Burroughs

XE 500 CENTIX™

Operations Guide

Volume 1: Basic Operations

Relative To Release Level 6.0
Priced Item
November 1986

Distribution Code SA
Printed in U S America
1207768

Burroughs

XE 500 CENTIX™

Operations Guide

Volume 1: Basic Operations

Relative To Release Level 6.0
Priced Item
November 1986

Distribution Code SA
Printed in U S America
1207768

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Burroughs, if any, with respect to the products described in this document are set forth in such License or Agreement. Burroughs cannot accept any financial or other responsibility that may be the result of your use of the information or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the Product Improvement Card at the back of this manual, or remarks may be addressed directly to Burroughs Corporation, Corporate Product Information East, 209 W. Lancaster Ave., Paoli, PA 19301, U.S.A.

About This Guide

Purpose

This guide presents basic information to help users understand how the CENTIX™ operating system works, and to help users begin working on the operating system.

Scope

This guide describes information needed to begin using the CENTIX operating system. It presents only concepts that are necessary to begin understanding the operating system. For complete documentation on CENTIX operating system capabilities, refer to your *CENTIX Operations Reference Manual*.

Audience

The audience for this guide is the beginning user. More experienced users should refer to the *CENTIX Operations Reference Manual* and the *CENTIX Programming Guide*.

Prerequisites

The user who reads this guide should be familiar with the computing environment in which he or she will be working.

How to Use This Document

Use this guide to learn how to begin using the CENTIX operating system. For more detailed information on the operating system, see the *System Software Operations Reference Manual*.

Organization

This guide contains the following sections:

Section 1, Introduction, describes what the operating system is and how it works.

Section 2, Getting Started, describes how to begin interacting with the operating system.

Section 3, Using the File System, introduces the operating system structure and shows where users fit in.

Section 4, Using the Shell, describes the tool by which users interact with the operating system. The second half of this section provides information on programming with the shell.

Section 5, Printing Files, discusses how to print files on a line printer.

Section 6, Using centreWINDOW (CENTIX Only), describes the CENTIX terminal screen window management utility.

Section 7, Using Removable Media, describes the different types of removable media you can use to transfer information on the operating system.

A glossary follows Section 7.

Related Product Information

XE 500 CENTIX Administration Guide

This guide discusses how to administer the XE 500 CENTIX operating system.

XE 500 CENTIX centrEASE Operations Reference Manual

This manual describes how to use the CENTIX administrative facility, centrEASE.

XE 500 CENTIX C Language Programming Reference Manual

This manual describes the programming language on which the CENTIX operating system is structured, C.

XE 500 CENTIX Programming Guide

The guide discusses how to program on the XE 500 CENTIX system.

XE 500 CENTIX Operations Reference Manual

This manual lists and describes all CENTIX shell commands, system calls, library functions, and special files.

Conventions Used in This Guide

- All commands within text are shown in boldface.
- Variables are shown in italics. For example, in the following command, *oldfile* and *newfile* are both variables:

  $ cp *oldfile newfile*

  When you enter the actual command, you substitute the names of the file that you are copying and the file to which you are copying for *oldfile* and *newfile*.
- In command lines, optional fields are enclosed in brackets.

Contents

Illustrations

Tables

Introduction

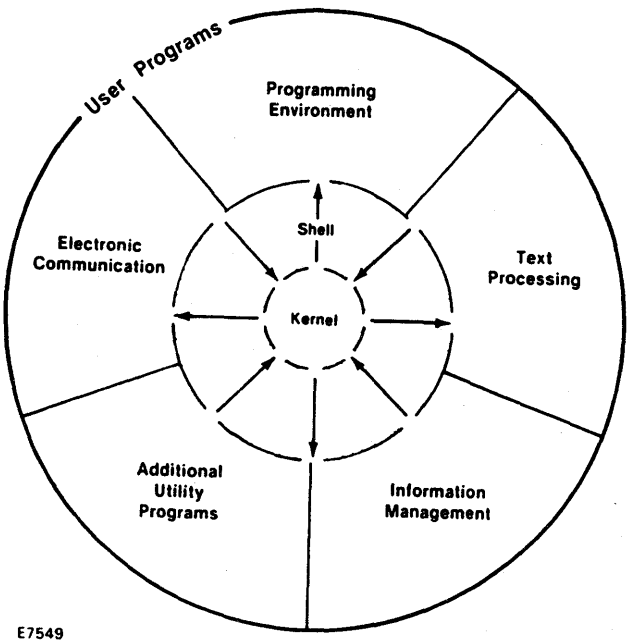
This guide describes information needed to begin using the XE 500 CENTIX operating system. The CENTIX operating system is based on UNIX™ System V and is licensed from AT&T.

This guide presents only those concepts that are necessary to begin understanding the operating system. For complete documentation on CENTIX operating system capabilities, refer to your *CENTIX Operations Reference Manual.*

CENTIX System

The CENTIX system is a set of programs, called the operating system, that acts as a link between you and the XE 500 computer.

Figure 1-1 **Operating system model**



™UNIX *is a trademark of AT&T Bell Laboratories*

Within the operating system itself, there are three major components of system software, working together, that allow you to communicate with the computer. They are:

- The kernel.
- The shell.
- Programs that run on command.

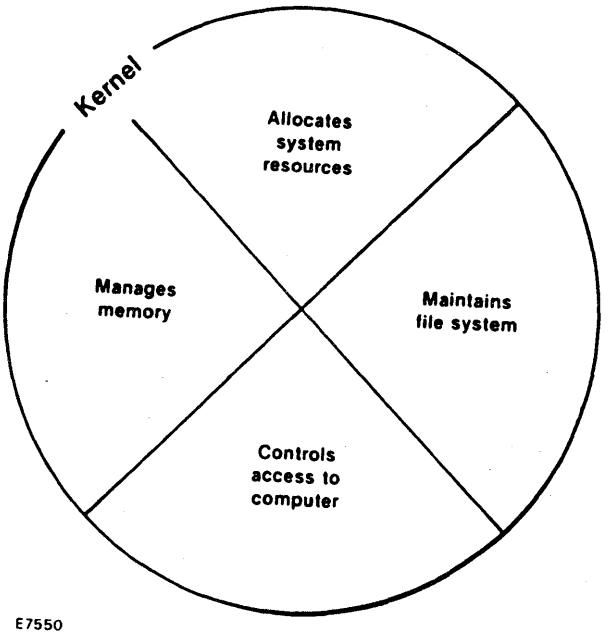
Figure 1-1 shows a model of the CENTIX operating system.

The Kernel

The heart of the CENTIX operating system is called the kernel. Figure 1-2 gives an overview of the kernel's activities.

The kernel is software that controls access to the computer, manages the computer's memory, and allocates the computer's resources to system users.

Figure 1-2 **Functional View of the Kernel**



The Shell

When you enter a request to the operating system by typing on the terminal keyboard, a program called the shell translates your request into language the computer understands. If your request is valid, the computer honors it and carries out an instruction or set of instructions. Because of its job as translator, the shell is called the command language interpreter.

The shell can also be used as a programming language. Section 4, "Using the Shell," describes the shell and its capabilities in detail.

Commands

A program is a set of instructions that the computer follows to do a specific job. In the CENTIX operating system, programs that can be executed by the computer are called executable programs or commands.

You enter commands from the shell. The standard shell command prompt is a dollar sign ($). See Section 4 for more information on shell prompts.

The Structure of a Command

When the CENTIX system displays the $ prompt on your screen, it is ready for you to enter commands. The line on which you enter commands is called the command line.

Command line syntax consists of one or more of the following elements, separated by a blank or blanks. Always follow a command by pressing the RETURN key:

```
command option(s) argument(s)
```

where

- *command* is the name of the program you wish to run,
- *option* modifies how the command runs, and
- *argument* specifies data on which the command is to focus or operate (usually a directory or file name).

A command line can contain a command name only, or it can list options and/or arguments in addition to the command. If you specify options and arguments on the command line, you must separate each with at least one blank. Blanks can be typed by pressing the space bar or the TAB key. If a blank is part of the argument name, enclose the argument in double quotation marks (for example, "sample 1").

Some commands allow you to specify multiple options and/or arguments on a command line. Both of the following examples are correct ways of entering the **wc** (**w**ord **c**ount) command with options:

```
$ wc -l -w file1 file2 file3
$ wc -lw file1 file2 file3
```

What Commands Do

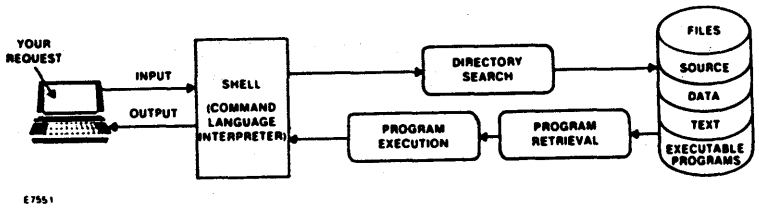
The outer circle of Figure 1-1 organizes operating system programs into general categories according to what they do. Among other things, the programs allow you to:

- Process text. This capability includes programs such as line and screen editors, which create and edit text.
- Manage information. The operating system provides many programs that allow you to create, organize, and remove files and directories.
- Communicate electronically. Several programs, such as **mail**, provide you with the capability to transmit information to other users and to other operating systems.

How Commands Execute

Figure 1-3 gives a general idea of what happens when the operating system executes a command.

Figure 1-3 **Flow of Control at Command Execution**



When the shell shows the $ prompt, signaling that it is ready to accept your request, you type in the command to be executed on the keyboard. The shell then searches one or more directories to locate the program you specified.

When the program is found, the shell brings your request to the attention of the kernel. The kernel then follows the program's instructions and executes your request.

After the program runs, the shell asks you for more information or tells you it is ready for your next command by showing the $ prompt.

How the Shell Finds Commands

When you invoke a command, the shell looks first in your current directory for a program with the given name. If the shell cannot find the program in the current directory, it then looks in system directories:

/bin	Most commands reside in the directory /bin (for binary programs). This is the first directory searched by the shell when a command is invoked.
/usr/bin	Some programs reside in /usr/bin to save space in the /bin directory.
/usr/local/bin	If the shell cannot find the program in the /bin or /usr/bin directories, it then searches /usr/local/bin. The /usr/local/bin directory is recommended for locally implemented programs.
/etc	This directory contains some administrative commands.

If the shell cannot find the command in any of these directories, it displays the message:

```
command: not found
```

and returns the $ prompt to the screen.

There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. If you want to execute a program that does not reside in one of these directories, specify the command using the full path name (path names are discussed in Section 3).

See the *CENTIX Operations Reference Manual* for full path names of commands.

CENTIX and BTOS

The CENTIX system is based on two operating systems: CENTIX and a Burroughs workstation-based operating system called BTOS.

All CENTIX-based system software and applications run on the Application Processor (AP) in the XE 500, and BTOS runs on all other processor types. The BTOS-based processors support most of the major internal services such as disk access, terminal input and output (I/O), printing operations, data communications (data comm) I/O, and so on.

Whenever CENTIX accesses devices, such as printers and terminals, it does so through BTOS. You specify to the CENTIX system that you want to access a device (for example, you want to print something on a printer). You do this by typing a command (see "Commands," later in this chapter). CENTIX then reads your command and requests BTOS to perform the printing function. BTOS then performs the function. Any information from the printer (such as errors in printing or specifics about the status of the printer) is sent back through BTOS to CENTIX. CENTIX then passes that information back to you.

Getting Started

This section describes how to begin using the CENTIX operating system. It covers the following topics:

- Booting up.
- Getting a login name.
- Logging in.
- Using and changing a password.
- CENTIX keyboard conventions.
- Logging off.

Booting Up

Before you can boot up—or start—your terminal, you must figure out what type of terminal you are using.

For the CENTIX operating system, you use either a PT 1500 connected directly to the computer or a remote PT 1500.

Your terminal is a remote PT 1500 if it is connected through an RS-232-C line, either directly to the computer or to a modem that communicates with the computer over data communication lines.

Check with your system administrator to see what type of terminal you are using.

Booting Up a PT 1500

In most cases, you boot up a PT 1500 by turning on the power switch on your terminal. The power switch is located at the base of your terminal on the left-hand side.

When you boot up your terminal, the operating system performs some initial configuration routines (which you do not see). After a moment, the Burroughs logo appears on your screen. Just beneath the Burroughs logo, on the bottom left of the screen appears the word:

```
login:
```

The operating system is ready for you to log on (see "Logging In," later in this section).

Booting Up a Remote PT 1500

There are two methods of booting up a remote PT 1500: the short method and the long method. You can use the short method only if your terminal is configured for one of the combinations of data communications characteristics shown in Table 2-1. Your system administrator can tell you which characteristics your terminal line uses.

Table 2-1 **Communications Characteristics Combinations for Booting Up a Remote PT 1500**

Bits	Parity	Baud Rate	Key
8	none	1200	1
8	none	2400	2
8	none	4800	3
8	none	9600	4
7	odd	1200	5
7	odd	2400	6
7	odd	4800	7
7	odd	9600	8
7	even	1200	9
7	even	2400	0
7	even	4800	-
7	even	9600	=

Using the Short Method

Table 2-1 shows 12 combinations of data communications characteristics. If your terminal line is configured for one of these combinations, follow these steps to boot up your remote PT 1500:

1 Check the **Key** column in Table 2-1 for the number or character that corresponds to the data communications characteristics of your terminal. Press and hold the appropriate key on the top row of your keyboard while you turn on your PT 1500. Release the key when the cursor appears on your screen.

   Your terminal automatically enters the emulate mode, in which it operates as a non-programmable terminal that uses ASCII code.

2 If your terminal is connected to a modem, establish your connection with the computer at the modem. This procedure differs depending on what type of modem you have. See your system administrator for instructions on how to use your modem.

3 At your terminal, if a login prompt does not automatically appear, press the RETURN key to get a prompt. Use a login name of "lpp":

```
login: lpp
```

   The screen displays:

```
-lpp: attempting to download /usr/lib/iv/ws101.232
```

   The screen clears, and a B appears at the bottom of the screen, followed by half a line of dashes. This indicates that the PT 1500 software is being loaded from the computer to the PT 1500.

   The screen clears again and another login prompt appears.

4 Your terminal is now ready for you to log in.

Using the Long Method

If the data communications characteristics for your terminal are not shown in Table 2-1, follow these steps to boot up your remote PT 1500:

1 While holding down the space bar, turn on your PT 1500. "V 2.0" should appear on the screen. This represents the version of the Read-Only Memory (ROM) that is in your terminal. If "V 1.0" appears instead, a new ROM must be installed in your terminal before you can boot up. See your system administrator.

2 The options prompt "B,C,E,F,M,R,S,T:" appears below the V 2.0. The F and R options are used to define the data communications characteristics for your terminal to the ROM. Ask your system administrator whether or not you can use the default characteristics. If you cannot use the default, go to the next step. If you can use the default, go to step 5.

   Note that the default characteristics are 8 data bits, a parity setting of zero, one stop bit, and a baud rate of 9600.

3 Press the F key. An equal sign (=) appears after the F that you entered. Enter three digits (check with your system administrator):

   - The first digit represents the number of data bits (7 or 8).
   - The second digit represents the parity setting (0 for none, 1 for odd, 2 for even).
   - The third digit represents the number of stop bits (1 or 2).

4 Press the R key. An equal sign (=) appears after the R that you entered. Enter a single digit that represents the baud rate, as follows (check with your system administrator):

- Enter a 1 for a baud rate of 1200.
- Enter a 2 for a baud rate of 2400.
- Enter a 3 for a baud rate of 4800.
- Enter a 4 for a baud rate of 9600.

The options prompt reappears.

5 Press the E key. Your terminal enters the emulate mode, in which it operates as a non-programmable terminal that uses ASCII code.

6 If your terminal is connected to a modem, establish your connection to the computer at the modem. This procedure differs depending on the type of modem you have. See your system administrator for instructions on how to use your modem.

7 At your terminal, if a login prompt does not automatically appear, press the RETURN key to get a prompt. Use a login name of "lpp":

```
login: lpp
```

The screen displays:

```
-lpp: attempting to download /usr/lib/iv/ws101.232
```

The screen clears, and a B appears at the bottom of the screen, followed by half a line of dashes. This indicates that the PT 1500 software is being loaded from the computer to the PT 1500.

The screen clears again and another login prompt appears.

8 The terminal is now ready for you to log in.

Getting a Login Name

To receive a login name, set up a system account through your system administrator.

Types of login names vary from system to system. Possible examples are your last name, your nickname, or a system account number. A login name should be three to eight characters in length. It can contain any combination of alphanumeric characters, as long as it starts with a letter. It cannot, however, contain any symbols.

Note: *Enter your login in lowercase letters only. CENTIX is a case-sensitive operating system, which means that lowercase letters are interpreted differently than uppercase letters. If you use uppercase letters, the system will also use uppercase letters until you log out and log in again.*

Logging In

To log in to the system, follow the procedure for booting up your terminal.

When the connection is made, the system prompts you for your login name:

```
login:
```

Enter your login name at the login prompt and press the RETURN key:

```
login: yourloginname
```

Remember to type in lowercase letters.

When you attempt to establish contact with the operating system, the system verifies that you are an authorized user. If you pass the system's security checks, the system allows you to log in.

If you make a mistake and do not correct it before pressing the RETURN key, the operating system displays the message "login incorrect" on your screen and asks you to try again by printing the login prompt.

If you do not complete the login successfully within a certain time, you will be disconnected.

Using a Password

After typing in your login name, the system may or may not prompt you for a password (depending on the securities established on your system). Your system administrator can provide you with a password.

If the system prompts you for a password, type in your password and press the RETURN key. For security reasons, the system does not print (or echo) your password on the terminal screen.

If you enter the correct login and password, the system prints messages (if there are any) for system users. These messages might include details about a new system tool or provide a schedule for system maintenance. These messages are followed by the operating system command prompt, which is the $ symbol:

```
login: yourloginname
password:
System messages
$
```

Changing Your Password

After you successfully log in for the first time, one of the first procedures you should perform is to change your password.

A password provides security for your files; only those who know the password and the system administrator can access your files.

Use the following procedure to change your password.

1 At the command prompt ($), type in the command **passwd** and press the RETURN key. The system displays a message indicating that it is changing your password, and it prompts you to enter your old password.

2 Type in your old password and press the RETURN key. The system prompts you to enter your new password.

3 Type in your new password and press the RETURN key. The password should be at least 6 characters long and should contain at least two alphabetic characters and one numeric or special character.

   When you press the RETURN key, the system prompts you to re-enter your new password.

4 Re-enter your new password. The $ command prompt appears again.

Problems When Logging In

Some problems may occur when you try to log in to the system.

For example, each character you type might appear twice on the terminal screen, the RETURN key might not work properly, or only uppercase letters appear on the screen.

Some problems can be corrected by logging off the system and logging on again. If logging on a second time does not remedy the problem, check with your system administrator.

CENTIX Keyboard Conventions

To interact effectively with the operating system, you should be familiar with certain CENTIX keyboard typing conventions.

Table 2-2 lists some of these conventions and their meanings.

Table 2-2 **Keyboard Conventions**

Key(s)	Meaning
BACKSPACE	Erase a character. This is a non-printing character.
@	Erase or kill an entire line.
DELETE	Delete or kill the current command line. Also stops the execution of a program or a command. This is a non-printing character.
GO	Use with another character to perform a specific function (called escape sequence). Also used to indicate the end of create mode when using the vi screen editor. This is a non-printing character.
RETURN	End a line of typing. Use to enter commands at the $ prompt. This is a non-printing character.
CODE-d	Stop input to system or log off; designated as ^d. This is a non-printing character.
CODE-h	Backspace for terminals without a backspace key; designated as ^h. This is a non-printing character.
CODE-i	Horizontal tab for terminals without a TAB key; designated as ^i. This is a non-printing character.
CODE-s	Temporarily stops output from printing on a screen; designated as ^s. This is a non-printing character.
CODE-q	Resumes printing after typing ^s; designated as ^q. This is a non-printing character.

Note that all code characters, also known as control characters, are sent to the computer by holding down the CODE key and pressing the appropriate letter.

Correcting Typing Errors

You can correct typing errors two ways (providing you have not yet pressed the RETURN key).

The BACKSPACE key allows you to erase previously typed characters on a line, and the @ sign allows you to delete the entire line on which you are working. The BACKSPACE and @ characters are default values for character and line deletion, respectively.

Pressing the BACKSPACE key erases the character previously typed, and repetitive use of BACKSPACE erases any number of characters back to the beginning of the line, but not beyond that.

To delete the entire line on which you are working, press the @ key.

When you press @, the system moves on to the next line, ignoring anything typed on the previous line.

If you want to use the @ character literally, that is, you want to use this character in a command sequence, you must precede @ with a backslash (\).

For example, to enter the sentence:

```
Only one @ appears on this page.
```

type \ before entering the @. Otherwise, the @ would move you to the next command line

Stopping a Command

To stop the execution of a command, press the DELETE key.

The system returns the $ prompt, indicating that the system has terminated the program and is ready to accept your next command.

Using Control Characters

Control characters are used in combination with other keyboard characters to initiate a specific action, such as backspacing or tabbing, across a line of typing. In addition, some control characters perform CENTIX system-specific commands, such as temporarily halting output from printing on a terminal screen.

You type control characters by holding down the CODE key and pressing the appropriate alphabetic key. Control characters do not print on the terminal screen when typed.

In many cases, control characters are designated with a preceding caret (^), such as ^s, to help identify them.

Temporarily Stopping Output

To temporarily stop the operating system from printing output on your terminal screen (for example, if a file is being displayed too quickly), enter CODE-s.

Printing of output ceases.

To resume printing of output, type CODE-q.

Terminating a Computing Session

When you have completed a session with the operating system, you can enter CODE-d. Logging off the system is described in more detail later in this section.

Additional Control Character Capabilities

The operating system provides other control character capabilities, such as:

- CODE-h, which is the same as BACKSPACE.
- CODE-i, which sets horizontal tabs. This is the same as the TAB key.

Typing Speed

After logging in to the system, you can type as fast as you want, even during periods when the operating system is responding to or executing a command.

The printout on your terminal screen may appear scrambled because your input is intermixed with the system's output. The operating system, however, has full read-ahead capability, which allows it to separate input from output and to respond to your command properly.

With full read-ahead capability, the system stores your next request while it is outputting information on your terminal screen in response to a previous request.

Logging Off

When you have completed a session with the operating system, you can log off in one of two ways: type in **exit** and press the RETURN key or use CODE-d.

In a few seconds, the system displays the login message, which indicates that you have successfully logged off the system.

The CENTIX File System

This section contains information on the file system and on how you fit into its organization. It also discusses how you can organize and manipulate files and directories in the file system.

File System Structure

The file system is made up of three major elements, organized in a hierarchy:

- The *file*. A file is a collection of information. A file must be contained within a directory (see the following definition).
- The *directory*. A directory is a grouping of files. A directory can have other directories (called subdirectories) assigned to it. A directory can also be considered a file (see the subsection **Files**).
- The *file system*. A file system is a collection of files that are stored on a defined physical memory device (such as a disk or a tape). A file system must be attached to, or is subordinate to, a directory. The file system physically contains the files that are logically assigned to that directory.

Note: *The term "file system" is used in two ways. It is used as defined above when discussing a specific file system, and it is used to describe the entire hierarchy of directories, specific file systems, and files in an operating system.*

Files

A file is a collection of information. A file can be as small as one byte, and as large as there is available space in the file system. Files are named and have defined access privileges (access privileges are described in "Protecting Your Files," later in this section).

There are three types of files in the operating system:

- *Ordinary files*, which contain the characters of a document or a program. Characters can be stored in binary or in ASCII code.
- *Directory files*, or directories, which contain the names of the files and subdirectories assigned to it. Directory files can be read, but not written to. Directory files will be referred to simply as directories in this document.
- *Device files*, or special files, which are associated with input and output devices. The information in the files is stored at the device itself, not in the file system. The same naming and access rules apply to device files as to the other files. The device files are described in your *System Software Operations Reference Manual.*

You create a file with one of the operating system editors (see Volume 2 of this guide). When you create a new file, it is automatically placed in the directory in which you are working.

Directories

Directories are the building blocks of the file system hierarchy. The base, or starting point, of the hierarchy is a directory called root (which is specified by a /). Every directory must be subordinate to root, and every subdirectory must be subordinate to a directory that is subordinate to root. There is no limit to the number of layers you can build (that is, the number of subdirectories you can specify) in this hierarchy.

Figure 3-1 **File System Hierarchy**

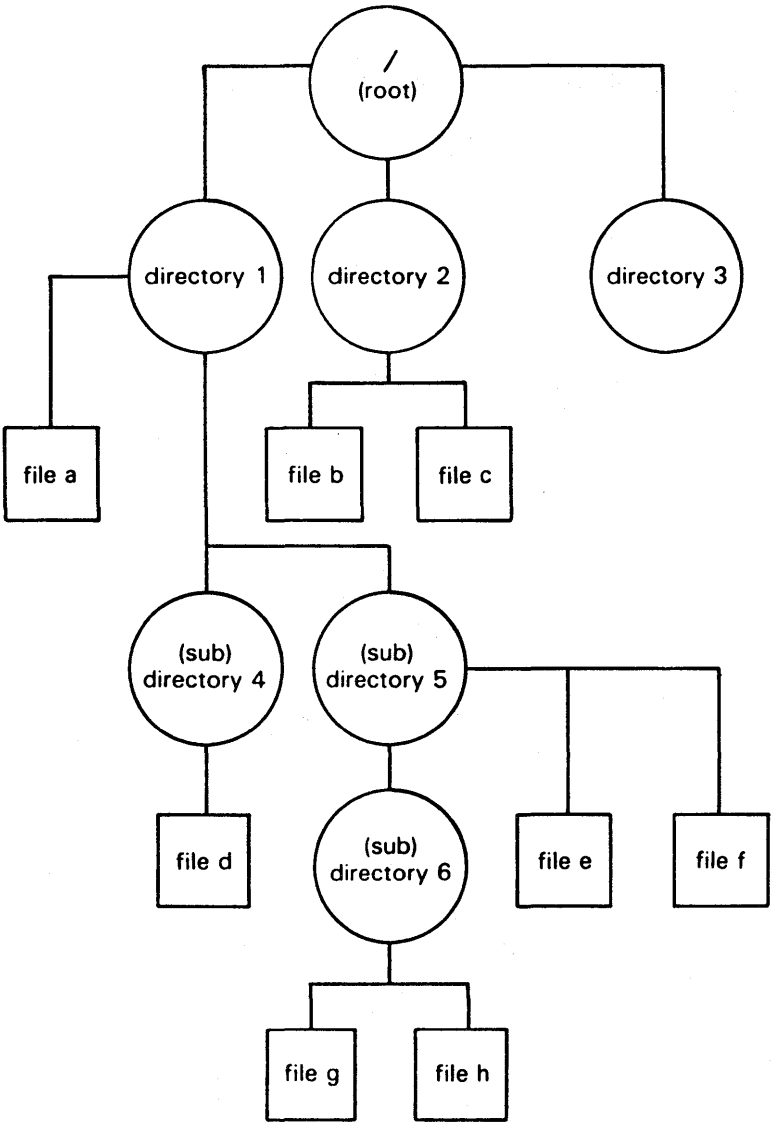


Figure 3-1 illustrates the file system hierarchy. Note the following points from the figure:

- A directory can contain files and other directories simultaneously.
- Files must be subordinate to directories; files cannot be subordinate to files.
- A directory can exist with no files or directories assigned to it. The directory is then "empty."
- The term subdirectory is generally used to describe a directory that is subordinate to any directory (including other subdirectories) other than root. A subdirectory is logically no different than a directory.

File Systems

A file system is a collection of files that are stored on a defined physical memory device. The files in a particular file system are those that are contained in the directory to which the file system is assigned, or "mounted."

When your operating system is first installed, one file system is automatically defined, mounted to the root directory. This file system is associated with a physical memory device, such as a disk. If no other file systems are ever defined, every directory and file that is created on your system will be stored on that device.

Your system administrator will probably determine, however, that more memory space is needed to store all of the files that will be created. He or she will then create other file systems, assign them to other memory devices, and mount them to subdirectories under root.

Your Place in the File System Structure

When you are interacting with the operating system, you do so from a location in its file system structure. The operating system automatically places you at a specific point in its file system every time you log in. From that point, you can move through the hierarchy to work in those files and directories that you have permission to use (which automatically include those files and directories that you own).

The following sections describe your place in relation to the file system structure and how this relationship changes as you move through the file system.

The Home Directory

When you log in, the operating system positions you at a specific point in its file system structure—your login or home directory. Your home directory is created when your system administrator opens your account.

The login name that was assigned to you when your account was set up is usually the name of this home directory. Every user with an authorized login name has a unique home directory in the file system.

Within your home directory, you can create files and additional directories (or subdirectories), you can move and delete files and directories, and you can control who can access your file and directories

The Working Directory

As long as you continue to work in your home directory, it is considered your current or working directory. If you move to another directory, that directory becomes your new working directory.

To find the name of your current working directory, use the **pwd** (print working directory) command:

```
$ pwd
```

For example, if your login name is danno and you issue the **pwd** command from your home directory, the operating system gives the following response:

```
$ pwd
/user1/danno
```

The system reply indicates that your working directory is /user1/danno. This the full or complete name of the working directory. The name of a directory (such as /user1/danno) or a file is also referred to as a path name (see "Path Names," below).

Path Names

Every file and directory in the operating system is identified by a unique path name. The path name tracks or indicates the location of the file or directory relative to the structure of the system.

In addition to identifying the location of a file or directory in the file system structure, a path name provides directions to that file or directory.

In the above example, /user1/danno tells you that the root directory / (indicated by the leading slash in the line) contains the directory user1, which in turn contains the current working directory, which is danno. All other slashes in the path name are used to separate names of directories and files.

In the file system, there are two types of path names—full and relative.

Full Path Names

A full path name (sometimes called an absolute path name) gives you directions that take you from the root directory down through a unique sequence of directories that leads to a particular directory or file.

You can use a full path name to reach any file or directory in the operating system.

A full path name always starts at the root of the file system. The first character in a full path name is always a / (slash). The final name in a full path name can be either a file name or a directory name. All other names in the path must be directories.

For example, if you have a directory named dog in the root directory, and a file named ozzie in the dog directory, ozzie's full path name is:

```
/dog/ozzie
```

This can be translated as: the ozzie file, which is assigned to the dog directory, which is assigned to the root directory (/).

Note that you cannot tell by ozzie's path name whether ozzie is a file or a subdirectory. It is obvious that dog is a directory because it has ozzie assigned to it. Files cannot be assigned to files. The **ls** command, which is discussed in "Listing the Contents of a Directory," later in this section, can be used to determine what ozzie is.

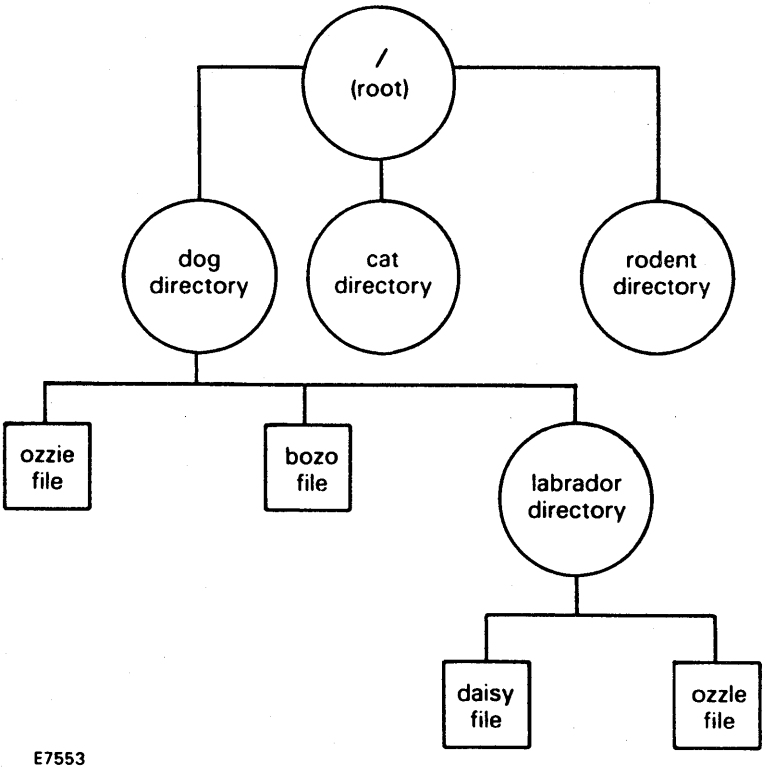
You can have more than one file or directory with the same name, but they must reside in different directories, and will therefore have different path names.

For example, you can have another file named ozzie in the directory labrador. In fact, labrador can even be a subdirectory in the directory dog. The path names for the two ozzie files are:

```
/dog/ozzie
/dog/labrador/ozzie
```

As long as you always identify the ozzie file that you want by the correct path name, there will be no confusion. Figure 3-2 illustrates the two ozzie files in the file system.

Figure 3-2 **Example of Files with the Same Names in Different Directories.**



Relative Path Names

A relative path name is the name of a file or directory that varies with relation to the directory in which you are currently working.

Relative path names allow you a shortcut when identifying files and directories. You do not have to give the complete path name when you are accessing a file or directory that resides in the directory in which you are currently working.

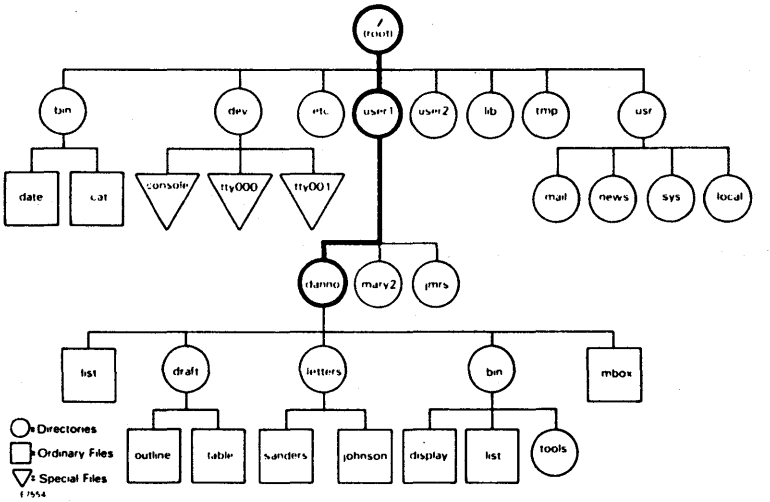
For example, if you are working in the laborador directory, and you want to work on the daisy file that resides in laborador, you can identify daisy as simply daisy. The system uses the relative path name to search for the daisy file in labrador, since that is where you are working.

A relative path name begins with a directory or file name. A relative path name can also begin with a . (dot), which is shorthand notation for the directory in which you are currently located, or a .. (dot dot), which is shorthand notation for the directory immediately above your current working directory in the file system hierarchy. The .. (dot dot) directory is called the parent directory of the one in which you are currently located, which is . (dot).

To ascend to the parent of your working directory, you can use the .. notation. This means that if you are in the directory named labrador, .. is the path name to dog, and ../.. is the path name to dog's parent directory, root. From labrador, you could also trace a path to the file ozzie in the dog directory by using the path name ../ozzie (.. brings you up to dog, and /ozzie brings you down to ozzie).

Figure 3-3 shows a sample file system. In this file system, danno is the home directory of the user named danno. danno contains directories named draft, letters, and bin, and two files named list and mbox. The relative path name to any of these is simply its name (draft, letters, bin, list, or mbox).

Figure 3-3 **Sample File System**



Some examples of full and relative path names are as follows:

Path Name	Meaning
/	Full path name of the root directory for the file system.
/bin	Full path name of the bin directory that contains most executable programs and utilities.
/user1/danno/bin/tools	Full path name of the directory called tools, which belongs to the directory bin, which belongs to the directory danno, which belongs to user1, which belongs to root.
bin/tools	Relative path name to the file or directory called tools in the directory bin. If the current directory is /, then the operating system searches for /bin/tools. But, if the current directory is danno, the system searches the full path /user1/danno/bin/tools.
tools	Relative path name of a file or directory called tools in the working directory.

Organizing a Directory Structure

You can use the following four commands to organize and use a directory structure:

mkdir	Create or make new directories and subdirectories within your current directory.
ls	List the names of all the subdirectories and files in a directory.
cd	Change your location from one directory to another in the file system.
rmdir	Remove a directory when you no longer have a need for it.

Creating Directories

To create a directory, use the **mkdir** (or **make dir**ectory) command:

```
$ mkdir directoryname
```

You should create subdirectories in your home directory according to some logical scheme to help you retrieve information you will keep in files. A convenient way to organize your files is to put all files pertaining to one particular subject together in one directory.

When you create directories, use the following guidelines:

- The name of a directory (or file) can be from one to fourteen characters in length.
- All characters other than / are legal. Some characters, however, are best avoided, such as blanks, spaces, tabs, or backspaces, and the following:

  @ # $ ^ & * ( ) ` [ ] \ | ; ' " < >

If you use a blank or tab in a directory or file name, you must enclose the name in quotation marks on the command line.

- Do not use +, -, or . as the first character in names.
- Upper case and lower case characters are distinct to the operating system. For example, the directory or file named draft is not the same as the directory or file named DRAFT.

Examples of legal directory or file names are:

memo	MEMO	section2	ref:list
file.c	chap3+4	item1-10	outline

Listing the Contents of a Directory

To obtain information about directories—such as the files and directories they contain, names, sizes, and the dates last modified—use the **ls** command.

The **ls (list)** command lists the names of the files and subdirectories of the directory you specify by path name. If you do not specify a path name, **ls** lists the names of files and directories in your working directory.

For example, to list the contents of the directory in which you are currently working , enter the following:

```
$ ls
```

To list the contents of a directory other than your current directory, enter the path name with the command:

```
$ /directory/subdirectory
```

The screen prints out the files and directories in *subdirectory*.

Note that the **ls** command will not print the contents of a file. If you wish to see what a file contains, you can use the **cat, more, pg,** or **pr** commands, which are described under "Manipulating Files and Directories" later in this section.

Frequently Used ls Options

The **ls** command also accepts options that cause specific attributes of a file or subdirectory to be listed.

Two of the most commanly used **ls** options are **-a**, which lists all names in a directory, and **-l**, which lists a directory's contents in long format.

Refer to your *System Software Operations Reference Manual* for information on the other **ls** options.

Listing All Names in a Directory. The **ls** command with the **-a** option lists all of the names in a specified directory.

Some important file names in your home directory begin with a . (dot), such as .profile, . (the current directory), and .. (the parent directory). The **ls** command will not print these names unless you use the **-a** option on the command line.

For example, to list all the files in your working directory danno, including those that start with a . (dot), type **ls -a** and press the RETURN key. The terminal prints the following:

```
$ ls -a
.
..
.profile
bin
draft
letters
list
mbox
```

Listing Contents in Long Format. The **ls** command with the **-l** option lists the contents of a directory in long format.

For example, if you type **ls -l** and press the RETURN key while in the danno directory, the following is printed:

```
$ ls -l
total 30
drwxr-xr-x 3 danno project      96 Oct 27 08:16 bin
drwxr-xr-x 2 danno project      64 Nov 01 14:19 draft
drwxr-xr-x 2 danno project      80 Nov 08 08:41 letters
-rwx------ 2 danno project 123 01 Nov 02 10:15 list
-rw------- 1 danno project      40 Oct 27 10:00 mbox
$
```

After the command line, the first line of output, total 30, shows the amount of memory used, which is measured 512-byte sections or blocks of memory. Next is one line for each directory and file. The first character in each of these lines tells you what kind of file is listed, where:

- d = Directory,
- - = Ordinary disk file,
- b = Block special file, and
- c = Character special file.

The next several characters, which are either letters or hyphens, describe who has permission to read and use the file or directory. (Access permissions are discussed with the **chmod** command under "Manipulating Files and Directories," later in this section.)

The following number is a link count, which, in the case of a file, equals the number of directories it is in. In the case of a directory, the link count shows the number of files linked to that directory, as well as the number of directories immediately under it in the file system structure.

Next is the login name of the owner of the file, which is danno, and then the group name of the file or directory, which is project.

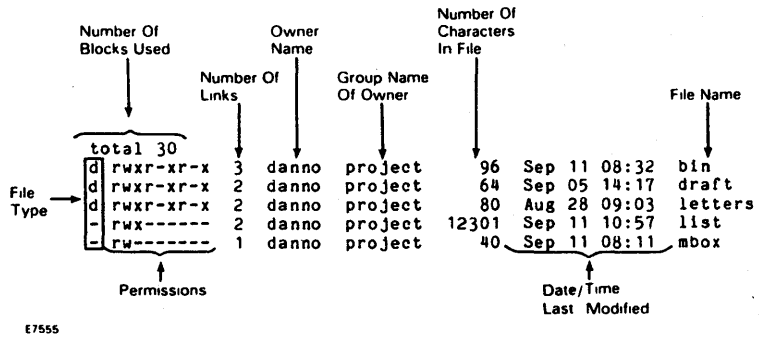
The following number indicates the length of the file or directory entry measured in units of information (or memory) called bytes.

Then there is the month, date, and time that the file was last modified.

Finally, the file or directory name is given.

Figure 3-4 shows what you get when you list the contents of a directory in long format.

Figure 3-4 **Output Produced by the ls -l Command**



Changing Your Working Directory

Use the **cd** (change directory) command to move around in the file system.

When you use the **cd** command to move to a new directory, that directory becomes your working directory.

To use the **cd** command, enter:

```
$ cd newdirectoryname
```

where the path name, whether full or relative, to the new directory is optional. Any valid path name of a directory can be used as an argument to the **cd** command.

If you use the **cd** command without specifying a path name, it moves you to your login directory regardless of where you are in the file system.

Removing Directories

To remove a directory, use the **rmdir** (remove directory) command.

The **rmdir** command removes a directory if that directory does not contain subdirectories and files (that is, if the directory is empty).

You cannot remove a directory that is not empty.

The standard format for the **rmdir** command is:

```
rmdir directoryname(s)
```

where one or more directory names can be specified.

The following example uses **rmdir** to try to remove the directory bin, which is not empty:

```
$ rmdir bin
rmdir: bin not empty
```

To remove the directory bin, you must first remove all subdirectories in bin with the **rmdir** command. You must also remove the files in bin. Removing files is discussed in "Manipulating Files and Directories" in this section.

Manipulating Files and Directories

This section introduces some basic commands that allow you to access and use the files in your directory structure. The following are some of these commands and their capabilities:

cat	Displays the contents of a file you name.
more	Prints on a terminal the contents of a file you name in lines or chunks.
pg	Prints on a video display terminal the contents of a file you name in chunks or pages.
pr	Prints on your terminal a partially formatted version of the file you name.
cp	Makes a duplicate copy of an existing file.
mv	Moves and renames a file.
ln	Creates links between files.
rm	Permanently removes a file when you no longer need it.
wc	Counts the lines, words, and characters in a file.
chmod	Changes permission modes for a file (and a directory).

Each of these commands is covered in one of the following sections. Refer to your *CENTIX Operations Reference Manual* for complete documentation on these commands.

Displaying the Contents of a File

There are four commands that allow you to display and print the contents of a file or files to your screen: **cat**, **more**, **pg**, and **pr**.

- The **cat** (concatenate) command outputs the contents of files you specify by name on the command line, and displays the result on your terminal (unless you tell **cat** to direct the output to another file or a new command).

- The **more** command is like **cat** except that it allows you to view the contents of files one line at a time, or one screenful at a time.
- The **pg (page)** command is used for reading the contents of lengthy files. The command displays the text of files in chunks or pages, a screenful at a time at your direction terminal.
- The **pr (print)** command partially formats and outputs specified files on your terminal.

Concatenate and Print the Contents of a File

The **cat** command displays the contents of a file or files.

The following example displays the contents of the file johnson:

```
$ cat johnson
This file contains a letter
to Mr. Johnson on the topic of
office automation.
```

To display the contents of two (or more) files, for example, johnson and sanders, type **cat johnson sanders** and press the RETURN key. The **cat** command reads johnson and sanders and displays their contents, in that order, on your terminal.

```
$ cat johnson sanders
This file contains a letter
to Mr. Johnson on the topic of
office automation.
This file contains a letter
to Mrs. Sanders inviting her to
speak at our departmental
meeting.
```

To direct the output of the **cat** command to another file or to a new command, see "Redirecting Input and Output" in Section 4.

Displaying the Contents of a File with the more Command

Use the **more** command to display the contents of a file one line at a time, or one screenful at a time.

On the command line, enter **more** and the name of the file or files you want to display:

```
$ more file(s)
```

The first screenful of text is displayed, followed by a prompt indicating the percentage of the file that has been displayed.

To advance the file one line, press the RETURN key.

To advance the file one screenful, press the space bar.

To cancel a **more** command before a display is finished, press the DELETE key.

The **more** program ends when 100% of all the files specified has been displayed.

Paging Through the Contents of a File with the pg Command

Use the **pg** (page) command to examine the contents of a file screenful by screenful on a terminal.

The **pg** command displays the text of a file in chunks or pages followed by a colon (:). After displaying the colon, the system pauses and waits for your instructions to proceed.

For example, your instructions can request **pg** to continue displaying the file's contents a page at a time or you can ask **pg** to search through the file to locate a particular character pattern. Table 3-1 summarizes some of the instructions you can give **pg** after the colon is displayed.

Table 3-1 **Summary of Selected pg Commands***

h	Help; display the list of available **pg** commands.
q or **Q**	Quit **pg** perusal mode.
RETURN	Display next page of text.

Table 3-1 **Summary of Selected pg Commands (Cont.)**

l	Display next line of text.
d or ^d	Display additional half page of text.
. or ^l	Redisplay current page of text.
f	Skip next page of text and display following one.
n	Begin displaying next file you specified on the command line.
p	Display previous file specified on the command line,
$	Display last page of text in file currently displayed.
/pattern/	Search forward in file for specified character pattern.
^*pattern*^	Search backward in file for specified character pattern.

* See your *CENTIX Operations Reference Manual* for a detailed explanation of all available **pg** commands.

To see the contents of a file with the **pg** command, use the following command line format:

```
$ pg filename(s)
```

For example, to display the contents of the file outline, type **pg outline** and press the RETURN key. The first page of the file appears on your screen. If the file has more lines in it than can be displayed in one page, the colon indicates there is more to be looked at when you are ready. Pressing the RETURN key prints the next page of the file.

When you reach the end of the file, (EOF): is displayed. The EOF designates that you have reached the end of the file and the colon is your cue for the next instruction.

When you have completed examining the file, you can type **q** or **Q** and press the RETURN key. The $ prompt appears on your screen.

Print Partially Formatted Contents of a File

The **pr** command is typically used to prepare files for printing.

Invoking **pr** without specifying any of the available options produces output in a single column with 66 lines per page, preceded by a short heading. The heading consists of five lines: two blank lines; a line containing the date, time, file name, and page number; and two more blank lines. The formatted file is followed by five blank lines.

Typically, the **pr** command is used along with the **lp** command to provide a paper copy of text as it was entered into a file (see Section 5, "Printing Files to a Line Printer"). However, you can also use the **pr** command to format partially and print the contents of a file on your terminal.

For example, to review the contents of the file johnson, type in the command **pr johnson** and press the RETURN key. The following example summarizes this activity.

```
$ pr johnson
Aug 25 11:29 1986  johnson Page1
This file contains a letter
to Mr. Johnson on the topic of
office automation.
.
.
.
$
```

Note that the ellipses after the last line in the file stand for the remaining 58 lines (all blanks in this case) that **pr** formatted into the output. On a PT 1500, which typically allows you to view about 24 lines at a time, the entire 66 lines of the formatted file will print continuously and rapidly to the end of the file. This means that the first 41 lines will "roll" off the top of your screen, making it impossible for you to read them.

In this case, use the CODE-s (^s) combination to stop printing on your terminal temporarily and CODE-q (^q) to resume the printing.

Copying Files

Use the **cp** (**copy**) command to copy the complete contents of one file into another:

```
$cp oldfile newfile
```

newfile is an exact copy of *oldfile*, with the same contents and access privileges. *oldfile* remains the same.

Note: *You cannot copy a file to another file with the same name in one directory. For example, if you try to copy the file outline to another file named outline in the same directory, the system tells you that the file names are identical and returns the $ prompt to you. Choose another name for the copy.*

The **cp** command also allows you to copy one or more files from one directory into a different directory while leaving the original file or files in place:

$cp *oldfile(s) /newdirectory/newfile(s)*

In this case, *newfile* can have the same name as the original *oldfile* because the two files are in different directories.

Caution: *If you copy to a file that already exists, the existing file is erased and its contents are replaced with the copy. The system gives no warning that a file is being erased. Use care when naming copies.*

Moving and Renaming Files

The **mv** (move) command allows you to rename a file in the same directory or to move a file from one directory to another.

As with the **cp** command, if you move a file to a different directory, you can rename the file or you can retain its original name.

To rename a file in a within directory, use the following command:

```
$ mv file1 file2
```

The **mv** command changes the file's name from *file1* to *file2*. The names *file1* and *file2* can be any valid names, including path names.

For example, to rename the file table as new.table in the same directory, use the following command:

```
$ mv table new.table
```

All of the following examples move the file table from the current directory named /user1/danno/draft to a file with the same name in the directory /user1/danno/letters:

```
$ mv table /user1/danno/letters
$ mv table /user1/danno/letters/table
$ mv table ../letters
$ mv table ../letters/table
$ mv /user1/danno/draft/table
  /user1/danno/letters/table
```

The file table can also be renamed table2 when moving it to the directory letters using any of the following:

```
$ mv table /user1/danno/letters/table2
$ mv table ../letters/table2
$ mv /user1/danno/draft/table
  /user1/danno/letters/table2
```

Linking Files

A link allows you to access one file by using one or more file names.

These names can be in the same directory, or they can be in different directories, but they all access the same actual file.

Each time you create a file, one link is automatically created (the link between the file name that you specify and the actual file itself).

Creating Additional Links

The **ln (link)** command allows you to create additional links to existing files.

To create an additional link to a file using the **ln** command, type:

```
$ ln file1 directory[/file2]
```

where *file1* is the name of the file that already exists, and *directory* and [*file2*] are the path and optional name of the file to which *file1* will be linked. Note the following points:

- A link is not a copy of a file; it is an additional pointer to an existing file. There is only one file.
- If *file2* already exists, it is overwritten and replaced with the contents of *file1*.
- If *file1* and *file2* are in the same directory, they must have different file names. This restriction does not apply to files in different directories.
- If *file2* is not specified, the new file is given the same file name as *file1*.
- The owner of the directory specified in *pathname* may have to change permissions so that the owner of *file1* has access to write to that directory. See "Protecting Your Files," later in this section.

The following example links a file called outline in the directory /user1/danno/draft to a file of the same name in the directory /user1/mary2. Assuming your working directory is danno/draft, type:

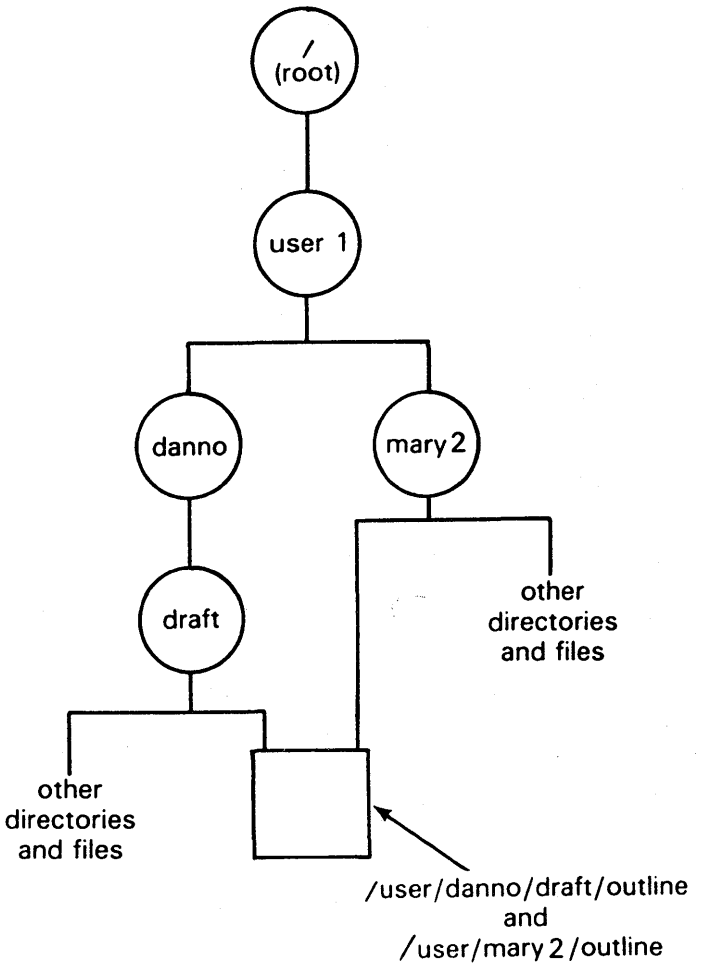
```
$ ln outline /user1/mary2/outline
```

Typing the following gives the same result:

```
$ ln outline /user1/mary2
```

since the linked file, outline, has the same name in both directories.

Figure 3-5 **Two Links to the Same File**



E7556

To give the linked file a different name in the /user1/mary2 directory, specify the new name when you use the **ln** command. For example:

```
$ ln outline /user1/mary2/plans
```

links the file outline in the /user1/danno/draft directory with the file plans in the /user1/mary2 directory. Although the names are different, the contents of the two files are exactly the same.

Changing Linked Files

Treat linked files the same as you treat any other files in the operating system. To make changes to links, use the one of the text editors described in Volume 2 of this guide.

Note that since links are pointers to particular files, any changes made to a file through one link will appear as changes through all other links. Note also that if two or more links are opened at the same time to change a file, only the changes made through the last link closed will show up (and all other changes will be ignored).

Linking Multiple Files

You can link multiple files in a directory with files in another directory by using the following command:

```
$ ln file1 ... filen directory
```

where *file1* ... *filen* are files 1 through *n* in the working directory, and *directory* is the path name of the directory to which the files are to be linked.

Note that you do not specify names for the linked files. When you link multiple files, the linked files have the same names as the original files.

Removing Links

To remove a link to a file, delete the linked file with the **rm** (remove files) command.

For example, if you want to remove the /user1/mary2/plans file, which is linked to outline in /user1/danno/draft, type:

```
$ rm /user1/mary2/plans
```

Because all links to a file are considered of equal value by the operating system, you can delete a file (that is,remove a link to a file) and still access the file through any remaining link.

For example, instead of removing the /user1/mary2/plans link as in the previous example, you can remove the outline link in the /user1/danno/draft directory. Even though outline was the file used to create the original link, you can still access the file through the remaining link, /user1/mary2/plans.

When you remove the last file in a link, you also remove the link.

Counting Lines, Words, and Characters in a File

The **wc** (**w**ord **c**ount) command reports the number of lines, words, and characters in a specified file.

If you name more than one file, the **wc** program counts the number of lines, words, and characters in each specified file and then totals the counts.

You can direct the **wc** program to give you only a line, a word, or a character count by using the **-l**, **-w**, or **-c** options, respectively.

To determine the number of lines, words, and characters in a file, use the following format on the command line:

```
$ wc file1
```

When you do this, the system responds with a line in the format:

```
l   w   c   file1
```

where

- *l* = Number of lines in file1.
- *w* = Number of words in file1.
- *c* = Number of characters in file1.

For example, to count the lines, words, and characters in the file johnson, type **wc johnson** and press the RETURN key. The terminal screen shows the following output:

```
$ wc johnson
      3    14    78    johnson
```

The system response displays the line count (3), the word count (14), and the characters count (78) for the file johnson.

To determine the number of lines, words, and characters in more than one file, use the following format:

```
$ wc file1 file2
```

In turn, the system responds with the following format:

```
l   w   c   file1
l   w   c   file2
l   w   c   total
```

where line, word, and character counts are displayed for *file1* and *file2* on separate lines and the combined counts appear on the last line called total.

As an example, if you request that the **wc** program count lines, words, and characters in the files johnson and sanders in the current directory, the system responds as follows:

```
$ wc johnson sanders
      3   14   78   johnson
      4   16   95   sanders
      7   30  173   total
```

In this case, the first line of the system response shows the line, word, and character counts for the file johnson. The second line of the output gives line, word, and character counts for sanders. The last line of output shows combined line, word, and character counts for both files in the line labeled total.

To get only a line, a word, or a character count, use the appropriate option of the **wc** command, that is, use **-l** if you want a line count, **-w** if you want a word count, or **-c** if you want a character count.

For example, by typing **wc -w sanders** on the command line and pressing the RETURN key, the following output is displayed:

```
$ wc -w sanders
    16   sanders
```

In response to the **-w** option of **wc**, the system tells you that the number of words in the file sanders is 16.

If the **-l** or **-c** option are specified for that file, the system responds with the number of lines or number of characters, respectively, in the file.

Protecting Your Files

The **chmod** (**ch**ange **mod**e) command allows you to decide who can read, alter, and use your files and who cannot.

If you own a file, then you are able to determine who has the right to read that file, to make changes to or write to the file, or to run or execute the file (if it is a program). These permissions are defined as:

r	Allows system users to read a file or to copy its contents.
w	Allows system users to write changes into a file or a copy of a file.
x	Permits system users to run an executable file.

You can determine who in the population of system users is entitled to these various permissions according to the following classifications:

u	You, the user and login owner of your files and directories.
g	Members of the group to which you belong (the group could consist of team members working on a project, members of a department, or a group arbitrarily designated by the person who set up your system account).
o	All other system users.

When you create a file or a directory, the system automatically grants or denies permission specifically to you, members of your group, and other system users.

Determining Existing Permissions

You can determine what permissions are currently in effect on a file or a directory by using the **ls** command with the **-l** option, which produces a long listing of a directory's content.

For example, typing **ls -l** and pressing the RETURN key while in the directory named danno/bin produces the following output:

```
$ ls -l
total 35
-rwxr-xr-x 1 danno project  9346 Oct 27 08:16 display
-rwx--x--x 1 danno project  6428 Nov 01 14:19 list
drwx--x--x 2 danno project    32 Nov 08 08:41 tools
```

Permissions for the files display and list and the directory tools are shown on the left of the terminal screen under the line total 35:

rwxr-xr-x **(display file)**

rwx—x—x **(list file)**

rwx—x—x **(tools directory)**

The nine characters in each of these sets represent three groups of three characters:

- The first set of three characters refers to your (or the user's/owner's) permissions.
- The second set refers to the members of the group.
- The last set refers to all other system users.

Within each set of characters, the r, w, and x indicate the permission currently enabled for the groups. If a dash appears instead of an r, w, or x, permission to read, write, or execute is denied.

As an example, look at the permissions for the file called display. The first three characters, which represent the owner's permissions, are r, w, and x. The owner of display, therefore, has read, write, and execute permissions for that file. The next three characters represent the group's permissions for the display file. The group can read (r) and execute (x) display, but cannot write to it or change it (since w is not specified). All other system users, represented by the last three characters, have the same permissions as the group.

Changing Existing Permissions: Symbolic Method

After you have determined what permissions are in effect, you can change them using either the octal method, which is described later, or the symbolic method.

The symbolic method uses the following symbols to specify instructions to **chmod**: r, w, x and u, g, o. You can change permissions using the following format:

```
$ chmod who + (or -) permission file(s)
```

where

- **chmod** is the name of the command.
- *who* is one of three user groups **u**, **g**, **o**, where **u** = user, **g** = group, and **o** = others.
- + or - is the instruction that grants (+) or denies (-) permission.
- *permission* is authorization to **r**, **w**, or **x**, where **r** = read, **w** = write, and **x** = execute.
- *files* are the files or directory names listed; assumed to be branches from your working directory, unless you use full path names.

As an example, refer to the display file in the previous subsection. The permissions show that, as the user and owner of display, you can read, write, and run this executable file.

To protect the file against accidentally changing it by denying yourself write (**w**) permission, type the following on the command line:

```
chmod u-w display
```

Use the **ls -l** command to verify the permission has changed.

```
$ ls -l
total 35
-rwxr-xr-x 1 danno project  9346 Oct 27 08:16 display
-rwx--x--x 1 danno project  6428 Nov 01 14:19 list
drwx--x--x 2 danno project    32 Nov 08 08:41 tools
$ chmod u-w display
$ ls -l
total 35
-r-xr-xr-x 1 danno project  9346 Oct 27 08:16 display
-rwx--x--x 1 danno project  6428 Nov 01 14:19 list
drwx--x--x 2 danno project    32 Nov 08 08:41 tools
```

The output of the first **ls -l** command shows that the user has permission to write to the display file, which is shown by a w in the first set of permission characters (remember that the first character position specifies what type of file is listed; see "Listing the Contents of a Directory," earlier in this section).

After using the **chmod** command and listing the contents of the directory with **ls -l**, the user no longer has permission to write to the file display.

As another example, notice that permission to write into the file display has been denied to members of the group and other users. These users, however, have read permission, which means that any of these users can copy the file into their own directories and then make changes to it.

To prevent all system users from copying the file display, enter the following:

```
chmod go-r display
```

The **g** and **o** stand for group members and all other users, respectively, and the **-r** denies them permission to read or copy the file display.

Note: *Although you can use the* ***chmod*** *command to grant or deny permissions for directories as well as files, granting or denying permissions for directories to various system users may cause problems. For example, if you grant read permission for a directory to yourself (**u**), members of your group (**g**), and other system users (**o**), every user who has access to the system can read the names of the files that directory contains by using the* ***ls -l*** *command. Similarly, granting write permission allows the designated users to create new files in the directory and change and remove existing ones. Granting permission to execute the directory allows the designated users the ability to move to that directory (and make it their working directory) by using the* ***cd*** *command.*

Changing Existing Permissions: Octal Method

Using the symbolic method to change permissions to read, write, and execute files and directories is one of two ways of using **chmod**. The other method uses a number system called octal, based on the numbers 0 through 7.

To use the octal method of **chmod** to change permissions for your files and directories, type the following on the command line:

```
$ chmod permission# file(s)
```

where *permission#* determines who can read, write to, and execute the named *file(s)*. Permission numbers are determined as follows:

add 400	if the owner is allowed to read *file(s)*.
add 200	if the owner is allowed to write to *file(s)*.
add 100	if the owner is allowed to execute *file(s)*.
add 40	if the group is allowed to read *file(s)*.
add 20	if the group is allowed to write to *file(s)*.
add 10	if the group is allowed to execute *file(s)*.
add 4	if all users are allowed to read *file(s)*.
add 2	if all users are allowed to write to *file(s)*.
add 1	if all users are allowed to execute *file(s)*.

For example, if the owner is allowed to read, write, and execute the file or directory, but no one else is allowed any access, you add 400 + 200 + 100 = 700. The command looks like this:

```
$ chmod 700 file
```

Or, if the owner, the group, and all users can read the file or directory, but only the owner can write to and execute it, you add 400 + 200 + 100 +40 +4 = 744. The command looks like this:

```
$ chmod 744 file
```

As an example, look at the list file in the following **ls -l** listing:

```
$ ls -l
total 35
-rwxr-xr-x 1 danno project  9346 Oct 27 08:16 display
-rwx--x--x 1 danno project  6428 Nov 01 14:19 list
drwx--x--x 2 danno project    32 Nov 08 08:41 tools
$
```

From the file listing, you can tell that list has execute (x) permissions for the owner of the file, the group, and all other system users; list has read (r) and write (w) permissions for its owner only.

To allow members of your group to read and change the contents of the file list (that is, you want to allow them to read and write to list). You also want other system users to be able to read the file, but you do not want to give them permission to change, or write to, list.

Because the owner of list is allowed to read, write to, and execute the file, add 400 + 200 + 100 = 700. To allow the group to read, write to, and execute the file, add 40 + 20 + 10 = 70. Since you do not want all other system users to be able to write to list, specify permissions for them to read and execute the file only: add 4 + 1 = 5. Add the sums:

700 + 70 + 5 = 775

Use this sum to specify the permissions for the file list:

```
$ chmod 775 list
```

An **ls -l** listing shows the result:

```
$ ls -l
total 35
-rwxr-xr-x 1 danno project  9346 Oct 27 08:16 display
-rwxrwxr-x 1 danno project  6428 Nov 01 14:19 list
drwx--x--x 2 danno project    32 Nov 08 08:41 tools
```

Note: *Permissions must be specified even if they already exist. For example, if you type* ***chmod 75 list*** *on the command line without specifying any permissions for the owner, the system assumes zero permissions for the owner (just as if you type* ***chmod 075 list****) and wipes out any existing permissions.*

Removing a File

To remove one or more files, use the **rm** (**rm**ove file) command:

```
$ rm file(s)
```

After the command executes, the file or files you specified are removed from the system permanently.

For example, to remove a file named new.outline in the current directory, type:

```
$ rm new.outline
```

To remove more than one file, such as the files outline and table, type:

```
$ rm outline table
```

Using the Shell

This section describes how to use the CENTIX shell. The first part of the section, "Shell Command Language," describes some of the methods and commands you can use to communicate within the shell. The second part of the section, "Shell Programming," shows how you can use the shell as a programming language. The following topics are covered:

- How to use special characters in the shell.
- How to redirect input and output.
- How to execute and terminate processes.
- How to create and execute a simple shell program.
- How to use variables in a shell program.
- How to use shell programming constructs for looping, conditional execution, and unconditional execution.
- How to locate problems and debug a shell program.
- How to modify your login environment.

Shell Command Language

Special Characters in the Shell

The shell language has special characters that perform various tasks in the shell. These special characters are listed below:

* ? [ ]	These are metacharacters. A metacharacter is a character that has a special meaning in shell command language. These metacharacters give you shortcuts for file names.
&	This character places commands in the background mode. Placing commands in the background mode allows the shell to execute programs, but with no need for communication to or from the terminal. While the shell performs the commands in the background, your terminal is free for you to work on other tasks.
;	This character allows you to type in several commands on one line. Each command must be followed by a ;. When you press the RETURN key, each command executes sequentially from the beginning of the line to the end of the line.
\	This character allows you to turn off the meaning of special characters such as *, ?, [ ], & and ;.
" ... "	
' ... '	Both double and single quotation marks turn off the meaning of certain special characters.

Blank spaces between arguments, called delimiters, are also special characters in the shell. They indicate when one argument ends and another begins.

Metacharacters

Metacharacters are characters that specify or represent other characters or a range of characters. They are used to help simplify file name generation. When input to the shell is broken down into words, the shell searches each word for any metacharacters that might exist. If a word contains metacharacters, the shell replaces the word with an alphabetically and numerically sorted list of file names that match the word. If no matching file names are found, the shell assumes the metacharacters to be a part of the word.

Metacharacter that Matches All Characters. The asterisk (*) matches any string of characters, including no characters at all (the null string). This metacharacter is also known as a wildcard character.

The * metacharacter alone refers to all the file names in the current directory. For example, to list all the files in your current directory, type **ls ***.

The * metacharacter is also used to expand file names in the current directory. It can be placed anywhere in a file name specification. For example, if you have several reports in a directory, all with different file names but all beginning with the string rep (for example, report1, repo, rep23, reports, and so on), you can list the names of the reports by typing:

```
$ ls rep*
```

All files beginning with rep are listed.

You can list all the files in your current directory that contain the letter s by entering the following:

```
$ ls *s*
```

To remove all the files containing the word letter, enter the following:

```
$ rm *letter*
```

This command removes all files beginning with the word letter (for example, letter1) and ending with the word letter (for example, bobletter), as well as all files containing the word letter somewhere in the middle (for example, bobletter1).

Caution: *Be careful when using the * metacharacter. Beware that typing* ***rm*** *** removes all files in your current directory.*

To print to your screen all files beginning with c and ending with s (for example, cs, c123s, chores, chapters), enter the following:

```
$ cat c*s
```

Metacharacter that Matches One Character. The question mark (?) metacharacter works the same way as *, except it matches only strings that are one character long.

For example, the following command lists chapters 1 through 9:

```
$ ls chapter?
```

Although ? matches any one character, you can use it more than once in a file name. For example, to list the rest of the chapters up through chapter99, enter the following:

```
$ ls chapter??
```

Of course, if you want to list all the chapters in the current directory, enter **ls chapter***.

Metacharacters that Match Specific Characters. The shell matches one or all of the specified characters or range of characters within brackets ([]).

Characters enclosed in [ ] act as a specialized form of the ? metacharacter. The shell matches only one of the characters enclosed in the brackets in the position specified in the file name. For example, if you use [crm] as part of a file name, the shell looks for c, r, or m:

```
$ ls [crm]ash
```

This command lists any or all of the files cash, rash, and mash, if they exist in the current directory.

Note: *Typing* ***ls [crm]ash*** *does not list the file crash, even if it exists in the current directory. The shell matches only one of the characters enclosed in brackets; crash has two (c and r).*

The shell also looks for one of a range of characters within the brackets. Two characters separated by a dash (-) indicate that all characters between the two, inclusive, are part of the character set. For example, chapter[1-9] is the same as chapter[123456789].

The shell also recognizes a range of letters. For [A-Z], the shell looks for uppercase letters. For [a-z], the shell looks for lowercase letters.

Running Commands in the Background Mode

Some shell commands take considerable time to execute. It is convenient to let these commands run in background mode to free your terminal so that you can continue to type in other shell tasks. To place a command in background mode, enter the command followed by an ampersand (&):

```
$ command&
```

For example, the **grep** command searches for specified patterns in files. If **grep** is searching long files, the program may take a considerable amount of time to execute. To place **grep** in the background mode, enter the following:

```
$ grep pattern files&
```

where *pattern* is the pattern that you want to look for in the specified *files*. **grep** executes while you perform other tasks in the shell.

Note: *When the **grep** command is finished executing, the output from the command will be displayed on your terminal. Although this will not cause a problem with whatever you are currently working on, it may be inconvenient. "Redirecting Input and Output," later in this section, discusses how to redirect the system responses of commands into files so that they do not display on your terminal and interrupt your current work.*

When you place a command in the background, a number prints on your screen just before the command prompt is displayed. This number is the process number. The process is essential if you want to stop the execution of a background command. See "Executing and Terminating Processes," later in this section.

Sequential Execution

You can enter several commands on one line by separating each command with a semicolon (;):

```
$ command1; command2; command3
```

For example, to change to your home directory, print the path of your current directory, and list the contents of your current directory, enter:

```
$ cd; pwd; ls
```

Notice that, after you press the RETURN key, the system responds to each command in the order that they appear on the command line.

Turning Off Special Character Meaning

The backslash (\) turns off the special meaning of a metacharacter.

For example, to search for the character * in a specified *file*, use the **grep** command:

```
$ grep \* file
```

Turning Off Special Characters by Quoting

The special characters in the shell lose their special meaning when they are enclosed by quotation marks:

- Single quotation marks (' ... ') turn off the special meaning of any character.
- Double quotation marks (" ... ") turn off the special meaning of any character except $, \, and `.

The following example of the **grep** command uses single quotation marks to search for all lines containing the words "This is" in the specified *file*:

```
$ grep 'This is' file
```

Without the quotation marks, **grep** assumes the words "This" and "is" to be separate arguments (in fact, **grep** would assume "is" to be the name of a file). The quotation marks turn off the special meanining of the delimiter between the two words, making "This is" one string.

Redirecting Input and Output

Standard input is a file from which the computer can read information. By default, this file is your terminal, and information that you enter at the terminal keyboard is read and processed as standard input by the computer.

Standard output is a file to which the computer sends information after processing input. By default, this file is your terminal, and standard output is printed on your terminal screen.

The redirection of standard input and output are important tools for performing many shell tasks and programs.

Redirecting Input. You can redirect the text of a file to be the input for a command by using the < symbol:

```
$ command < file
```

The following example redirects the contents of the text file called report to the **mail** command:

```
$ mail boss < report
```

In the above example, **mail** is the command, boss is the login name of the user to which the mail is being sent, and report is the file that is being sent instead of standard input from the terminal.

Redirecting Output. You can redirect the output of a command to be the contents of a file by using the > symbol.

When you redirect output into a file, you can either create a new file, append the output to the bottom of a file, or erase an old file and replace it with the redirected output.

Caution: *The format for creating a new file with redirected output is the same as erasing an old file and replacing it with redirected output. If you redirect a command into a file that exists, the shell erases the existing file and places the output of the command into that file. No warning is given that you are erasing a file. Use caution when redirecting output.*

To create a new file with redirected output, use a single redirection symbol (>) in the following command format:

```
$ command > file
```

The following example places a file listing of the current directory into a file called listing:

```
$ ls > listing
```

To append the output of a command to the end of an existing file, use a double redirection symbol (>>):

```
$ command >> file
```

The following example appends the output of a **cat** command to the end of the file called listing:

```
$ cat list.add >> listing
```

In the above example, list.add is the name of the file printed by the **cat** command to the file called listing.

The section on "Running Commands in the Background Mode" showed how to execute the **grep** command in background mode by using the ampersand (&).

To redirect the output of the **grep** command into a file called wordfile, and then look at the file when you have finished your current task, do the following:

```
$ grep pattern files > wordfile&
```

where *pattern* is the string that you are searching for in the specified *files*.

Using Pipes. The | character is called a pipe. A pipe redirects the output of one command to be the input of another command.

If two or more commands are connected by a pipe, the output of the first command is "piped" into the next command as input, and the output from that command is "piped" into the next command as input, and so on.

The general format for the pipe line is:

```
$ command1   command2   command3 ...
```

The output of command1 is used as the input of command2. The output of command2 is then used as the input for command3.

The following example takes the output of the **who** command as input for the **sort** command, producing an alphabetized listing of the users on the system:

```
$ who   sort
```

Command Output Substitution

The output of any command line or shell program that is enclosed in grave marks (`) *can be substituted anywhere on a shell command line.*

The following example substitutes the output of the **date** command for the argument of an **echo** command:

```
$ echo 'date'
```

Instead of returning the word "date" to your screen, the command echoes the output of the date command:

```
Tue Oct  7 10:56:21 GMT 1986
```

(provided it is 10:56:21 on Tuesday, October 7, 1986).

Executing and Terminating Processes

Running Commands at a Later Time

Note: *To run the following two commands,* ***at*** *and* ***batch****, you must have permission from your system administrator.*

When you type in a command line at your terminal, the CENTIX system tries to execute that command immediately. It is possible to tell the system to execute those commands at another time with the **batch** and **at** commands. End the commands with <^d> to let the shell know you have finished listing the commands to be executed.

The **batch** command is useful if you are running a process or shell program that uses a long amount of system time. The **batch** command submits a "batch" job, which consists of the commands to be executed, to the system. The job is put in queue, and then run when the load on the system falls to an acceptable level. This frees the system to respond to other input by you and other users on the system.

The general format for batch is:

```
$ batch
first command
.
.
.
last command
CODE-d
```

When you enter **batch** and press the return key, **batch** waits for you to enter the command or commands that you want to execute at a later time. When you finish entering the commands, press the CODE and d keys simultaneously.

If there is only one command to be executed later, it can be specified on the command line with **batch**:

```
$ batch command
CODE-d
```

The following example uses **batch** to execute the **grep** command at time when the system load is not heavy:

```
$ batch grep CENTIX * > cenfile
CODE-d
```

The system returns a job number, followed by the date and time you invoked the **batch** command.

The **at** command is like the **batch** command, except **at** gives the system a specific time that commands are to be executed. The general format of the **at** command is:

```
at time
first command
.
.
.
last command
CODE-d
```

Time must first give the time of day (in hours and minutes, am or pm), followed by the date (month and day, with the month abbreviated), if the date is not today.

The following example mails the file called memo to the user named mary at 8:15 am, October 8.

```
$ at 8:15 am Oct 8
mail mary < memo
CODE-d
```

The system returns a job number, followed by the date and time you invoked the **at** command.

To erase **batch** or **at** jobs that are waiting in the job queue, use the **-r** option of the **at** command:

```
$ at -r jobnumber
```

where *jobnumber* is the job number that is returned by the system when you invoke **at** or **batch**.

If you have forgotten the job number, the **at -l** command gives you the current jobs in the **batch** or **at** queue.

Obtaining the Status of Running Processes

The **ps** command gives you the status of processes you are running.

To execute the **ps** command, enter **ps** on the command line.

If you have commands that are running in the background mode (see "Running Commands in the Background Mode," earlier in this section), **ps** tells you the following information about those background processes:

PID	The *PID* is the process identification number of the command that is running in background mode. The *PID* is important if you decide to stop the execution of a background command.
TTY	*TTY* is the current number identification assigned to the terminal on which you are logged in.
TIME	The *TIME* is the cumulative execution time of each background process.
COMMAND	*COMMAND* tells the command that is being executed in background mode.

Terminating Active Processes

The **kill** command is used to stop active shell processes. The general format for the **kill** command is:

```
kill PID
```

To terminate a command that is running in the background mode, first find the *PID* of the process by using the **ps** command. Then terminate the process with **kill**.

Using the No Hangup Command

Another way to kill all processes is to hang up on the system, to log off.

If, however, if you want background processes to continue to run after you log off, use the **nohup (no hangup)** command. The **nohup** command allows background commands to continue to run even if you log off.

To allow a background process to continue to run after you log off, use the following format:

```
nohup command&
```

The **nohup** command can be terminated by the **kill** command.

Shell Programming

Getting Started

A shell program is a CENTIX system file that contains commands that you can use to perform specific tasks.

This section introduces some of the basics of programming within the shell.

Note: *Before you begin reading this section, make sure that you have a good understanding of how to use CENTIX text editors. See Volume 2 of this guide.*

Creating a Simple Shell Program

To create a shell program, use one of the editors discussed in the *CENTIX Operations Guide, Volume 2: Editing Operations* to create a file containing the program.

Once you enter one of the editors, type in the commands that you want to execute in the program.

The following simple shell program tells you the directory you are in with the **pwd** command, lists the contents of that directory with the **ls** command, then echoes on your screen "This is the end of the shell program" with the **echo** command:

```
pwd
ls
echo This is the end of the shell program
```

When you create a file containing a shell program, you can execute it in two ways: by using the **sh** command or by making the program itself executable.

Executing a Shell Program

To execute a shell program, such as the one created in the previous example, use the **sh** command:

```
$ sh file
```

For example, if the program in the above example is called prog1, enter the following:

```
$ sh prog1
```

The current working directory, its contents, and the line "This is the end of the shell program" are printed to your screen.

You can also create a shell program that can be invoked as a command (that is, the name of the file is also the command name that causes your program to run). Use the **chmod** command to make files executable.

The following example makes prog1 an executable program:

```
$ chmod u+x prog1
```

Now, prog1 can be invoked as a command in the directory in which it was created.

Creating a bin Directory for Executable Files

A bin directory is a directory that contains files that are executable from all of your directories.

To create an executable bin directory, use the following procedure:

1 Use the **cd** command to move to your login directory.

2 Use the **mkdir** command to make a directory called bin.

3 Use the **chmod** command to make the bin directory executable.

4 Add the bin directory to your PATH variable (see "Modifying Your Login Environment" at the end of this section).

To place executable files into your bin directory, use the **mv** command.

Caution: *You can give your shell program file any appropriate file name. However, do not name your programs with the same names as a system commands. Executable programs in your own directory override system commands of the same name, and the system will execute your command and not the command of the system.*

Variables

Variables can be placed in your shell programs to perform specific functions. For example, if you create a program that accepts any of a set of items, you can use a variable to represent each item of that set.

Variables are specified in a shell program by a dollar sign ($) and can be in one of two forms:

- Positional parameters.
- Variables that you define.

Positional Parameters

A positional parameter is a variable that is found in a specified position in the command line of your shell program. Your program then reads the value of a parameter according to its position on the command line.

To call on a positional parameter within a shell program, use $*n*, where *n* is the position of the parameter. For example, to call on the parameter in the first position, enter $1 in your program. For the fourth position, enter $4.

Positional parameters are typed in directly after the command. They are strings of characters delimited by spaces, except for the last parameter, which is ended by pressing the RETURN key:

```
$ program pp1 pp2 pp3 ... pp9
```

In the above shell program command line, *program* is the name of your shell program, *pp1* is the first positional parameter, *pp2* is the second positional parameter, and so on through *pp*9.

The following sample program uses the **echo** command to call on positional parameters.

```
echo The first positional parameter is: $1
echo The second positional parameter is: $2
echo The fourth positional parameter is: $4
echo The sixth positional parameter is: $6
```

The above example reads the first, second, fourth, and sixth parameters entered on the command line. For example, if the above program were named pp (and were made executable using the **chmod** command), entering:

```
$ pp 1 2 3 alpha beta gamma
```

would produce the following output:

```
The first positional parameter is: 1
The second positional parameter is: 2
The fourth positional parameter is: alpha
The sixth positional parameter is: beta
```

The following sample program, called **mm**, uses a positional parameter to mail a file called memo to a variable user:

```
mail $1 < memo
```

To execute **mm**, enter **mm** on the command line followed by the name of the user to whom the memo is to be sent:

```
$ mm debby
```

Parameters with Special Meaning

The following two parameters have special meanings in your shell programs:

$#	Records and displays the number of positional parameters typed in for the shell program.
$*	Substitutes sequentially all positional parameters, starting with the first positional parameter. **$*** does not restrict you to nine parameters.

The following sample program, called **num**, counts all the positional parameters and displays that number:

```
echo The number of parameters is: $#
```

To execute the program, enter **num** and a series of parameters on the command line. For example:

```
$ num This is a test for the program
```

returns the following:

```
The number of parameters is: 7
```

The next sample program, called **show.param**, echoes all of the parameters entered on the command line:

```
echo The parameters for this command are : $*
```

To execute the program, enter **show.param** and a series of parameters on the command line. For example:

```
$ show.param one two three
```

returns the following:

```
The parameters for this command are: one two three
```

Naming Variables

The shell allows you to name the variables within a shell program.

Named variable must first be assigned, then declared in a shell program.

To assign a value to a variable, use the following format:

```
var=value
```

where the variable *var* is separated from *value* by an equal sign (=). Note that there are no spaces on either side of the equal sign.

To declare a variable within a shell program, use the same format as for declaring positional paramater variables within a program:

```
$var
```

where the dollar sign ($) specifies that what follows is a variable, and *var* represents that variable.

The first character of a variable name must be a letter or an underscore. The rest of the name can be composed of letters, underscores, and digits. As in the case of shell program file names, it is a not recommended that you use a shell command as a variable name. Also, the shell has reserved some variable names to be used by the shell. The following names are used by the shell and should not be used as the name of one of your variables. A brief explanation of each variable is given.

CDPATH	This variable defines the search path for the **cd** command.
HOME	This is the default variable for the **cd** command (Home Directory).
IFS	This variable defines the internal field separators, normally the space, the tab, and the carriage return.
MAIL	This variable is set to the name of the file that contains your electronic mail.
PATH	This variable determines the path that is followed to find commands.

PS1	
PS2	These variables define the primary and secondary prompt strings. The defaults are $ and >.
TERM	This variable tells the shell what kind of terminal you are working on. It is important to set this variable if you are editing with vi.

Many of these named variables are explained in the "Using Shell Variables," later in this section.

Assigning Values to Variables

There are several ways to assign values to variables, besides using the equal sign (=).

- You can use the **read** command to assign input to the variable.
- You can use the output of a command, using grave accents ('...'), to assign the value.
- You can assign a positional parameter to the variable.

Assigning Values by the read Command

Use the **read** command to assign input to the specified variable:

```
read var
```

The values assigned by **read** to *var* will be substituted for $*var* in the program.

The following example uses the **echo** command to echo instructions, the **read** command to assign the input value to the variable *name*, and the **grep** command to search a file called list for *name*.

```
echo Type in the last name:
read name
grep $name list
```

The following program creates a file called list, which contains names and phone numbers. When list is created, the above program can be used to retrieve those phone numbers.

```
echo Enter name:
read name
echo Enter phone number
read number
echo $name $number >> list
```

Note the >>, which adds names and phone numbers to the end of the list file instead of replacing the file each time the program is called up.

Substituting Command Output for the Value of a Variable

Another way to assign a value to a variable is to substitute the output of a command for the value.

The following is the general format to assign output as the value for a variable:

```
var='command'
```

The variable *var* has the value of the output from *command*.

The following sample program takes the output from the **date** command, which prints the date and time, and pipes the output to the **cut** command, which extracts the specified characters. The resulting final output is then used as a variable, *time*. The program, when executed, returns just the current time.

```
time='date   cut -c12-19'
echo The current time is: $time
```

Assigning Values with Positional Parameters

A positional parameter can be assigned to a named parameter. For example:

```
var1=$1
echo $var1
```

takes the first positional parameter, $1, and assigns it to thje variable *var1*. The **echo** command then echoes the value of $var1, which is the first positional parameter.

Shell Programming Control Structures

The shell programming language has several control structures that give you more flexibility in your programs.

- The "here document" allows you to redirect lines of input into a command.
- The looping constructs for or while cause a program to reiterate commands in a loop.
- The conditional control commands, if or case, execute a group of commands only if a particular set of conditions is met.
- The break command gives the unconditional end of a loop.

Inserting Comments

You can insert comments in your executable files that do not affect their execution.

To place comments in a program, begin the comments with pound sign (#):

```
#comment
```

The shell ignores all characters between the # and the end of the line. For example, the following lines, if placed in a file, will be ignored when the file is executed:

```
# This program sends a generic birthday greeting
# This program needs a login as
  the positional parameter
```

The Here Document

The here document allows you to redirect lines of input of a shell program into a command.

The here document consists of the redirection symbol << and the delimiter that specifies the beginning and end of the lines of input. The delimiter can be one character or a string of characters. An exclamation point (!) is often used as a delimiter. The general format for the here document is:

```
command <<!
...input lines...
!
```

The following shell program redirects lines of input into the **mail** command. The program sends a generic birthday greeting with the **mail** command. The program is called **gbday**:

```
mail $1 <<!
Best wishes to you on your birthday.
!
```

Note that the login of the person to whom you send this message is the first positional parameter $1.

To send the greeting to a user named mary, enter the following:

```
$ gbday mary
```

Looping

Looping control structures give you repetitive execution of a command or group of commands.

The for or while commands cause a program to loop and execute a sequence of commands several times.

The for Loop. The for loop executes a sequence of commands for each member of a list.

The for command loop also requires the keywords in, do, and done. The for, do, and done keywords must be the first word on a line. The following is the general format of the for loop:

```
for variable
      in list of values
do
      command1
      command2
      .
      .
      .
      last command
done
```

The *variable* can be any name you choose. If it is var, then the values given after the keyword in will be sequentially substituted for $var in the command list. If the in keyword is omitted, the values for var will be the positional parameters. The command list between the keywords do and done will be executed for each value.

When the commands have been executed for the last value, the program will execute the next line below the keyword done. If there is no line, the program ends.

Indentations within loops are to make the shell programs easier to read. The lines need not be indented.

The following sample shell program, called mv.file, moves files to another directory.

```
echo Please type in the directory path
read path
for file
      in memo1 memo2 memo3
do
      mv $file $path/$file
done
```

In the above program:

- **echo** indicates directions to type the path name to the new directory.
- **read** reads the path name and assigns it to the variable *path*.
- **for** *variable* contains the variable that you name. In this example, *variable* is called *file*. It will appear as $file in the command sequence.
- **in** *list of values* is the list of file names you want to move. If the **in** clause is omitted, the list of values is taken to be $*, that is, the parameters entered on the command line.
- **do** *command sequence* is the command sequence that you want carried out. In this example, the command sequence is **mv $file $path/$file**.
- **done** indicates that the for loop is finished.

In the above example, the values for the variable *file* are already in the program.

To allow the files to be changed each time the program is invoked, use positional parameters or variables that you name. To do this, do not include the **in** keyword in the program. The program accepts your input from the keyboard as its values.

The while Loop. The while loop is like the for loop. However, it continues to execute the sequence of commands in the do...done list as long as the final command in the while command list returns a status of true (that is, can be executed).

The while, do, and done keywords must be the first characters on the line. The following is the general format of the while loop:

```
while
      command1
      .
      .
      .
      last command
do
      command1
      .
      .
      .
      last command
done
```

The following sample program uses the while loop to enter a list of names into a file.

```
echo "Please enter each person's name and press RETURN"
echo Please end the list with a CODE-d
while read x
do
     echo $x >> xfile
done
echo xfile contains the following names:
cat xfile
```

Note that after the loop is completed, the program executes the commands below the done keyword.

Note also that in the first **echo** command line, the apostrophe (') character, which is special to the shell, was used. To turn off the special meaning of the ' character, enclose the argument in double quotation marks.

Conditional Constructs if...then

The **if** command tells the shell program to execute the **then** sequence of commands only if the final command in the **if** command list is successful. The **if** construct ends with the keyword **fi**. The general format for the if construct is as follows:

```
if
        command1
        .
        .
        .
        last command
then
        command1
        .
        .
        .
        last command
fi
```

The following shell program demonstrates the if...then construct.

The program searches for a word in a file. If the **grep** command is successful, then the program will echo that the word is found in the file. In this example, the variables are read into the shell program.

```
echo Type in the word and the file name
read word file
if grep $word $file
        then echo $word is in $file
fi
```

The **read** command assigns values to two variables. The first characters typed in, up to a space, are assigned to *word*. All the rest of the characters, including spaces, are assigned to *file*.

Note: *Although the above program works, it returns more to your terminal than "*word *is in* file*": it also returns the output from the* ***grep*** *command. The subsection, "Shell Garbage Can: /dev/null," later in this section, describes how to redirect the output of such commands within your shell programs.*

Conditional Constructs if...then...else

The **if...then** construction can also issue an alternate set of commands with **else** when the **if** command sequence is false.

The following is the general format of the **if...then...else** construct:

```
if
      command1
      .
      .
      .
      last command
then
      command1
      .
      .
      last command
else
      command1
      .
      .
      .
      last command
fi
```

The **if...then...else** construct can be used to modify the shell program created in the previous example:

```
echo Type in the word and the file name
read word file
if
      grep $word $file
then
      echo $word is in $file
else
      echo $word is NOT in $file
fi
```

This program tells you whether or not the specified *word* is in *file*.

Shell Garbage Can: /dev/null

You can deposit any unwanted output from a command in a shell program by redirecting the output to a file called /dev/null.

The following example deposits the results of the **who** command in the /dev/null file:

```
$ who > /dev/null
```

The response to the **who** command is placed in /dev/null and becomes null, or nothing.

To redirect the output from the **grep** command in the example in the previous subsection, change the **if** command line:

```
if grep $word $file > /dev/null
```

The program now echoes only the text of the **echo** command line.

The test Command for Loops

The **test** command checks to see if certain looping conditions are true. If the condition is true, then the loop continues. If the condition is false, then the loop ends and the next command is executed.

Some options for the test command are:

test -r *file*	True if *file* exists and is readable.
test -w *file*	True if *file* exists and has write permission.
test -x *file*	True if *file* exists and is executable.
test -s *file*	True if *file* exists and has at least one character.

The following sample program uses the **test** command to move only executable files from your current directory to your bin directory (see "Creating a bin Directory for Executable Files," earlier in this section):

```
echo Type in the directory path
read path
for file
     do
          if test -x $file
               then mv $file $path/$file
          fi
     done
```

The directory path is the path from the current directory to the bin directory.

The Conditional Construct case...esac

The conditional construct **case...esac** is a multiple choice construction that allows you to choose one of several patterns and then execute a list of commands for that pattern.

The keyword **in** must begin the pattern statements, and a) must follow the last character of each pattern. The command sequence for each pattern is ended with ;;. The case construction must be ended with **esac** (letters of case reversed). The general format for the case construction is:

```
case characters
in
        pattern)
                command1
                .
                .
                .
                last command
        ;;
        pattern2)
                command1
                .
                .
                .
                last command
        ;;
        pattern3)
                command1
                .
                .
                .
                last command
        ;;
        *)
                command1
                .
                .
                .
                last command
        ;;
esac
```

The case construction will tries to match characters with the first pattern. If there is a match, the program executes the command lines after the first pattern and up to the ;;.

If the first pattern is not matched, then the program proceeds to the second pattern. After a pattern is matched, the program does not try to match any more of the patterns, but goes to the command following **esac**.

The * used as a pattern at the end of the list of patterns allows you to give instructions if none of the patterns are matched. The * means any pattern, so it must be placed at the end of the pattern list if the other patterns are to be checked first.

The following sample shell program uses **case...esac** to determine whether the user can follow directions:

```
echo Enter A, B, or C:
read letter
case $letter
in
     A)
          echo You entered A
     ;;
     B)
          echo You entered B
     ;;
     C)
          echo You entered C
     *)
          echo You did not enter A, B, or C
     ;;
esac
echo end of program
```

Unconditional Control Statement break

The **break** command unconditionally stops the execution of any loop in which it is encountered, and goes to the next command after the **done**, **fi**, or **esac** statement. If there are no commands after that statement, the program ends.

In the example in the previous subsection, a **break** command in place of the **echo** command (after the * pattern) would end the program if an A, B, or C was not entered.

The **continue** and **exit** commands can also be used in shell programs. The **continue** command causes a a program to go immediately to the next iteration of a **do** or **for** loop without executing the remaining commands in the loop.

The **exit** command ends a program at some other point than the end of the file.

Debugging Shell Programs

There are two options to the **sh** command that can be used to debug shell programs:

sh -v	Prints the shell input lines as they are read by the system.
sh -x	Prints commands and their arguments as they are executed.

Both of these commands, when used on a program that does not run properly, stop at the point that causes the program to stop. Use these commands to determine where programs have problems, then edit the programs to correct them.

Modifying Your Login Environment

.profile File

A .profile ("dot" profile) is a file that you can place in your home directory that issues commands to control your shell environment.

Note: *When you log in to the system, the shell first looks for a system file, called the /etc/profile file, for information about your shell environment. The shell then looks in your login directory for a .profile file. If both files exist and information between the two files contradicts, your .profile overrides what is in the /etc/profile file.*

The .profile is a regular file, that can be edited and changed to suit your needs. On some systems you can edit this file yourself; on other systems the system administrator will do this for you.

Caution: *Use care when modifying your .profile file since its contents can affect your whole shell environment.*

Adding Commands to .profile.

By using the CENTIX editors, you can add commands to your .profile just as you can add commands to any other shell program. If, for example, you want a welcome message to appear on your screen every time you log in, add the following as the last line of your .profile file:

```
echo Welcome to CENTIX.
```

Whenever you make changes to your .profile and you want to initiate them in the current work session, type a . and space before .profile. The shell reinitializes your environment by reading and executing the commands in your .profile.

To identify your PT 1500 with the operating system, add the following to your .profile file:

```
TERM=pt
export TERM
```

Using Shell Variables

Several of the variables reserved by the shell can be specified in your .profile. By default, these variables are declared in system files such as /etc/profile.

HOME. The HOME variable gives the path for your login, or home directory. To find your current home directory, type **echo $HOME** and press RETURN.

To change your home directory, enter the following to your .profile:

```
HOME=path
```

where *path* is the new pathname of your home directory.

PATH. The PATH variable gives the system the search path for finding and executing commands.

If you want to see the current values for your PATH variable, enter **echo $PATH** and press RETURN. The following appears on your screen:

```
:/bin:/etc:/usr/bin:/usr/local/bin:/usr/include:
```

The : is a delimiter between the various path names. When you specify a command, the shell looks first in the /bin directory, then in /etc, then /usr/bin, and so on.

To add a directory to your PATH variable, add the full path name to the end of your path in your .profile file.

For example, to add the directory /group/bin to your path, enter the following to the end of .profile:

```
:/bin:/etc:/usr/bin:/usr/local/bin:
/usr/include:/group/bin
```

Caution: *Use care when changing your PATH variable. If you do not specify the correct directories with the PATH variable, you may lose access to some or all of the CENTIX system commands. See your system administrator before making any changes to PATH.*

TERM. The TERM variable tells the shell what kind of terminal you are working on. To identify your PT 1500 to the operating system automatically each time you log in, add the following to your .profile:

```
TERM=pt
export TERM
```

PS1. The PS1 variable specifies your shell prompt. The default for this is the dollar sign ($). To change your shell prompt, add the following to your .profile:

```
PS1=newprompt
```

where *newprompt* is the new prompt that you specify. *Newprompt* can be just about any character or a series of characters.

To specify several words as the new prompt, use quotation marks around the phrase. For example, to make "Your wish is my command" your shell prompt, add the following to your .profile:

```
PS1="Your wish is my command "
```

Printing Files to a Line Printer

To print a file to a line printer, you will use either the **lp** or the **lpr** command. Which command you use depends on how your system has been configured. Check with your system administrator.

Printing Files with the lp command

When you use the **lp** command to print a file or files, you have three choices of where you want your file printed:

1 You can use the default destination (either a printer or a class of printers) that either you or your system administrator has defined. See "Defining Your Default Destination for an lp Request" later in this section.

2 You can select a specific printer.

3 You can select a group—or class—of printers. Your file or files will be printed at the first free printer in that class.

To send a file or files to the default printer, use the **lp** command without specifying a destination:

```
$ lp file(s)
```

where *file(s)* is the file or files that you want printed.

To send a file or files to a particular printer, enter:

```
$ lp -dprintername file(s)
```

where *printername* is the name of the printer to which you want *file(s)* sent.

To send a file or files to a class of printers, enter:

```
$ lp -dclassname file(s)
```

where *classname* is the class of printers to which you want *file(s)* sent.

Check with your system administrator for the names and classes of printers from which you can select.

After you enter the **lp** command, the shell returns to your screen:

```
request id is printername-id (n files)
```

where

- *printername* is the printer to which the file was sent. If you selected a class of printers, this field shows the specific printer within the class to which the file was sent.
- *id* is the unique identification number assigned to your print request by the shell.
- *n* is the number of files you sent with the command.

For a complete description of the **lp** command, see your *CENTIX Operations Reference Manual*.

Using the lp Command with a Printer Connected to a PT 1500

Before any files can be printed to a printer that is connected to a PT 1500, you must enter the **mvtpy** command. This command internally connects the printer with the terminal. Enter the following at the terminal to which the printer is connected:

```
$ mvtpy printername
```

where *printername* is the name that your system administrator assigned to your printer.

The **mvtpy** command must be run each time that you log in on your terminal. It must also be run whenever the lp scheduler, **lpsched**, is turned off, then on again. Only your system administrator should turn **lpsched** on and off.

Cancelling an lp Printer Request

You can cancel a print request in two ways:

- You can cancel a specific print request.
- You can cancel the request that is currently printing at a specific printer. The next available request for that printer automatically prints.

To cancel a specific print request that you made with the **lp** command, use the **cancel** command. Enter:

```
$ cancel printername-id
```

where *printername* is the printer to which the file was sent and *id* is the unique identification number assigned to your print request by the shell.

Note that the printer name and the id number are printed to your screen when you use the **lp** command.

To cancel the request that is currently printing at a specific printer, enter:

```
$ cancel printername
```

The next available request for that printer automatically prints.

Determining the Status of the lp Spooler

You can use the **lpstat** (line printer status) command to have the current status of the lp spooler printed to your terminal.

If you use **lpstat** with no options, the current status of all of the current **lp** requests made under your login name is printed to the screen. The status line for each request includes:

```
printername-id   owner   size (in bytes)   date/time
```

where

- *printername* is the printer to which the file was sent.
- *id* is the unique identification number assigned to your print request by the shell. If you selected a class of printers, this field shows the specific printer within the class to which the file was sent.

If one of the **lp** requests is currently printing, the printer name is displayed at the end of the status line for that request.

The **lpstat** command has several options. Among the information that you can request are the class names and their members, the system default destination that was set up with the **lpadmin** command by your system administrator, and the current status of **lpsched**. See your *CENTIX Operations Reference Manual* for a list of all of the **lpstat** options.

Defining Your Default Destination for an lp Request

Your system administrator may have defined a default destination for your **lp** requests. If not, or if you want to set your own default destination, after you log on, enter:

```
$ LPDEST=destination
$ export LPDEST
```

where *destination* is your default destination.

Each time that you use the **lp** command without specifying a destination, the file automatically prints to this default destination.

Printing Files with the lpr Command

To print a file or files with the **lpr** command, enter:

```
$ lpr file(s)
```

The **lpr** command automatically looks for a parallel printer queue on which to print the file or files. The default name of this queue is SPL. If your system administrator has changed the default name of the printer queue, or if you want your file to print at a printer other than the default printer, you must give the name of the printer queue in the command:

```
$ lpr -q queuename file(s)
```

Ask your system administrator if you need the **-q** option when you print with the **lpr** command.

Note that you cannot cancel a print request made with the **lpr** command.

See your *CENTIX Operations Reference Manual* for more details on the **lpr** command.

Using centreWINDOW

Introduction

The **centreWINDOW** program provides a facility that you can use to divide the terminal screen into "windows" that you can use like separate terminals. You can run up to four programs simultaneously, just as you would if you used four separate terminals. **centreWINDOW** does not interfere with the running of your programs.

Caution: *Do not run* ***centreWINDOW*** *at the terminal that has been designated as the system console. Ask your system administrator which terminal is the system console.*

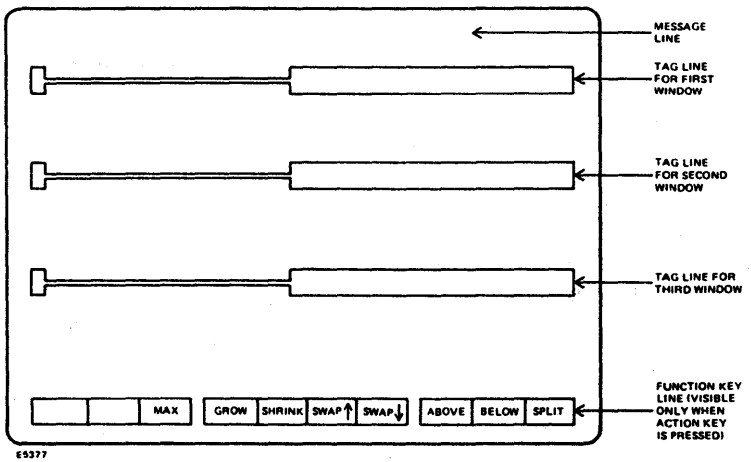
What is a Window?

When you work on a document that is displayed on a computer terminal, the screen serves as a window into that document. It may be useful to imagine that the information in a document (or some other file) is on an electronic roller that moves past the fixed frame of your screen.

When you use **centreWINDOW**, the screen consists of four parts (see Figure 6-1):

- The *message line*. The message line appears at the top of the screen. It holds messages and prompts from application programs. The message line stays at the top of the screen regardless of how many windows are open.
- The *tag line*. The tag line appears at the top of each window. The tag line displays the window number and the name of any application that you are running in that window.
- The *window*. The window is the display area. If only one application is running (that is, only one window is open), the entire screen is available for display. Each window that is opened after that decreases the display area.

Figure 6-1 **The centreWINDOW Screen.**



- The *function key line*. The function key line is displayed at the bottom of the screen when you press the ACTION key on your keyboard. This line shows how you can use the function keys on your keyboard. The function keys, used with the ACTION key, create and manipulate windows. For example, function key F10 is labeled SPLIT in the function key line. When you press ACTION and SPLIT, the current (active) window splits into two windows.

Starting Up centreWINDOW

Your system can be configured so that you automatically have **centreWINDOW** capabilities when you log in. Check with your system administrator.

If your system is not configured with automatic **centreWINDOW** capabilities, activate **centreWINDOW** by entering:

```
$ exec /usr/local/bin/wm
```

Opening Windows

When **centreWINDOW** is activated, your screen automatically becomes window 1. This is indicated by a number 1 on the left side of the tag line at the top of the screen. Unless you open other windows, your terminal screen will remain as one window.

Use the following procedure to open another window:

1 Press the ACTION key on your keyboard. The function key line appears at the bottom of your screen.

2 While holding down the ACTION key, press the function key labeled SPLIT (F10). The terminal screen is cut in half, and a new window, called window 2, is created.

When you open window 2, it becomes the new active window; window 1 becomes inactive. Any programs that are running in window 1, however, will continue to do so.

Note that the tag line for the active window (in this example, window 2) is at full-intensity and the tag lines for other windows (which are inactive) are at half-intensity. The tag line for the active window is always the only tag line at full-intensity.

You can create up to four separate windows by pressing ACTION and SPLIT (F10). Each new window that you create becomes the new active window (indicated by a full-intensity tag line), while the other windows become inactive. This process of opening new windows does not affect programs running in other windows.

Managing the Windows

There are times when you need to change your active window. For instance, if a program running in an inactive window calls for input from the keyboard, the program will read only input into that window. A program in window 2 will not process input from window 4. Keyboard input goes to the active window only. To make another window the active window, you must place the cursor in that window.

Just as you press the ACTION and SPLIT (F10) keys to create new windows, use ACTION and the other function keys to manipulate those windows.

Moving from Window to Window

Two of the ACTION function keys allow you to move freely between windows: ABOVE (F8) and BELOW (F9).

Moving Up

To move from a lower window to the window above it, press ACTION and ABOVE (F8) at the same time. The cursor moves to the next window up, making that window the active one.

You can also move up to the next window by pressing ACTION and the UP ARROW key at the same time. This produces the same result as ACTION/ABOVE (F8).

Moving Down

To move from an upper window to the one below it, press ACTION and BELOW (F9) at the same time. The cursor moves to the next window down, making that window the active one.

You can also move down to the next window by pressing ACTION and the DOWN ARROW key at the same time. This produces the same result as ACTION/BELOW (F9).

Moving Directly to a Specified Window

Instead of moving up and down from window to window, one at a time, you can also move directly to a specified window.

Suppose you have all four windows open, and the cursor is in window 4. You can move directly to window 1 (without having to pass sequentially from window 4 to window 3 to window 2 to window 1) by pressing ACTION and 1 at the same time. The cursor moves directly to window 1. The cursor will move to any number you specify (between 1 and 4, of course), making that window the active one.

Enlarging a Window

The MAX (F3) function key allows you to enlarge the active window so that it occupies the entire screen:

1 Use ACTION/ABOVE or ACTION/BELOW to move the cursor to the window you wish to enlarge.

2 Press ACTION and MAX (F3). The active window expands to fill the entire display area. Windows above the active window are pushed flush against the top of the screen. Windows below the active window are pushed to the bottom.

You can perform the same function by pressing ACTION, CODE, and GROW (F4) at the same time.

To increase the size of the active window one line at a time, press ACTION and GROW (F4).

To increase the size of the active window by several lines, press ACTION/SHIFT/GROW (F4).

Shrinking a Window

There are several methods you can use to make the active window smaller.

- To make the active window shrink by one line, press ACTION/SHRINK (F5).
- To make the active window smaller by several lines, press ACTION/SHIFT/SHRINK (F5).
- To make the active window shrink completely (to 0 lines), press ACTION/CODE/SHRINK (F5).

Swapping Windows

The SWAP UP (F6) and SWAP DOWN (F7) functions allow you to reverse—or swap—the positions of two windows.

To make the active window change places with the window above it, press ACTION/SWAP UP (F6).

To make the active window change places with the one below it, press ACTION/SWAP DOWN (F7).

Closing a Window

As with the other **centreWINDOW** commands, you can close only the window in which you are currently working—the active window. You can do this in one of two ways:

- Press the FINISH key.
- Type **exit** at the shell prompt.

The contents of each window are saved when the windows are closed, and the cursor moves to the window that was last opened. When you close the last window, you also log out of CENTIX.

For more information on the **centreWINDOW** facility, see your *CENTIX Operations Reference Manual*.

Using Removable Media

Disk cartridges, QIC tapes, and half-inch tapes provide portable media on which to store information. In the XE 500 system, they can provide the source for the initial system software and for future software updates. They can also be used as backup storage devices for important files in the XE 500 system.

This section discusses how to use disk cartridges and QIC tapes. For half-inch tape handling information, refer to the tape drive's documentation.

Using Disk Cartridges

Handling Disk Cartridges

A disk cartridge consists of a rigid disk enclosed within a protective plastic cartridge. The disk is covered with magnetic material on both sides. Every disk cartridge comes with its own case, which serves as a dust jacket.

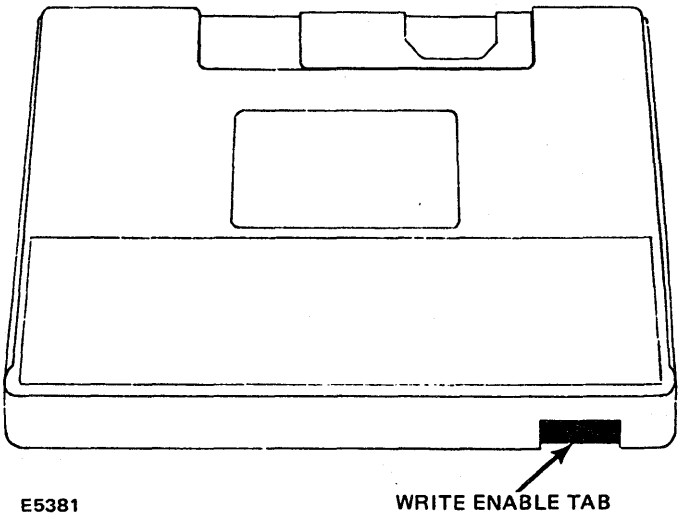
There are some simple rules to follow when you handle disk cartridges.

- Use care when handling disk cartridges. Sudden shocks can cause internal damage to the cartridge.
- Store disk cartridges upright in their dust jackets. Keep them out of direct sunlight. Store them in a dry area at normal room temperature.

- Store disks away from magnetic or electrical devices.
- Do not use disks that have recently been in an extremely hot or cold environment. Wait for them to adjust to room temperature.

Caution: *DO NOT remove the red write enable tab at the side of the disk cartridge. (See Figure 7-1.)*

Figure 7-1 **Disk Cartridge**



Operating the Cartridge Slot

You will need to know how to operate the cartridge slot when using disk cartridges. There are only two controls for the cartridge slot—the release latch and the slot door. Referring to Figure 7-2, locate these two parts of the cartridge slot.

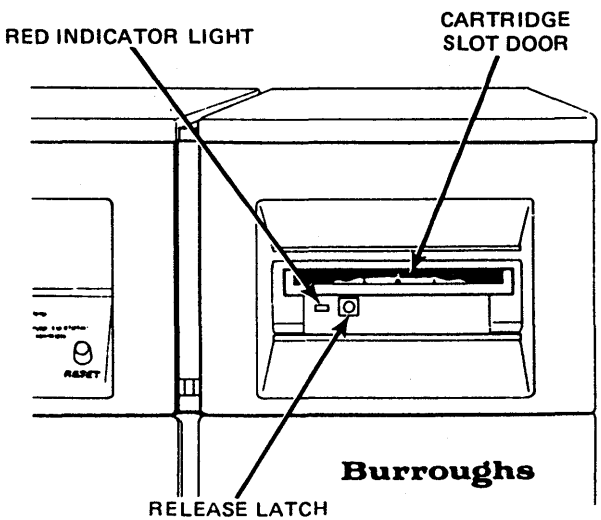
The release latch releases the door from the locked position. The door must be manually opened before you can insert or remove a disk cartridge.

Notice that there is a red indicator light to the left of the release latch. This light goes on whenever the system is retrieving or storing information on the disk cartridge.

While a disk cartridge is inserted, the disk spins at a certain operating speed. The indicator light flashes whenever the disk is being brought down from its operating speed.

Caution: *Never push the release latch when the indicator light is on. Also, you should never open the slot door to remove a disk cartridge while the light is flashing. These actions could damage the disk and/or the drive unit.*

Figure 7-2 **Disk Cartridge Slot**



Inserting Disk Cartridges

Caution: *The XE 500 base enclosure should always be powered up before a disk cartridge is inserted.*

Use the following procedure to insert a disk cartridge.

1 Be sure that the XE 500 base enclosure is powered up.

2 Press the release latch at the cartridge slot.

3 Open the cartridge slot door.

Caution: *To avoid internal damage to the cartridge drive when inserting a disk cartridge, be sure that the cartridge slot door is fully open.*

4 Remove the disk cartridge from its dust jacket.

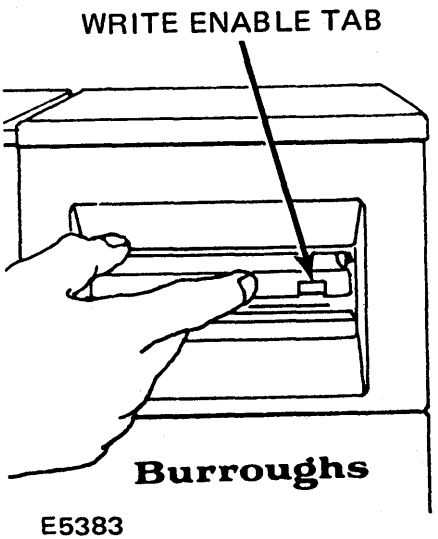
5 Insert the disk cartridge as shown in Figure 7-3. The side that has the write enable tab should be facing out, and the write enable tab should be to the right.

6 Push the disk cartridge into the drive until you hear a "click."

7 Close the door. Note that the red indicator lights up when the disk cartridge drive is being used.

Caution: *Do not turn off power to the XE 500 base enclosure after a disk cartridge has been inserted.*

Figure 7-3 **Inserting the Disk Cartridge**



Removing a Disk Cartridge

Caution: *Attempt to remove a disk cartridge only when the disk drive indicator light is off. Never press the release latch when the light is on or blinking.*

Use the following procedure to remove a disk cartridge.

1 Make sure the disk drive indicator light is off.

2 Press the release latch.

After the cartridge release latch is pressed, the indicator light at the cartridge slot blinks for about 30 seconds (until the disk stops spinning).

Caution: *After pressing the release latch, do not open the drive door until the blinking stops.*

3 Be sure that the indicator light has stopped blinking.

4 Open the cartridge slot door.

As you fully open the door, the disk automatically ejects from the drive. Hold onto the disk during this process to prevent it from falling.

5 Place the disk cartridge into its dust jacket.

6 Close the cartridge slot door.

Note: *Always keep the cartridge slot door closed when not in use.*

Using QIC Tapes

Handling QIC Tapes

Here are some rules to follow when handling QIC tapes:

- Use care when handling QIC tapes. Do not touch or manually move the magnetic tape inside the cartridge.
- Store cartridges in their cases, in a dry area at room temperature. Unlike half-inch tapes, it is not necessary to store QIC tapes vertically.
- Keep QIC tapes away from magnetic devices, such as CRT screens.

Under certain conditions, the QIC tape should be *packed* (that is, rewound) to ensure that the tape operates properly during I/O operations. Before using a QIC tape, execute the **MQic Retension** command to rewind the tape if any of the following conditions apply:

- Occurrence of excessive read/write errors (more than 50 software errors per pass).
- Exposure of the cartridge to temperatures outside the range of 40°F to 110°F (5°C to 44°C).
- Prolonged storage (more than two weeks) of the cartridge.
- Physical shock to the cartridge, such as dropping it or dropping something on it.

Figure 7-4 shows the components of a QIC tape. Figure 7-5 shows the positions of the write protect plug for enabling and disabling write operations to the tape.

Figure 7-4 **Components of a QIC Tape**

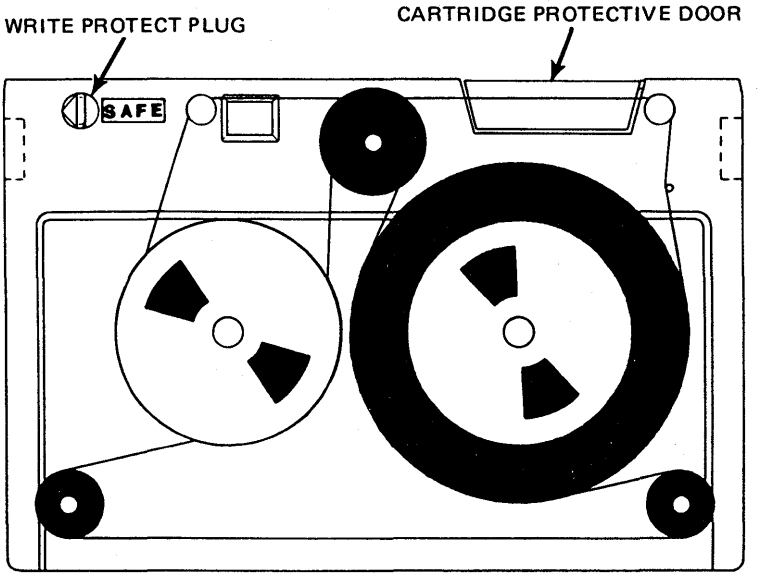
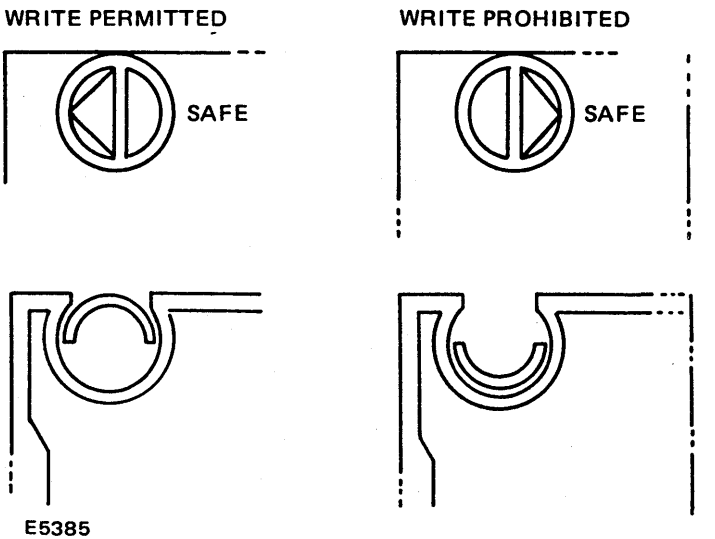


Figure 7-5 **QIC Tape Write Protect Plug Positions**



Operating the QIC Tape Drive

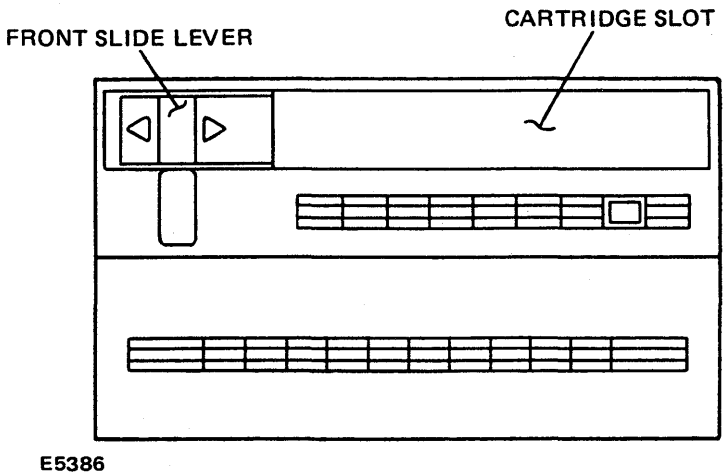
You will need to know how to operate the QIC tape drive when using QIC tapes. There are only two controls for the drive—the front slide lever and the cartridge slot. Referring to Figure 7-6, locate these two parts of the drive.

The front slide lever controls the locking of the QIC tape in the drive.

Notice that there is a red indicator light on the right side of the drive. This light goes on whenever the system is retrieving or storing information on the drive.

Caution: *If you push the front slide lever when the indicator light is on, you could interrupt a write operation to the tape.*

Figure 7-6 **The QIC Tape Drive**



Inserting a QIC Tape

Use the following procedure to insert a QIC tape.

1 Be sure that the XE 500 base enclosure is powered up.

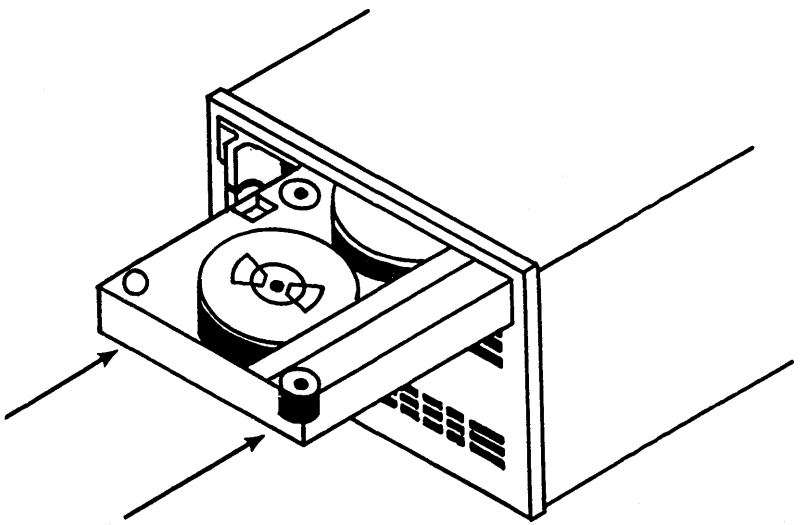
2 Remove the QIC tape from its case.

3 Insert the QIC tape into the drive with the cartridge protective door facing left and the base plate of the cartridge facing down, as shown in Figure 7-7.

4 Push the QIC tape into the drive.

5 When the QIC tape is fully inserted, it descends into the tape drive. At this time, the protective door on the cartridge opens.

Figure 7-7 **Inserting the QIC Tape**



E7517

6 Move the front slide lever to the right until it reaches the lever stop. This secures the cartridge and brings the head assembly to its correct operating position.

7 The QIC tape is now loaded. Note that the red indicator lights up when the QIC tape drive is being used.

Removing a QIC Tape

Use the following procedure to remove a QIC tape.

1 Make sure the drive indicator light is off.

2 Move the front slide lever to the left until it reaches the lever stop.

3 The head assembly in the drive retracts, and the protective door on the cartridge closes. A cartridge ejector automatically raises the cartridge out of the drive and slowly pushes it forward.

4 Grasp and remove the QIC tape, and return it to its case, with the protective door to the inside of the case and the base plate down.

Glossary

Applications Processor (AP). Processor board in the XE 500 system that runs the CENTIX operating system.

AP. See Applications Processor.

ASCII (American National Standard Code for Information Interchange). Control and graphic character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange between data communications systems.

background process. Process that, once started up, runs underneath any active processes, with no interaction with the user through the terminal.

B 20. Burroughs microcomputer from the clustered workstation series.

block. On a disk device, a 512-byte subdivision of data on the disk. Also referred to as a sector.

block device. A hardware device that handles I/O data in 1024 bytes (1 kB) blocks. The I/O size is controlled by the operating system's buffer size and is independent of the user's I/O size. Disk and tape devices can be configured as block devices.

Bourne Shell. A command-oriented shell used to search for programs in specific places on the CENTIX file system.

BTOS. B 20 Operating System. All XE 500 boards except the Applications Processor run a version of BTOS. BTOS-based processors handle all of the actual data transfers for CENTIX between the XE 500 and I/O devices (such as disk drives, tape drives, terminals, and modems).

CENTIX. Burroughs version of the System V UNIX operating system.

CENTIX BASIC. ANSI-standard BASIC adapted for use with CENTIX.

CENTIX C. Standard C Language adapted for use with CENTIX operating system.

CENTIX COBOL. High-level programming language conforming to ANSI X3.33-1974.

CENTIX FORTRAN. CENTIX adaptation of high-level programming language conforming to ANSI standard X3.9-1978.

CENTIX Pascal. CENTIX adaptation of high-level programming language suited for large projects.

CENTIX shell. Command interpreter; program acting as interface between operating system and users.

centrEASE. A menu driven, interactive facility that you can use to perform many administrative tasks on the CENTIX system.

centreCAP. A programming tool that can be used to create function-key driven user interfaces.

centreSCREEN. The CENTIX forms programming tool through which forms can be designed and retrieved.

centreSNA. Burroughs software products; provide an interface for CENTIX application products to SNA networks.

centreSPHERE. Environmental and application software.

centreWINDOW. Allows multiple windows at a PT 1500, with each window running its own program.

centrOFFICE. An office information management system for efficient communication and information accessing, including word processing, spreadsheet, and mail exchange.

character device. A hardware device that handles raw data streams. The size of I/O transfers in raw data streams are determined either by the software design of the device itself (for terminals and printers) or by the program controlling the device (for disks and tapes).

Cluster Processor (CP). Board in XE 500 system; runs communications software and supports PT 1500 terminals, B 20 workstations, a parallel printer, and up to three RS-232-C serial devices.

CMS. See **Computer Management System.**

COBOL Animator. Source level debugging tool that allows the programmer to interact with an executing COBOL program.

Computer Management System (CMS). Operating system; run-time system and post-compilation system.

console. The terminal designated by the system software for use by the system administrator.

CP. See **Cluster Processor.**

daemon. A system background process.

data communication (data comm). The transfer of data between a data source and data link using one or more data links according to the designated protocol.

demand paging. A form of memory management that keeps in on-board memory only those parts of the program code and data required for execution.

device. A terminal, printer, disk, tape, or other input/output medium that can be attached to the system. A device can be physical or logical.

device file. In the CENTIX file system, a file in the /dev directory that represents a terminal, printer, disk, tape, or other input/output device.

directory. In CENTIX, a directory is a list of files that are assigned to the directory. A directory can also contain other directories.

disk cartridge. Magnetic disk storage medium utilizing a hard disk enclosed in a portable cartridge. Disk cartridges are used with a disk cartridge drive of an XE 500 base enclosure.

disk extent. One or more contiguous disk sectors that contain all or part of a file.

Disk Processor (DP). Processor board in an XE 500 system that is formed by connecting SC to SP. The DP supports I/O to half-inch magnetic tape drives and MD3 disks.

DP. See
Disk Processor.

dumb terminal. Unprogrammable terminal that uses ASCII code.

/etc/checklist. File that lists the CENTIX file systems that are checked by **fsck** , the file system checking program, at boot time.

/etc/getty. Process that readies terminal connections for login.

/etc/gettydefs. File that contains line communications information on terminals. It is read by the /etc/getty process.

/etc/inittab. A CENTIX file that defines the CENTIX terminal assignments and the CENTIX processes that are started at boot time.
(SC)

/etc/passwd. File that lists information about each user on the system, including the user's login name, password, home directory, and so on.

/etc/profile. A shell script that defines the users' terminal environment when the system is booted.

/etc/rc. File used as a general purpose start-up file for various background processes such as the lp spooler scheduler and cron.

File Processor (FP). Processor board in an XE 500 system that supports I/O operations to disk devices.

file system. In CENTIX, a collection of files that are all stored on the same logical disk device. A file system must be attached to, or is "subordinate to," a directory. The file system physically contains the files that are logically contained in that directory. The term can also be used, as in "the CENTIX file system," to describe the entire hierarchy of directories, specific file systems, and files in a CENTIX system.

FP. See
File Processor.

home directory. For a user, the directory into which the user is automatically placed when he or she logs onto the system.

Indexed Sequential Access Method (ISAM). Programming tool that uses an index to sequence file records on disk and to access those records directly.

INGRES. Relational data base management system.

inode. In a CENTIX file system, there is one inode for each file and directory in the file system. The inode contains status information for its file or directory, such as the size, its owner and permissions, its disk address list, and whether it is a directory, an ordinary file, or a special file.

I/O. Input/output.

ISAM. See
Indexed Sequential Access Method.

kernel. Portion of the CENTIX operating system that controls system processes and allocates system resources.

Large-Scale Integration (LSI). Monolithic integrated circuits of very high density.

LSI. See **Large-Scale Integration.**

Master Commands. The BTOS commands that can be accessed through CENTIX to administer the BTOS portion of the operating system.

Master Utilities. Utilities that are invoked when the BTOS Master Commands are used.

MCommands. See **Master Commands.**

MD3. Enclosure containing up to three Memorex 166 SMD disk drives.

ME. See **Memory Expansion Board.**

Memory Expansion Board (ME). Board attached to a processor board to supply 1/2 M-byte or 1 M-byte additional memory capacity.

Memory Management Unit (MMU). Part of the AP that supports multiprogramming and demand paging.

mixed system. An XE 500 system that contains a complete BTOS operating system and a complete CENTIX operating system.

MMU. See **Memory Management Unit.**

multiuser mode. An operating state defined in the /etc/inittab*nn* files. In multiuser mode, user terminals are readied for login.

MUtilities. See **Master Utilities.**

ofcli. CENTIX command used to access the BTOS Command Line Interpreter (CLI) mode, from which BTOS MCommands can be initiated.

partition. The name of a BTOS file that is associated with a CENTIX logical disk device.

path name. For a CENTIX file, the name that identifies the file's position in the CENTIX file system. A complete (absolute) path name always begins with /, which stands for the root directory.

Pilf factor. A value that can be specified when you create a file system to control the size of the blocks of data that are moved in and out of the file system in I/O operations.

piping. Linking of programs so that the output of one program becomes the input for another program.

port. The part of a data processor dedicated to a single data channel for receiving data from, or transmitting data to, one or more external remote devices.

printer spooler. A system service that manages the transfer of data from disk files to printers.

QIC. See **Quarter-Inch Cartridge Tape.**

Quarter-Inch Cartridge (QIC) Tape. Magnetic tape storage medium that utilizes quarter-inch-wide tape enclosed in a portable cartridge. QIC tapes are used with a disk cartridge drive of an XE 500 base enclosure.

raw device. A block device configured to accept data one character at a time, rather than in blocks.

root. The base directory of the CENTIX file system. Every CENTIX directory must either be subordinate to root, or subordinate to a directory that is subordinate to root, or subordinate to a directory that is subordinate to a directory that is subordinate to root, and so on. In a file path name, root is represented by a slash (/).

Run-Time System. CENTIX shell commands and software to support an office environment running office application programs.

saf. Command entered that initiates the centrEASE administrative facility.

SCCS. See **Source Code Control System.**

sector. See **block.**

shell. The portion of the CENTIX operating system that provides a user interface to the kernel.

shell script. An executable CENTIX file that contains a program comprised of shell commands.

single user mode. An operating state defined in the /etc/inittab*nn* files. In single user mode, only the system console can access the system.

SMD. See **Storage Module Device.**

SNA. See **Systems Network Architecture.**

Sort/Merge. Programming tool that provides sequencing of file records and merging of sorted records from more than one file.

Source Code Control System (SCCS). A group of software commands that control and account for changes to text files.

SP. See **Storage Processor.**

special file. See **device file.**

Storage Module Device (SMD). 132-byte (formatted) Memorex 166 Disk Drive.

Storage Processor (SP). Processor board in XE 500 system; controls half-inch magnetic tape.

superblock. The portion of a CENTIX file system that contains descriptions of the file system, including the file system name, its size in blocks, the number of blocks reserved for inodes, the free inode list, and the free block list.

superuser. The name by which the system administrator is called in CENTIX documentation. To become superuser, the administrator signs onto the system as "root".

system bus. Path over which the system processors communicate.

Systems Network Architecture (SNA). A formal set of rules for designing, building, and operating the components of a data communications network.

terminal. A device, usually equipped with a keyboard and a display, which is capable of sending and receiving information over a communication channel.

Terminal Processor (TP). Processor board in XE 500 system that supports a parallel printer and up to ten RS-232-C serial devices.

TP. See **Terminal Processor.**

UNIX. AT&T Bell Laboratories operating system designed for application program development on various computer systems.

volume. In BTOS, the complete file system unit of information stored on a formatted disk.

XE 550 System. Burroughs multiprocessor computer; runs CENTIX, a UNIX-based operating system.

Index

I

K

L

M

U

V

W

X

Y

Title: _____

Form Number: _____ Date: _____

Burroughs Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information.

Please check type of suggestion: ☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments: _____

_____

_____

_____

Name _____

Title _____

Company _____

Address _____
Street City State Zip

Telephone Number ( ) _____
Area Code

Title: _____

Form Number: _____ Date: _____

Burroughs Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information.

Please check type of suggestion: ☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments: _____

_____

_____

_____

Name _____

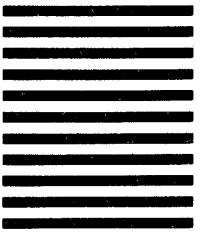
Title _____

Company _____

Address _____
Street City State Zip

Telephone Number ( ) _____
Area Code

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY CARD

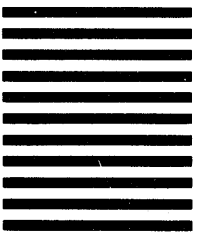
FIRST CLASS PERMIT NO. 817 DETROIT, MI 48232

POSTAGE WILL BE PAID BY ADDRESSEE

Burroughs Corporation
Production Services – East
209 W. Lancaster Avenue
Paoli, Pa 19301 USA

ATTN: Corporate Product Information

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 817 DETROIT, MI 48232

POSTAGE WILL BE PAID BY ADDRESSEE

Burroughs Corporation
Production Services – East
209 W. Lancaster Avenue
Paoli, Pa 19301 USA

ATTN: Corporate Product Information