute the following command"--whatever command follows **-a -exec** executes each time Find locates a file. For example, the following command echoes the phrase "I found another one!" whenever Find locates a file with a **.bak** (for back-up) filename extension.

```
# find / -name "*.bak" -a -exec echo "I found another one!"  
I found another one!  
I found another one!  
I found another one!  
I found another one!
```

Because the directory pathname is /, Find searches the entire Cromix file system (the root directory and all its descendants).

Cromemco Introduction to Cromix-Plus
7. Writing Command Files

If the command following **-a -exec** must act on each file as it is found, add a pair of braces to the command line.

```
# find / -name "*.bak" -a -exec del -v {}
```

As Find locates a file, the Shell substitutes the corresponding filename for the braces. Thus, the preceding command deletes all files with **.bak** filename extensions, one by one. The **-v** (for verbose) option causes Delete to display each filename as it deletes the corresponding file.

Note: Although anyone can use the Find command, only a privileged user should instruct Find to look for files from the root directory. If you are a non-privileged user, Find will search only those directories for which you have execute access.

```
% find / -name "*.bak" -a -exec echo "I found another one!"  
Directory not accessible: "/usr/mail"  
Directory not accessible: "/usr/spool"  
.  
.
```

You can avoid a "Directory not accessible:" message by instructing Find to look for files from your home directory or one of its descendants:

```
% find .. -name "*.bak" -a -exec echo "I found another one!"  
I found another one!  
I found another one!  
.  
.  
.
```

By writing a command file with a similar command, you can periodically rid your directory structure of the numerous backup files created by the Screen program or the Cromemco WriteMaster word-processing program.

```
% ty clean.cmd  
find /usr/you -name "*.bak" -a -exec del -v {}  
% clean  
/usr/you/letter_1.bak  
/usr/you/letter_2.bak  
/usr/you/memo.bak  
.  
.  
.
```


This section discussed only a few of the Find command options. For more information, consult the Cromix-Plus User's Reference Manual.

Sample Command Files

This sample command file illustrates argument substitution, the Shift command, the Repeat command, and the Rewind command.

```
% ty shift_args.cmd
echo #*
shift
echo #1
repeat 2 shift
echo #1
rewind
echo #1
%
```

Executing the file first echoes all arguments. Then, after a shift, the second argument is echoed. After two more shifts, the fourth argument is echoed. A Rewind command cancels all shifts, and the first argument is again echoed.

```
% shift_args 1 2 3 4
1 2 3 4
2
4
1
%
```

The next example combines argument substitution with the Shift, If, and Goto commands:

```
% ty clist_all.cmd
%start
clist #1                % display file
shift
if #1. != . goto start
echo That's all!
```

To execute this file, you might give the following command:

```
% clist_all *.c
```

Before executing the file, the Shell generates the names of files in the current directory with `.c` (for C program) filename extensions. Then, the file executes as follows:

1. Clist displays the file corresponding to the first command argument (#1).
2. The Shift command shifts the arguments by one (#1 is now the second command argument, #2, the third, and so on).
3. The If command tests for another argument. If "#1." is not equal to ".", there is another argument--a filename generated by the Shell. In this case, the Goto command jumps to the label **start**.
4. The process repeats until "#1." (as determined by successive Shift commands) is equal to "."

By taking advantage of the Shell's ability to pass arguments to a command file, you can define options for the command files you write. For example, the following command executes the file `ccomp.cmd` with a user-defined option, `-l` (for link):

```
% ccomp -l *.c
```

"-l" is the first argument. The other arguments are the filenames in the current directory with `.c` (for C program) extensions.

Ccomp, written to compile and (optionally) link C programs, needs to know if "-l" is present so it can call Crolinker (the Cromemco linker) to link the programs to the Cromemco C libraries. During compilation and linking (if applicable), Ccomp must ignore "-l".

The next examples show portions of the file `ccomp.cmd`, illustrating how the Shift and Rewind commands solve the problem.

```
% ty ccomp.cmd
if #1 = -l shift           % get rid of -l option
if #1. = . goto no-args   % if no other arguments
.
.
.
```

The Shift command shifts command-line arguments so "-l" (if present) is ignored. All arguments are now filenames, ensuring a successful compilation.

A Rewind command later in the file reinstates all arguments:

```
.  
.   
.   
rewind   
if #1 != -l exit          % no linking requested   
shift                    % get rid of -l option   
crolinker #* usr/lib/clib /usr/lib/paslib   
.   
.   
.
```

Then, Ccomp tests for "-l" once more. If linking is not required (#1 != -l), the Exit command returns control to the Shell. If linking is required (#1 = -l), the rest of the file executes:

1. The Shift command again gets rid of the option, so the next command (Crolinker) receives only filenames.
2. Crolinker then links all files to the Cromemco C-language libraries.

The file **ccomp.cmd** is shown in its entirety in the Cromix-Plus User's Reference Manual, under the discussion of the Exit command.

Appendix A

THE CROMIX-PLUS COMMAND-LINE EDITOR

The Cromix-Plus command-line editor lets you correct or change a command line with a minimum of retyping. To illustrate, consider the following command line, in which the command name "find" and the keyword "name" have been mistyped. The command-line editor is ideal for corrections like these.

```
# fnd / -nsme ".bak" -a -exec del -v {}
```

While the editor is enabled, the LEFT and RIGHT ARROW keys move the cursor left and right on the command line--without erasing typed characters. Pressing the DELETE key deletes characters to the left of the cursor (one character is deleted each time you press the key).

Pressing CONTROL-I puts the editor in the Insert mode. With the editor in the Insert mode, any characters you type will be inserted to the left of the cursor. To take the editor out of the Insert mode, press ESCAPE. If the editor is not in the Insert mode, any characters you type replace existing characters.

When you are through editing a command line, pressing RETURN sends all visible characters to the Shell. The cursor's position when you press RETURN is therefore irrelevant.

The editor's features are available whenever the editor is enabled. You can determine if the editor is enabled as described in the following section.

Making Sure the Editor is Enabled

The Shell command-line editor is enabled under the following conditions:

- the system knows what type of terminal you are using.
- the terminal is described in the `/etc/termcaps` file.
- the description terminal includes the kR, kI, kE, kI, kr, nb, and nd capabilities (enter `help termcaps` for more information).

Enter the term command to see which terminal you are using:

```
% term  
dumb
```

Cromemco Introduction to Cromix-Plus
A. The Cromix-Plus Command-Line Editor

A "dumb" terminal cannot support the Shell editor.

Currently, the `/etc/termcaps` file describes the following terminals:

C-10	Cromemco C-10 terminal
C-05	Cromemco C-05 terminal
3102	Cromemco 3102 terminal
3101	Cromemco 3101 terminal
T925	TeleVideo TV 925 terminal
A230	Ampex A230 terminal
M100	Mac-Term VT100 terminal emulator
Z100	Zenith VT100 terminal emulator
Z-29	Zenith Z-29 terminal

If your terminal is described in the `/etc/termcaps` file, you can tell the system what kind of a terminal you have with the `Term` command. For example:

```
% term 3102
```

Since the description of the 3102 terminal contains sufficient information for the Shell editor to work, the Shell editor is automatically enabled.

If the system administrator enters your terminal type in the `/etc/ttys` file, you should not need to use the `Term` command.

Retrieving the Previous Command

One of the editor's most convenient features is its ability to retrieve, or "retype," a previous command.

If the editor is enabled, you can easily demonstrate this feature by giving any command to the Shell. When the Shell displays a new prompt, press `CONTROL-R`, as in the following example:

```
% who
betty      qtty1      Nov-14-1984 07:16:22      0      0
jim        qtty2      Nov-14-1984 09:42:29      0      0
fred       qtty3      Nov-14-1984 09:53:47      0      0
% (CONTROL-R)
```

`CONTROL-R` as the first character on a command line redisplay your entire previous command. If `CONTROL-R` is typed somewhere within the command line, any characters remaining in the previous command line will be redisplayed.

Cromemco Introduction to Cromix-Plus
A. The Cromix-Plus Command-Line Editor

```
% who
betty      qtty1      Nov-14-1984 07:16:22      0      0
jim        qtty2      Nov-14-1984 09:42:29      0      0
fred       qtty3      Nov-14-1984 09:53:47      0      0
% who
```

That command may be one the system executed (such as the Who command, above) or a command the system could not execute, as in the next example:

```
# fnd / -nsme ".bak" -a -exec del -v {}
Command not found: "fnd"
# find / -nsme ".bak" -a -exec del -v {}
```

After "retyping" a command in this manner, you can edit the command or give it again by pressing RETURN.

Appendix B

THE CROMEMCO SCREEN PROGRAM-ENTRY EDITOR

This appendix describes the Cromemco Screen program-entry editor. Discussed in detail are the basic editing modes and some of the editor's most important commands.

The illustrations in this appendix show a simple computer program; however, you can also use the editor to prepare reports, correspondence, or any kind of text. As you read the appendix, you may wish to create and edit your own file. If this file contains five or six lines of text, and a few intentional errors, you can easily learn the editor's most important features. The illustrations in the section "Basic Editing" show what might be displayed on the terminal screen during a practice editing session.

When you're comfortable with the program's editing modes, you can experiment with some of the editor's more advanced features. These are described in the command summary at the end of this appendix.

There is additional information about the Screen program in the Cromix-Plus User's Reference Manual.

Selecting the Editor to Create a New File

When you wish to use the editor to create a file, give the Screen command with a new filename as an argument. The name you choose for the new file must observe the file-naming conventions discussed in chapter 3.

This command calls the Screen editor to create a file named **sample**:

```
% screen sample
```

The editor is ready to use when the following command line is displayed at the top of the terminal screen:

```
>Edit: At Copy Delete Exit Find Insert Jump Move Other Page Substitute Xchange
```

When the editor's command line is visible, the editor is in Command mode. From the Command mode, you can give any of the editor's available commands.

Giving Screen Commands

The preceding command line lists most of the editor's commands. A second command line, which can be viewed using the Other command (refer to the command summary), lists the remaining commands.

Before you can give a command to the editor, the editor must be in the Command mode (with either command line displayed). If it is, you can give any command by typing the first letter of the command name.

The ESCAPE Key

Following execution, some commands automatically return the editor to the Command mode -- other commands do not. For those commands that do not, you must return the editor to the Command mode yourself before you can give a new command. To do so, press the ESCAPE key.

Entering Text in the Insert Mode

When you enter text in a new file or add text to an existing file, you will use the editor's **Insert mode**. Insert mode is selected by giving the Insert command.

Typing **I** for Insert replaces the command line with:

```
>Insert: <text...> <ESCAPE>
```

indicating you can either enter text or return to the Command mode by pressing the ESCAPE key.

As you enter text, press the RETURN key to end each line. If you notice an error a short distance from the cursor, use the LEFT ARROW key or the DELETE key to back up so you can correct it. Pressing either of these keys deletes text to the left of the cursor, a character at a time. If you prefer, ignore any mistakes you make--you can correct them later.

When you are through entering text, press the ESCAPE key to return the editor to the Command mode.

Shown here is a program that was entered in the Insert mode.

```
>Insert: <text...> <ESCAPE>  
10 Strig$="26.6321"  
20 Increment=1  
30 Value=Incremental+Val(String$)  
40 Print Value  
50 End  
□
```

As you can see, there are several errors. Later sections show how to correct these errors using the Insert mode and the other basic editing modes, **Delete** and **Xchange** (exchange).

Basic Editing

For basic text editing, you need a knowledge of only four commands--Insert, Delete, Xchange (exchange), and Exit. The Insert, Delete, and Xchange commands select the editing modes discussed in detail in the following sections.

Before you select an editing mode, the editor must be in the Command mode. (If you're not sure that it is, press the ESCAPE key; the keyboard will beep if the editor is already in the Command mode.) Then use the ARROW keys (or any of the cursor movement keys described in the next section) to move the cursor to the location of the change, and give the command that selects the appropriate mode.

When you wish to exit from one mode to select another, press the ESCAPE key to return the editor to the Command mode. Then give another command.

Moving the Cursor While the Editor is in the Command Mode

If the text in a file exceeds the capacity of the terminal screen, and the cursor is on the bottom screen line, pressing the RETURN key or the DOWN ARROW key brings additional text, a line at a time, onto the terminal screen. This process is called **scrolling**.

The > symbol at the left of the command line is the **cursor direction indicator**. With the indicator pointing to the right, the cursor moves to the right when you press the SPACE bar or the TAB key. When you press the RETURN key, the cursor moves down. Sometimes you may want the cursor to move to the left or up when you press these keys. For example, with the cursor at the end of a file, you may wish to use the RETURN key to scroll up through the file. To reverse cursor movement, type < while the editor is in the Command mode. To return the setting to forward, type >.

The cursor direction indicator has no effect on the ARROW keys. Pressing an ARROW key while the editor is in the Command mode always moves the cursor in the direction shown on the key.

You can move the cursor a specific number of lines by typing a number and pressing the RETURN key. For example, typing 15 and pressing RETURN moves the cursor 15 lines in the file. The cursor moves up or down in the file as determined by the cursor direction indicator. To move the maximum number of lines (65335), type # and press RETURN.

The Jump, Page, and Home commands also move the cursor, as described in the command summary at the end of this appendix.

Insert Mode--the Insert Command

In this example, the cursor is positioned to correct the word "String" by inserting the letter "n".

```
>Edit: At Copy Delete Exit Find Insert Jump Move Other Page Substitute Xchange
10 String$="26.6321"
20 Increment=1
30 Value=Incremental+Val(String$)
40 Print Balue
50 End
```

After selecting the Insert mode by giving the Insert command, space is allotted for the insertion, as shown in the next example. To make the change, type n, and then press the ESCAPE key to return the editor to the Command mode.

```
>Insert: <text...> <ESCAPE>
10 Stri  
      g$="26.6321"
20 Increment=1
30 Value=Incremental+Val(String$)
40 Print Balue
50 End
```

While you are editing a file in the Insert mode, you can add as much new text (including spaces or blank lines) as you like. To return to the Command mode, press the ESCAPE key.

Delete Mode

The Delete mode is selected by giving the Delete command. While the editor is in the Delete mode, pressing any of the following keys will delete text.

SPACE Bar--deletes a character at a time, in the direction shown by the cursor direction indicator.

RETURN Key--deletes a line at a time, in the direction shown by the cursor direction indicator.

TAB Key--deletes text between TAB stops, in the direction shown by the cursor direction indicator.

ARROW Keys--delete text in the direction shown on the key. The LEFT and RIGHT ARROW keys delete a character at a time, while the UP and DOWN ARROW keys delete entire lines of text.

Cromemco Introduction to Cromix-Plus
B. The Cromemco Screen Program-Entry Editor

To delete a specific number of lines after giving the Delete command, type a number and press the RETURN key. For example, typing 5 and pressing RETURN while the editor is in Delete mode deletes five screen lines.

Until you leave the Delete mode by pressing the ESCAPE key, deleted text can be easily recovered by pressing the ARROW keys. To illustrate, assume the cursor direction indicator is pointing to the right, and you press the RETURN key to delete one or two lines of text. If you wish to recover this text, pressing the UP ARROW key restores it, line by line.

The next change to be made to the sample file is to correct the word Increment by deleting the letters al. In the following example, the cursor is positioned for the correction.

```
>Edit: At Copy Delete Exit Find Insert Jump Move Other Page Substitute Xchange
10 String$="26.6321"
20 Increment=1
30 Value=Increment+Val(█(String$))
40 Print Balue
50 End
```

To make this change, select the Delete mode by typing D and then press either the SPACE bar or the RIGHT ARROW key twice to remove the unwanted letters.

Whenever you are through deleting text, press the ESCAPE key to return the editor to the Command mode.

Xchange (Exchange) Mode

To replace a character by typing over it, move the cursor to the location of the change and select the editor's **Xchange mode**. To do so, give the Xchange command by typing X. The next example shows the cursor positioned to correct the word "Value" in the sample file. Here, the letter "V" must be exchanged for the incorrect letter, "B".

```
>Edit: At Copy Delete Exit Find Insert Jump Move Other Page Substitute Xchange
10 String$="26.6321"
20 Increment=1
30 Value=Increment+Val(String$)
40 Print █alue
50 End
```

While the editor is in the Xchange mode, you can type over characters, words, or entire lines of text. Text that has been typed over cannot be recovered.

Saving a File on Disk--the Exit Command

When you have edited a file, use the Exit command to **update** the file on the disk. "Updating" (or "saving") a file means:

1. If you are entering text in a new file, that text is stored for the first time under the filename you chose when you first selected the editor.
2. If you are editing an existing file, the edited version replaces the previous version of the file on the disk. (The previous version is stored as a back-up file, with the filename extension **.bak.**)

With the editor in the Command mode, typing E for **Exit** displays the command options shown below.

```
>Exit: Continue Quit Update <ESCAPE>
      Continue - update filename and continue editing
      Quit - exit without updating filename
      Update - update filename and exit
      <ESCAPE> - return to editor
```

As shown, there are four options: 1) **Continue** to update the file and return to the editor; 2) **Quit** to exit from the editor without updating the file; 3) **Update** to update the file and then exit from the editor; and 4) **ESCAPE** to return to the Command mode.

Except for **ESCAPE**, command options are selected by typing the first letter of the option name. For example, if you are entering text in a new file and wish to save the file on disk and then continue entering text in that file, you would:

1. Type **E** to give the Exit command.
2. Type **C** to select the Continue option.

Giving the Exit command, and then selecting the Continue option, stores the file on the disk and returns the editor to the Command mode. (The cursor will be where it was when you gave the Exit command.)

After you give the Exit command with the Update option, you are no longer using the Screen program. The Screen program returns control to the Shell after saving the file on the disk.

```
>Exit: Update
34 bytes written to: sample
%
```

As shown, the Screen program displays the size of the updated file (in bytes) before returning control to the Shell.

To avoid losing text (for example, there may be an unexpected power failure), you should update files regularly. Select either the Continue or Update option, depending on whether or not you wish to continue using the editor.

The Quit option aborts an editing session restoring the file to the version at the beginning of the session or the last use of the Continue option during this session. Quit is useful if you make a mistake while editing and wish to begin again, or if you have edited a file and decide you prefer the unedited version.

Editing a File that is Stored on the Disk

An existing file can be displayed for editing by giving the Screen command with the name of that file as an argument:

```
% screen letters
```

If you have a file named **letters** in your directory, the preceding command will display the file for editing.

Setting Markers with the At Command

To execute certain commands, the editor requires reference points. For example, if you wish to move a block of text to a new location in a file (the Move command), you need a way of marking the text to be moved.

You can set a marker at any point in a file by moving the cursor to the location to be marked and then giving the At command. After typing A or @ for **At**, the command line is replaced by:

```
set marker At: 1 2 3 4 5 6 7 <ESCAPE>
```

If there are markers already set, their numbers will be flashing. To set a marker, type any number from 1 to 7. (Typing a number that is flashing resets that marker.)

The commands that use markers are: Beautify, Copy, Jump, List, Move, Write, and Zap (refer to the command summary).

Whenever you give a command that uses markers, the names of the editor's preset markers (Begin, Cursor, End) will always be displayed. To their right will be the numbers of any markers you've set using the At command. The preset markers let you perform command functions in relation to the beginning of the file, the cursor position, or the end of the file. To select these markers, type the first letter of the marker name.

The Screen Editor's Shell Command

The editor's Shell command (%) lets you temporarily exit from the Screen program so you can give a command to the Cromix-Plus Shell. To give the command, type a percent sign (%) with either of the editor's command lines displayed.

After you type %, the editor's command line (and the text you were editing) is replaced by the following display:

```
>Shell:

EXit will return to Screen
%
```

At this point, you can give one or more commands to the Shell. You can even give the Screen command to edit or create a file. When you want to resume editing, give the Exit command (the same command that normally logs you out).

```
% ex
```

If you prefer type CONTROL-Z instead of giving the Exit command. (CONTROL-Z cannot log you out by accident.) Whichever method you use, the result is the same. Your file will be redisplayed exactly as it was when you gave the Shell command.

For more information about CONTROL-Z in place of Exit, refer to the discussion of the Shell command in the Cromix-Plus User's Reference Manual.

Note: If, after giving the Shell command, you subsequently give the Screen command to edit a file, you may see the following message:

```
% screen letters
Cromix Screen editor version xx.xx
File in use: "letters"
```

This message means you are using the file. You were editing this file when you gave the editor's Shell command. To redisplay the file and return to the editor, give the Exit command.

Considerations When Using the Editor

Whenever you give the Screen command, the command argument is the pathname of a file. You can use a relative or absolute pathname, as you would with any other Cromix-Plus utility that uses filenames. For example, the following

command uses an absolute pathname to create a file named **summary** in a user's home directory:

```
% screen /usr/betty/summary
```

Or, this command calls the Screen program to edit a file named **stock_on_hand** in the parent of the current directory.

```
% screen ^stock_on_hand
```

Although you may decide it's easiest to change directories before giving the Screen command, you do not have to change directories to edit a file.

Some of the redirection techniques discussed in chapter 6 also work with the Screen program. For example, the following command calls the Screen program to edit a file named **notes**, using editing commands from the file **edit**:

```
% screen notes < edit
```

This variation of the same command redirects the output from the Screen program to a null device (the file **/dev/null** is discussed in chapter 7):

```
% screen notes < edit > /dev/null
```

Because editing commands are being given from the **edit** file, there is little reason to watch the editing on the terminal screen. Redirecting Screen's output to **/dev/null** discards that output--without displaying it or saving it in a file.

COMMAND SUMMARY

- COMMAND:** At
Purpose: Sets markers the editor can use for reference while executing certain commands.
- Summary:** A marker-number
- Arguments:** Marker number from 1 to 7.
-
- COMMAND:** Beautify
Purpose: Formats text files. The default is aligned text (right margin ragged as though typed).
- Summary:** B [jm] markers
- Arguments:** Two markers.
- Options:** j justifies text.
m displays margin settings so they can be changed before formatting text. Type the first letter of the setting you wish to change (for example **R** for **Right Margin**), and type the new setting. Then press RETURN to enter it.
-
- COMMAND:** Copy
Purpose: Copies text designated by markers to the cursor position.
- Summary:** C markers
- Arguments:** Two markers.
-
- COMMAND:** Delete
Purpose: Selects the editor's Delete mode.
- Summary:** D
-
- COMMAND:** Exit
Purpose: Exits from program while (optionally) updating file on the disk. Or, updates file on the disk and returns the editor to the Command mode.
- Summary:** E [cqu]
- Options:** C continue -- update file and return to the Command mode.
Q quit -- exit from program without updating file.
U update -- exit from program and update file.

COMMAND: Find
Purpose: Locates text strings. The editor searches the file in the direction shown by the cursor direction indicator. The search begins with the first character after the cursor when searching down in a file. When searching up in a file, the search begins with the first character before the cursor.

When a string is found, you can edit the text and then find the next occurrence of the string without retyping it, as described in Note 4 below.

Summary: [n] F ;string;

Arguments: ; a delimiter character to open and close the string. A delimiter can be any nonalphanumeric character (commonly used delimiters are the slash mark (/), the period, the colon, the semicolon, or a SPACE).

string a sequence of characters (including TABs, RETURNS, or SPACES). A string may be ambiguous, using the asterisk (*) and the question mark (?). The asterisk stands for any number of characters, while the question mark stands for a single character. In this way, entering the string:

/oc*/

might find "occur," "socket," "octal," etc., while entering the string:

/th?n/

might find "thin," "both now," "than," or "then."

Options: n the occurrence of the string, from the cursor, to be found. (One is the default.) For example, if you type 5, the editor will find the fifth occurrence of the string, 10, the tenth, and so on.

- Notes:**
1. When a string is found, the cursor moves to the first character of the string. If the string is not found (you may have mistyped it), the cursor remains where it was when you gave the command.
 2. If you do not specify a particular occurrence to find by typing a number, the editor looks for the first occurrence, from the cursor, in the direction shown by the cursor direction indicator.
 3. The Find command ignores letter case unless you use quotation marks as delimiter characters.

Cromemco Introduction to Cromix-Plus
B. The Cromemco Screen Program-Entry Editor

4. After finding one occurrence of the string, you can find the next without retying the string. To do so, give the Find command again by typing F and then type any two delimiter characters. You may find it easiest to press the SPACE bar twice.

If you wish the editor to obey letter case (for example, you are searching for all instances of "AND," and wish to bypass "and" or "And"), type two quotation marks as delimiters.

5. You must type the same delimiter character to open and close a string. If the string is ambiguous (containing * or ?), do not use * or ? as delimiters.
6. If you wish to treat ? or * in a string as a standard, printing character, press the \ key twice to type \ before you type either ? or *. For example:

:file\?:

will find "file," followed by "?", while

:file?:

will find "file," followed by any character.

7. If the Find command does not seem to be working as described: 1) check the setting of the cursor direction indicator; 2) check the position of the cursor in the file (the cursor may be past the string); or 3) make sure you opened and closed the string with the same delimiter character.
8. To abort the command while it is executing, press CONTROL-C.

COMMAND: Home
Purpose: Moves the cursor to the top left hand corner of the terminal screen.

Summary: H

COMMAND: Insert
Purpose: Selects the editor's Insert mode.

Summary: I

COMMAND: Jump
Purpose: Moves the cursor directly to a marker, such as the beginning or end of a file, without scrolling text.

Summary: J marker

Cromemco Introduction to Cromix-Plus
B. The Cromemco Screen Program-Entry Editor

Arguments: A marker.

The Jump command is especially useful if you cannot remember where you set markers in a file. After giving the command, type the number of the marker whose position you wish verified. The cursor will move to the location of that marker.

COMMAND: List
Purpose: Prints text between markers on printer.

Summary: L markers

Arguments: Two markers.

COMMAND: Move
Purpose: Moves text designated by markers to the cursor position.

Summary: M markers

Arguments: Two markers.

COMMAND: Other
Purpose: Displays second command line.

Summary: O

COMMAND: Page
Purpose: Moves the cursor directly to another page in the file. A "page" is one screen (23 lines) of text. Whether the cursor moves forward or backward depends on the setting of the cursor direction indicator.

Summary: [n] P

Options: n a number. The specific page--from the cursor position--to be displayed (one is the default).

COMMAND: Read
Purpose: Copies another file into the file being edited, inserting it at the cursor position.

Summary: R filename

Press the RETURN key to give the command.

Arguments: The name of another file on the disk.

Cromemco Introduction to Cromix-Plus
B. The Cromemco Screen Program-Entry Editor

COMMAND: Shell
Purpose: Temporarily exits from the Screen program.

Summary: %
To return to the Screen program, give the Exit command in response to the Cromix-Plus Shell prompt.

COMMAND: Substitute
Purpose: Finds and replaces text strings, in the direction shown by the cursor direction indicator.

Summary: [n] S [q] :old-string: :new-string:

Arguments: : delimiter character (the convention for delimiting strings is the same as with the Find command).
old-string the string to be replaced. The old text string may contain ambiguous characters (see the Find command).
new-string the replacement string.

Options: n a number (the number of times to find and replace a string). One is the default. To select the maximum number of replacements, type #.
Q (query) Instructs the editor to ask for confirmation before replacing a string. If you do not select this option, the specified number of strings will be automatically replaced.

Notes: 1. If you wish to abort the substitution process, press CONTROL-C.
2. If you select the Query option, by typing Q after giving the Substitute command, the editor will ask you to confirm each substitution. Type Y for Yes, N for No, or press the ESCAPE key to return the editor to the Command mode.
3. Use care when substituting ambiguous text strings, especially when using the asterisk to denote a string of undetermined length. The string you enter might match one or two characters, several lines, or entire paragraphs.
4. To insert the same string n times along the left or the right margin, Substitute using the RETURN key (^M) in the strings.
[n] [S]/^M/,/@b^M/ will insert @b at the end of n lines of text.

[n] [S]/^M/,/^M@b/ will insert @b at the beginning of n lines of text.

COMMAND: Tabs
Purpose: Displays the editor's TABs scale (each TAB stop on the scale is indicated by the letter T).

Summary: T

- Notes:**
1. To set new TAB stops after giving the Tabs command, press the SPACE bar to move the cursor to the proper location on the scale--then type T.
 2. To erase TAB stops, move the cursor over the "T" on the scale by pressing the SPACE bar.
 3. To enter changed TAB stops, press the RETURN key (do not press the ESCAPE key unless you want to cancel the command).

COMMAND: Verify
Purpose: Refreshes screen display, clearing the screen of any characters that are not a part of the file.

Summary: V

COMMAND: Write
Purpose: Writes all or a portion of the current file to a new file on the disk.

Summary: W markers new-filename

Press the RETURN key to give the command.

Arguments: Two markers

A new filename (using the file-naming conventions explained in chapter 3).

COMMAND: Xchange
Purpose: Selects the editor's Xchange mode.

Summary: X

Cromemco Introduction to Cromix-Plus
B. The Cromemco Screen Program-Entry Editor

COMMAND: Zap
Purpose: Deletes text between markers.

Summary: Z markers

Arguments: Two markers.

Text deleted with the Zap command cannot be recovered, as can text deleted in the editor's Delete mode. If you make a mistake using the Zap command, consider giving the Exit command, and then selecting the Quit option, so you can begin the editing session again.

Cromemco Introduction to Cromix-Plus
Index

- .bak filename extension, 16
- .bin filename extension, 16, 30, 54
- .cmd filename extension, 16, 30, 54, 83
- .com filename extension, 16, 30, 54
- .reminder file, 55
- .save filename extension, 16
- .startup.cmd file, 55

- /, root directory, 35
- /bin directory, 36, 54
- /cmd directory, 54, 83
- /dev directory, 56
- /dev/null, 85, 113
- /usr directory, 36
- /usr/mail directory, 40, 60

- Absolute pathname, 36, 37, 39, 44
- Absolute pathname, displaying, 41
- Access privileges, for owner of a file, 21
- Ambiguous file reference, 78
- Ambiguous file references, 80, 87
- Appending output to a file, 70
- Argument passing, 87
- Argument substitution, 86
- Argument, testing for, 90, 91
- Arguments, shifting, 95

Background processing, 73

- Calling a program, 6
- Changing directories, 43
- Clist utility, 22
- Command arguments, 7
- Command file, 30
- Command file structure, 86
- Command file, communicating interactively with
94
- Command file, executing, 30, 83
- Command file, testing for an argument, 90
- Command file, use of the asterisk in, 87
- Command file, user-defined option in, 98
- Command files, 83
- Command files, examples, 97
- Command line arguments, reinstating, 91
- Command line arguments, shifting, 91
- Command line, editing, 101
- Command line, redisplaying the previous, 102
- Command line, use of parentheses, 75
- Command syntax, Copy, 23, 46, 47

Cromemco Introduction to Cromix-Plus
Index

Command syntax, Cptree, 53
Command syntax, D command, 43
Command syntax, Delete, 25
Command syntax, Deltree, 52
Command syntax, Help, 8
Command syntax, introduced, 7
Command syntax, Makd, 45
Command syntax, Move, 46, 47
Command syntax, Rename, 25
Command syntax, Spool, 27, 28
Command syntax, Type, 22
Command syntax, Who, 8
Command, defined, 5
Command, editing, 5
Command, sequential, 74
Command-line editor, 101
Command-line editor, enabling/disabling, 101
Command-line editor, Insert mode, 101
Conditional jumps, 89
CONTROL-C, 17, 28, 60, 74
CONTROL-C, abort signal, 9
CONTROL-C, to stop program execution, 9
CONTROL-I, command-line editor, 101
CONTROL-P, to retrieve the previous command, 102
CONTROL-Q, 22
CONTROL-Q, to display additional screens, 9
CONTROL-U, as "line-kill" key, 5
CONTROL-Z, end-of-file, 60, 71, 73
Conventions, 5
Copy command, 46, 47
Copy command, -f option, 26
Copy utility, 23
Copying files to another directory, 46
Cptree command, -v option, 54
Cptree utility, 53
CRDEV setting, Mode utility, 10
Creating a directory file, 45
Creating a file, 17
Current directory, 49

D command, 36
D command, to change directories, 43
Del command, 51
Delete command, 25
DELETE key, 4
Deleting a file, 25
Deleting text while using the Screen editor, 108
Deltree utility, 51
Destination file, 23
Detached process, 73
Device, 56
Device file, creating, 57

- Device files, 56
- Directories, changing, 43
- Directories, shortcuts for specifying, 49
- Directory, 33
- Directory file, 35
- Directory, ancestor, 44
- Directory, copying, 53
- Directory, creating, 45
- Directory, deleting, 51
- Directory, descendant, 44
- Directory, destination, 53
- Directory, displaying the name of the current
36
- Directory, execute access, 96
- Directory, execute access for, 40
- Directory, home, 33
- Directory, mail, 40
- Directory, parent, 36, 40
- Displaying a file with page headings, 22
- Displaying a list of files, 18
- Displaying a list of system users, 6
- Displaying invisible filenames, 56
- Displaying the contents of a file, 22
- DOWN ARROW key, to enter command, 10

- Echo command, 94
- Echo command, example in command file, 93, 94
95, 97
- Echo utility, 85
- Echoing special characters, 86
- Editing a command line, 101
- Editor, command-line, 101
- Error message, "Directory file", 35
- Error message, "Directory not accessible", 40
96
- Error message, "File already exists", 26
- Error message, "File in use", 112
- Error message, "File not found", 26, 27, 40
- Error message, "Illegal filename", 30
- Error message, "Must be privileged user", 13
- Error message, "Program not found", 10
- Error message, "Wrong number of arguments", 26
- Error messages, introduced, 10
- Error messages, redirecting, 75
- Error messages, redirecting to a null device, 85
- Error messages, some command, 26
- Error return value, 92
- ESCAPE key, to select Screen editor's Command
mode, 106
- Execute access for a directory, 96
- Exit command, 12
- Exit command, example in command file, 99

Cromemco Introduction to Cromix-Plus
Index

Exit command, to log out, 12
Exit command, to return an error value, 91
Exit command, to return control to the Shell, 91
Exit command, to return to Screen editor, 112

File, accidentally deleting, 70
File, copying, 23
File, copying to a directory, 46
File, creating, 17
File, creating with Screen program, 105
File, deleting, 25
File, device, 56
File, directory, 35
File, displaying the contents of, 22
File, displaying with page headings, 22
File, moving to a directory, 46
File, printing, 27
File, printing more than one copy, 27
File, renaming, 25, 47
Filename extension, 16
Filename generation, 78, 80
Filenames, displaying, 18
Filenames, invisible, 55
Files, naming, 15, 16
Find command, 95
Find command syntax, 95

Goto command, 88
Goto command, example in command file, 98

Help utility, 8, 9
Home directory, 33, 49
Home directory, making the current directory, 51

If command, 89
If command, example in command file, 93, 94, 97
98, 99
If command, relational operators used with, 90
If-else-endif, 90
If-err, 89, 94
If-goto, 89
Inode number, 46
Input command, 92
Input command, example in command file, 93, 94
Input file, 72
Inserting text while using the Screen editor
108
Intrinsic commands, 12
Invisible filenames, 55

Invisible filenames, displaying, 56

Jumping to a label, 88

Jumps, conditional, 89

Kill utility, to stop a background process, 74

Killing a job in the print queue, 28

Label, jumping to, 88

Logging in, 4

Logging in, defined, 3

Logging out, 12

Login name, 3, 6

Login prompt, 4

Ls command options, 18

Ls command, -a option, 56

Ls command, -e option, 19

Ls command, -i option, 19

Ls command, -l option, 18, 21

Ls command, -m option, 18

Ls command, with directory pathname as argument
38

Ls utility, 18, 37

Mail utility, 59, 66

Mail utility, correcting mistakes while using
60

Mail utility, redirecting input, 72

Mail, reading, 61

Mail, sending, 59

Mail, sending to several users, 63

Makd command, 45, 53

Makdev utility, 57

Match utility, 93

mbox file, 62

Message of the day, 5

Mode settings, 101

Mode utility, 9, 10, 11

Move utility, 46, 47

Moving files, 46

Naming files, 15, 16

Null device, 85, 113

On-line manual, 8

Options, defining in a command file, 98

Options, grouping after one hyphen, 20

- Options, requiring arguments, 21
- Output file, 68, 70
- Output, redirecting to a file, 68

- PA setting, Mode utility, 9
- Parent directory, 36, 49
- Parentheses on a command line, 75
- Passwd utility, 11
- Password, 3
- Password prompt, 5
- Password, changing, 11
- Path command, 41
- Pathnames, used when calling Screen editor, 112
- Peripheral device, 56
- PID number, 73
- Pipes, 76
- Print queue, 27
- Printing a file, 27
- Printing, cancelling, 28
- Printing, listing the jobs in the queue, 27
- Printing, stopping, 28
- Privileged access, 5, 13
- Privileged user, 5, 13
- Process identification number, 73

- qTTY number, 6
- Query utility, 9
- Queue, 27
- Queue, listing the jobs in, 27
- Quotation marks, on a command line, 86
- Quotation marks, to enclose a multiword string
93
- Quoting special characters, 86

- Reading mail, 61
- Redirecting and appending output, 70
- Redirecting error messages, 75
- Redirecting input from a file, 72
- Redirecting output to a file, 68
- Redirecting output to a null device, 85, 113
- Redirecting output to a temporary file, 77
- Redirection with pipes, 76
- Redirection within a command file, 85
- Redirection, when using Screen editor, 113
- Relational operators used with If command, 90
- Relative pathname, 42
- Rename command, 25
- Renaming a file, 25, 47
- Repeat command, 94
- Repeat command, example in command file, 97

Repeating a command, 94
RETURN key, 4, 6
Return values, 89, 91, 92
RETYPE mode setting, 101
Rewind command, 91
Rewind command, example in command file, 97, 99
Root directing, finding files from, 96
Root directory, 35

Scan command, 97 .
Screen editor, 105
Screen editor, basic editing techniques, 107
Screen editor, command mode, 106
Screen editor, command summary, 114
Screen editor, correcting mistakes, 106
Screen editor, creating a new file, 105
Screen editor, cursor direction indicator, 107
Screen editor, Delete mode, 108
Screen editor, entering text in Insert mode, 106
Screen editor, giving commands to the, 106
Screen editor, Insert mode, 106, 108
Screen editor, moving the cursor, 107
Screen editor, redirecting input and output, 113
Screen editor, saving a file on disk, 110
Screen editor, setting markers with At command
111
Screen editor, temporarily exiting from the
program, 112
Screen editor, to edit an existing file, 111
Screen editor, use of the ESCAPE key, 106
Screen editor, Xchange mode, 109
Screen utility, 17
Sending mail, 59
Sequential commands, 74
Sequential processing, 74
Shell, 65
Shell characters with special meaning, 17
Shell commands, 12
Shell prompt, 5
Shell, search for program or command, 54
Shift command, 91
Shift command, example in command file, 97, 98
99
Shifting arguments, 95
Shifting command line arguments, 91
Shortcuts for working within a directory
structure, 49
Shutdown utility, 93
Sort utility, 67
Source, 53
Source file, 23
Special Shell characters, 17, 86

Cromemco Introduction to Cromix-Plus
Index

Spool command, -a option, 28
Spool command, -k option, 27
Spool command, -l option, 27
Spool command, -m option, 27
Spool utility, 27
Spool utility, important consideration, 74
Spool utility, using Kill with, 74
Standard Error, 75
Standard input, 66
Standard output, 66
STDERR, 75
STDIN, 66, 67, 71, 92
STDOUT, 66, 67, 92
Summary of Screen editor commands, 114
System administrator, 3

Tee utility, 77
Terminal keyboard, 3
Testing for an argument, 90, 91
Testinp command, 92
Testinp command, example in command file, 93, 94
Testinp, -d option, 92
Testinp, -f option, 92
Testinp, the -d option, 94
Testinp, the -f option, 94
Text editor, 105
Time command, 94
Time utility, 7
Type command, 22
Type command, redirecting output, 71

Who utility, 6, 7, 66, 68

Reader Responses To This Documentation

Dear Reader,

We have made a sincere effort to provide you with the information you need in this manual. If you should find the documentation deficient or in error, let us know so we can correct it. We appreciate and value your response; it will be useful in improving the documentation. Please detach and use the Reader Response Card below to send us your comments.

Thank you for your time and interest in Cromemco products.

Sincerely,

Winthrop A. Stiles III
Technical Publications Manager

(Detach Here)

Cromemco[®]

Reader Response Card

To: Winthrop A. Stiles III,
Technical Publications Manager

Re (Manual title): _____

My System is (Specify configuration): _____

The following information is incorrect (Please specify page number): _____

(Detach Here)

(Fold Here)

The following additional information would be helpful: _____

What general suggestions do you have for improving this manual? _____

If you need a response from Cromemco, please print your name, mailing address, and telephone number:

Name: _____

Address: _____

Telephone: () _____



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 599 MOUNTAIN VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Cromemco[®]

Attn: Winthrop A. Stiles III
Technical Publications Manager
280 Bernardo Avenue
P.O. Box 7400
Mountain View, CA 94039



Cromemco[®]

280 Bernardo Ave.
P.O. Box 7400
Mountain View, CA 94039
