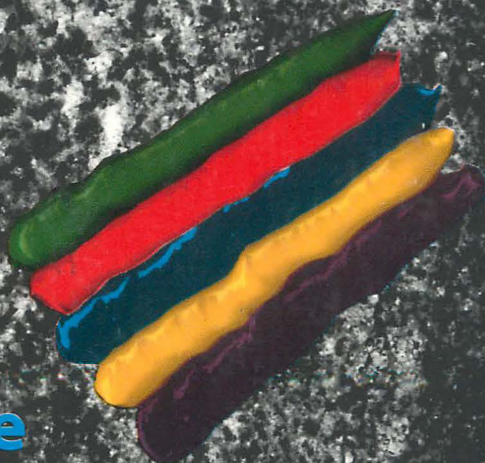


AMIGA[®] OS 3.1

DOS



 Commodore

COPYRIGHT

Copyright © 1993 by Commodore Electronics Limited. All rights Reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Commodore Electronics Limited.

If this product is being acquired for on behalf of the United States of America, its agencies and/or instrumentalities, it is provided with RESTRICTED RIGHTS, and all use, duplication, or disclosure with respect to the included software and documentation is subject to the restrictions set forth in subdivision (b) (3) (ii) of The Rights in Technical Data and Computer Software clause at 252.227-7013 of the DOD FAR. Unless otherwise indicated, the manufacturer/integrator is Commodore Business Machines, Inc., 1200 Wilson Drive, West Chester, PA 19380.

The material set forth in the *AmigaDOS User's Guide* is adapted from *The AmigaDOS Manual*, 2nd Edition, Copyright © 1987 by Commodore-Amiga, Inc. used by permission of Bantam Books. All Rights Reserved. The Times Roman, Helvetica Medium, and Courier fonts included in the Fonts directory on the Fonts disk are Copyright © 1985, 1987 Adobe Systems, Inc. The CG Times, Univers Medium, and LetterGothic fonts included on the Fonts disk are Copyright © 1990 by Agfa Corporation and under license from the Agfa Corporation.

DISCLAIMER

With this document Commodore makes no warranties or representations, either expressed or implied, with respect to the products described herein. The information presented herein is being supplied on an "AS IS" basis and is expressly subject to change without notice. The entire risk as to the use of this information is assumed by the user. IN NO EVENT WILL COMMODORE BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY CLAIM ARISING OUT OF THE INFORMATION PRESENTED HEREIN, EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITIES OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE LIMITATION OF IMPLIED WARRANTIES OR DAMAGES, SO THE ABOVE LIMITATIONS MAY NOT APPLY.

TRADEMARKS

Commodore, the Commodore logo, CBM, and AUTOCONFIG are trademarks of Commodore Electronics Limited in the United States and other countries. Amiga, AmigaDOS, Kickstart, Workbench and Bridgeboard are trademarks of Commodore-Amiga, Inc. in the United States and other countries.

MS-DOS is a registered trademark of Microsoft Corporation. CrossDOS is a trademark of Consultron. Compugraphic, CG, and Intellifont are registered trademarks of Agfa Corp. CG Triumvirate is a trademark of Agfa Corp. CG Times is based on Times New Roman under license from The Monotype Corporation plc. Times New Roman is a registered trademark of Monotype Corporation. Univers is a registered trademark of Linotype AG. Universe is under license from Haas Typefoundry Ltd. Diablo is a registered trademark of Xerox Corporation; Epson is a registered trademark of Epson America, Inc.; IBM and Proprietary XL are registered trademarks of International Business Machines Corp; Apple, Macintosh, and Imagewriter are trademarks of Apple Computer, Inc.; LaserJet and LaserJet PLUS are trademarks of Hewlett-Packard Company; NEC and Pinwriter are registered trademarks of NEC Information Systems; Okidata is a registered trademark of Okidata, a division of Oki America, Inc.; Okimate 20 is a trademark of Okidata, a division of Oki America, Inc. This document may also contain references to other trademarks which are believed to belong to the sources associated therewith.

Coverdesign and Print by Village Tronic

Village Tronic Marketing GmbH, Wellweg 95, 31157 Sarstedt, Germany

This book was produced using a variety of Commodore systems by Kitsel Outlaw, Ross Hippely, Barbara Siwirski, and Carina Ahren.

P/N: 371085-01

Table of Contents

Chapter 1

Selecting an Interface

Choosing Your Interface.....	1-2
Workbench Users.....	1-2
Shell Users.....	1-3
AmigaDOS Tasks	1-3

Chapter 2

Understanding the AmigaDOS Shell

About the Shell.....	2-1
Opening Shell Windows	2-2
Closing Shell Windows.....	2-3
Using the Shell	2-3
Command Line Editing and Control	2-5
Using the Command History	2-6
Copying and Pasting.....	2-7
Working with the Shell.....	2-9

Chapter 3

Working With AmigaDOS

Managing Files, Directories, and Disks	3-1
File System Terms	3-2
File Management.....	3-2
Devices	3-3
Directories	3-5
Files	3-5
.Info Files	3-5
Naming Conventions	3-6
Keywords	3-7
Command Line Basics	3-7
Files, Programs, Commands, and Scripts.....	3-7
Files	3-8
Programs.....	3-8
Commands.....	3-8
Scripts.....	3-8
Search Path	3-9
Current Directory.....	3-10
Types of Commands	3-11
AmigaDOS Command Structure	3-12
Special AmigaDOS Characters.....	3-14
Command Line Characters	3-14
Pattern Matching	3-16
Wildcard Characters.....	3-16
Redirection.....	3-18
Angle Brackets.....	3-18
Running Programs	3-20
Running Programs in the Background	3-21
Refining Your AmigaDOS Environment	3-22

Chapter 4

Using the Editors

ED	4-1
Starting ED.....	4-3
Using ED.....	4-4
Immediate Commands.....	4-4
Moving the Cursor in Immediate Mode.....	4-4
Inserting Text in Immediate Mode.....	4-6
Deleting Text in Immediate Mode.....	4-6
Changing Case in Immediate Mode.....	4-7
Extended Commands.....	4-7
Using String Delimiters.....	4-8
Using a File Requester.....	4-8
ED Menus.....	4-8
Enabling Expanded Menus.....	4-9
Project Menu.....	4-11
Edit Menu.....	4-12
Movement Menu.....	4-13
Search Menu.....	4-13
Settings Menu.....	4-15
Set FN Key.....	4-15
Special Key Mappings.....	4-16
Command Menu.....	4-18
Other ED Commands.....	4-18
Repeating Commands in Extended Mode.....	4-20
Customizing ED.....	4-21
Set Menu Item.....	4-21
Printing From ED.....	4-23
Quitting ED.....	4-24
ARexx Support.....	4-24
ED/ARexx Example Program.....	4-25
MEmacs.....	4-27
Starting MEmacs.....	4-27
MEmacs Commands.....	4-28
Menu Commands.....	4-30
Project Menu.....	4-31
Edit Menu.....	4-33
Window Menu.....	4-35
Move Menu.....	4-35
Line Menu.....	4-36
Word Menu.....	4-37

Search Menu.....	4-38
Extras Menu.....	4-39
Commands Not in Menus.....	4-41
Customizing MEMacs.....	4-42
Quitting MEMacs.....	4-43
EDIT.....	4-43
Starting EDIT.....	4-44
EDIT Commands.....	4-45
Selecting the Current Line.....	4-46
Editing the Current Line.....	4-47
Inserting and Deleting Lines.....	4-47
Editing Line Windows.....	4-48
Splitting and Joining Lines.....	4-50
Renumbering Lines.....	4-50
Verifying Lines.....	4-51
Inspecting the Source File.....	4-51
Making Global Changes.....	4-52
Changing Command, Input, and Output Files.....	4-53
Ending EDIT.....	4-56

Chapter 5

Using Scripts

Understanding Scripts.....	5-1
Kinds of Scripts.....	5-2
When to Use ARexx.....	5-2
Simple Scripts.....	5-3
Automatic Scripts.....	5-3
Special Script Characters.....	5-3
Script Commands.....	5-5
Script-Specific Commands.....	5-6
Dot Commands.....	5-6
Allowing Arguments.....	5-7
Substitution.....	5-8
Defaults.....	5-9
Comments.....	5-10
Nesting Commands.....	5-10
Interactive Script Files.....	5-11
Repeating Commands.....	5-12
Ending a Script.....	5-12

Condition Flags	5-13
Debugging Script Files	5-14
Using Environment Variables	5-14
Creating Environment Variables	5-16
SET	5-16
SETENV.....	5-16

Chapter 6

AmigaDOS Command Reference

Command Documentation	6-5
Format.....	6-6
Template	6-8
Command Listing.....	6-10
ADDBUFFERS.....	6-10
ALIAS.....	6-11
ASK.....	6-12
ASSIGN.....	6-13
AVAIL.....	6-17
BREAK.....	6-18
CD.....	6-19
CHANGETASKPRI.....	6-21
COPY.....	6-22
CPU.....	6-24
DATE.....	6-26
DELETE.....	6-28
DIR.....	6-29
DISKCHANGE.....	6-32
ECHO.....	6-32
ED.....	6-33
EDIT.....	6-34
ELSE.....	6-34
ENDCLI.....	6-35
ENDIF.....	6-35
ENDSHELL.....	6-36
ENDSKIP.....	6-36
EVAL.....	6-37
EXECUTE.....	6-38
FAILAT.....	6-39
FAULT.....	6-40

FILENOTE.....	6-41
GET.....	6-42
GETENV.....	6-43
ICONX.....	6-43
IF.....	6-45
INFO.....	6-46
INSTALL.....	6-47
JOIN.....	6-48
LAB.....	6-49
LIST.....	6-49
LOADRESOURCE.....	6-52
LOADWB.....	6-53
LOCK.....	6-54
MAGTAPE.....	6-55
MAKEDIR.....	6-56
MAKELINK.....	6-57
MOUNT.....	6-57
NEWCLI.....	6-59
NEWSHELL.....	6-59
PATH.....	6-62
PROMPT.....	6-64
PROTECT.....	6-65
QUIT.....	6-66
RELABEL.....	6-67
REMRAD.....	6-68
RENAME.....	6-68
REQUESTCHOICE.....	6-69
REQUESTFILE.....	6-70
RESIDENT.....	6-72
RUN.....	6-74
SEARCH.....	6-76
SET.....	6-77
SETCLOCK.....	6-78
SETDATE.....	6-79
SETENV.....	6-79
SETFONT.....	6-80
SETKEYBOARD.....	6-81
SKIP.....	6-82
SORT.....	6-84
STACK.....	6-85
STATUS.....	6-85
TYPE.....	6-86
UNALIAS.....	6-87
UNSET.....	6-87

UNSETENV	6-88
VERSION	6-88
WAIT	6-89
WHICH	6-90
WHY	6-91
System Commands	6-92
ADDDATATYPES	6-92
BINDDRIVERS	6-92
CONCLIP	6-93
IPREFS	6-93
SETPATCH	6-94

Chapter 7

Workbench-Related Command Reference

Preferences Editors	7-4
Font	7-5
IControl	7-5
Input	7-6
Locale	7-6
Overscan	7-7
Palette	7-7
Pointer	7-8
Printer	7-8
PrinterGfx	7-9
PrinterPS	7-9
ScreenMode	7-10
Serial	7-10
Sound	7-11
Time	7-11
WBPattern	7-12
Commodities Programs	7-12
AutoPoint	7-13
Blanker	7-14
ClickToFront	7-15
CrossDOS	7-15
Exchange	7-16
FKey	7-16
MouseBlanker	7-17

NoCapsLock.....	7-17
Other Workbench-Related Tools and Programs.....	7-18
Calculator.....	7-18
Clock.....	7-19
CMD.....	7-20
DiskCopy.....	7-21
FixFonts.....	7-22
Format.....	7-23
GraphicDump.....	7-25
IconEdit.....	7-26
InitPrinter.....	7-26
Intellifont.....	7-26
KeyShow.....	7-27
MEmacs.....	7-27
More.....	7-27
MultiView.....	7-29
NoFastMem.....	7-32
PrepCard.....	7-32

Chapter 8

Command Examples

Basic Tasks.....	8-1
Opening a Shell Window.....	8-1
Running Programs from the Shell.....	8-2
Stopping a Program.....	8-2
Changing the Current Directory.....	8-3
Changing the Search Path.....	8-4
Displaying the Contents of a Directory.....	8-4
Copying Files and Directories.....	8-7
Creating a User-startup File.....	8-8
Creating an Assignment.....	8-9
Accessing the Expanded ED Menus.....	8-10
Working with a Single Shell.....	8-10
Attaching Icons.....	8-11
Creating Scripts Conveniently.....	8-12
Occasional Tasks.....	8-12
Creating Aliases To Reduce Keystrokes.....	8-13
Customizing NEWSHELL.....	8-13
Modifying the Prompt.....	8-14
Creating a Custom Ram Disk Icon.....	8-15

Deleting Files with Icons	8-16
Testing Commands	8-16
Creating a Script to Move Files	8-17
Deleting with Interactive DIR	8-18
Generating Scripts with LIST LFORMAT	8-19
Customizing LIST Output	8-20
Using ICONX to Run Scripts	8-20
Preventing Displayable Output From Scripts	8-21
Entering and Testing ARexx Macros	8-21
Sorting and Joining Files	8-21
Advanced Tasks	8-22
Testing Software Versions	8-22
Flushing Unused Fonts and Libraries	8-22
AmigaDOS Loops Using EVAL	8-23
Using PIPE:	8-25
Recursive AmigaDOS Command Scripts	8-26

Appendix A

Error Messages

Appendix B

Additional Amiga Directories

DEVS:	B-3
Device Files	B-3
Other Files	B-4
Using Mount Files or a MountList	B-4
Creating a Mount File or MountList Entry	B-5
S: Directory	B-8
ED-Startup	B-8
SPat, DPat	B-9
PCD	B-9
L: Directory	B-10
Aux-Handler	B-10
Queue-Handler (PIPE:)	B-11
Port-Handler	B-11
CrossDOSFileSystem	B-11

FileSystem_Trans B-12
CDFileSystem B-12
FONTS:..... **B-12**
 Bitmap Fonts B-12
 Outline Fonts B-13
LIBS: Directory **B-13**
REXX: **B-15**
LOCALE: **B-15**
ENVARC: **B-16**
ENV: **B-16**
CLIPS: **B-16**
T:..... **B-16**
Classes **B-17**
C: **B-17**

Appendix C

Using Floppy-Only Systems

Making Commands Resident..... **C-1**
Preloading Resources **C-2**
Using ASSIGN's PATH Option..... **C-2**
Removing Files From Your Workbench Disk..... **C-3**
 Files You Can Delete C-4
 Files To Avoid Deleting C-4
Using the Ram Disk..... **C-5**
 Copying From One Disk to Another C-6
 Recoverable Ram Disk..... C-6
 Bootable RAD: C-7

Appendix D

Advanced AmigaDOS Features

Customizing the Window	D-1
Public Screens - PUBSCREEN Option	D-1
Customizing the Shell.....	D-2
Using Aliases	D-2
Changing the Prompt	D-3
Using Escape Sequences.....	D-3
Customizing Startup Files.....	D-6
Editing Startup Files.....	D-7
Common Additions to the Startup Files.....	D-8
Using PIPE:	D-9

Glossary

Index

Welcome

The Commodore® Amiga® line of personal computers offers a unique combination of versatility, computing power, and usability. The fast, multitasking Amiga operating system allows users at any level of experience to take advantage of their system's resources.

AmigaDOS™ is the Amiga Disk Operating System. A disk operating system is software that manages data manipulation and control on the computer, such as:

- Providing a filing system that organizes the data that programs use and produce
- Handling information storage and retrieval from floppy disks, hard disks, and other storage media
- Providing an interface to peripheral devices, such as printers and modems

AmigaDOS provides a Command Line Interface (CLI), which means that you work with it through typed commands. Some of these commands parallel familiar Workbench™ operations, such as Copy, Rename, and Format Disk. There are also advanced commands that allow you to create scripts for performing repetitive tasks, to monitor the use of memory, and to perform other tasks unavailable through the Workbench. The commands are entered through a special window, known as a Shell window. Shell windows open on the Workbench screen and are similar to other Workbench windows, except that Shell windows only display text.

Together AmigaDOS and the Amiga Shell offer you a powerful and flexible operating environment with these features.

Operating System Features

- Complete control over all aspects of Amiga operation
- Hierarchical file system
- Filenames up to 30 characters, upper/lower case preserved without case-sensitivity
- Configurable command search path
- Pattern matching
- Background command processing
- Many commands internal, others can be made memory resident
- Shared libraries
- Multiple file systems supported, including CrossDOS (MS-DOS file system)

Shell Features

- Multiple, independent Shell windows
- Shell windows sizable, draggable, depth-adjustable
- Configurable prompt, font, and text color and style
- Command history and command line editing
- Fast character-mapped display
- Aliases
- Local and global environment variables
- Scripting
- Command input and output redirection
- Multiple directory assignment
- Copy and paste text among console windows
- ARexx support

Using this Manual

This manual, which should be used in conjunction with the *Workbench User's Guide*, describes the AmigaDOS software, its components, and how to use it. It assumes that you are familiar with Workbench, but have never worked with AmigaDOS. If this is the case, we recommend that you read through the entire manual to learn the concepts associated with the Amiga operating system before beginning to use it. After you have familiarized yourself with AmigaDOS, use this manual as a reference tool when executing commands or writing programs or scripts.

The following is a brief description of each chapter and appendix:

Chapter 1. Selecting an Interface: This chapter gives information to help you determine when to use AmigaDOS rather than Workbench.

Chapter 2. Understanding the AmigaDOS Shell: This chapter describes the AmigaDOS Shell in detail.

Chapter 3. Working with AmigaDOS: This chapter describes the file management system, types of commands, and components of AmigaDOS commands.

Chapter 4. Using the Editors: This chapter provides a full explanation for using the ED text editor and command listings for the MEMacs and EDIT text editors.

Chapter 5. Using Scripts: This chapter describes AmigaDOS scripts and how to create them.

Chapter 6. AmigaDOS Command Reference: This chapter describes each AmigaDOS command in detail.

Chapter 7. Workbench-Related Command Reference: This chapter describes the Workbench-related commands usable from AmigaDOS.

Chapter 8. Command Examples: This chapter provides examples of how to perform common tasks with AmigaDOS commands.

Appendix A. Error Messages: This chapter contains a list of possible program problems and suggested solutions.

Appendix B. Additional Amiga Directories: This chapter describes S:, DEVS:, L:, FONTS:, and other directories.

Appendix C. Using Floppy-Only Systems: This chapter tells you how to make the most of your system if you only have one floppy drive and no hard drive.

Appendix D. Advanced AmigaDOS Features: This chapter provides information on customizing AmigaDOS for advanced Amiga users.

A Glossary and an Index follow Appendix D.

Documentation Conventions

The following conventions are used in this manual:

COMMANDS, ASSIGNS, DEVICES, and NAMES	Commands, their keywords, device names, and assigned directories are displayed in all upper case letters. File and directory names are displayed in initial caps. However, they do not need to be entered this way. The Amiga ignores case differences in commands and arguments.
<n>	Angle brackets enclose variable information that you must supply. In place of <n>, substitute the value, text, or option desired. Do not enter the angle brackets when entering the variable.
Courier	Text appearing in the Courier font represents information that you type in or text displayed in a window in response to a command.
Key1 + Key2	Key combinations displayed with a + (plus) sign connecting them indicate pressing the keys simultaneously. For example, Ctrl+C indicates that you hold down the Ctrl key and, while holding it down, press C.
Key1, Key2	Key combinations displayed with a comma separating them indicate pressing the keys in sequence. For example, Esc,O indicates that you press and release the Esc key, followed by the O key.

Return	Directions to press the Return key indicate that you press the large odd shaped key on the right side of the keyboard above the right shift key.
Enter	Directions to "enter" something indicate that you type in the indicated information and press Return.
command line indentation	On command lines that are long enough to wrap to the next line, this manual shows the wrapped lines as indented for documentation purposes only. In practice, the wrapped lines align with the first character of the Shell prompt.

Related Documentation

AmigaDOS Quick Reference

Workbench User's Guide

ARexx User's Guide

In addition, the Amiga *ROM Kernel* manuals published by Addison-Wesley provide technical documentation of AmigaDOS for programmers and developers.

Chapter 1

Selecting an Interface

Although the Amiga comes with the Workbench graphical user interface (GUI) and most AmigaDOS operations can be run from the Workbench without opening a Shell window, there are several reasons to also learn how to use the AmigaDOS command line interface. Among the advantages of working directly with AmigaDOS are:

- **Personal preference**

Some users prefer working with text and keyboard rather than a mouse and icons. This may be a matter of personal taste or due to familiarity with some other text-based computer system.

- **Workbench limitations**

Although most basic operations can be accomplished with equal ease through the Shell or the Workbench, there are functions that can be done only with AmigaDOS commands. These include certain basic system configuration tasks and the running of scripts and utilities that do not have icons. Note that all AmigaDOS functions are available in Workbench using the Execute Command item in the Workbench menu.

- **Speed**

Users who can type reasonably well and are familiar with AmigaDOS commands often find that typing a command is faster than performing the equivalent operation with the mouse. This is particularly true when more than one command must be executed. Shell-based capabilities, such as pattern matching and redirection, make some tasks particularly easy when compared to Workbench methods. In addition, the text output of AmigaDOS commands usually displays faster than do requesters and windows full of icons.

- **Control and flexibility**

Running programs from the Shell makes it easier to control Amiga multitasking. Also, when using applications, such as software compilers that offer numerous run-time options, it is quicker to specify often-changed options on a command line than by editing the Tool Types of an icon.

- **Scripting**

Performing complex, repetitive, and/or unattended tasks is difficult, if not impossible, using a GUI. Such tasks are ideally suited to scripts, which are text files of AmigaDOS commands.

- **Saving resources for your applications**

The text interface requires less memory, disk space, and other system resources than the graphic imagery of a GUI.

Choosing Your Interface

Although some users prefer to use the Shell or the Workbench exclusively, most can make use of both once they learn the basics of AmigaDOS. Because Shell windows open on the Workbench screen, it is easy to switch back and forth between the two methods of working. Whether you do something through the Workbench or the Shell depends on the method that appears easiest to you for that particular task.

Workbench Users

Although you can work primarily in Workbench, we recommend that you become familiar with AmigaDOS because you may need to use AmigaDOS commands or examine a script to determine its function. The many convenience features of the Amiga Shell make the process of learning and using AmigaDOS considerably easier than most command line systems. If you prefer not to use AmigaDOS directly, you can attach scripts and Shell-only commands to icons.

Shell Users

For Shell users who choose not to open the Workbench, the Amiga's built-in GUI support is still an asset. Shell windows—like Workbench windows—can be quickly moved, sized, depth-adjusted, and opened or closed at will using the mouse. Shell windows can be opened on public screens other than the Workbench. FKey, MouseBlanker, and other Commodity utilities allow you to further customize your command line working environment.

AmigaDOS Tasks

When determining the most efficient way to interface with your Amiga, use the following table to identify where specific tasks are discussed. Then compare the AmigaDOS method to that available through Workbench. Select the method that is easiest and most comfortable for you to accomplish your goal.

Setting up your Preferences	Where to find it
Selecting a language	Chapter 7
Selecting a keyboard type and mouse options	Chapter 7
Selecting the default display mode	Chapter 7
Editing Workbench colors	Chapter 7
Setting the system time and date	Chapter 7
Selecting system fonts	Chapter 7
Selecting printer options	Chapter 7
Editing the mouse pointer	Chapter 7
Selecting Workbench background patterns	Chapter 7
Specifying the display beep sound type	Chapter 7

Setting up your disks and work environment	Where to find it
Formatting and copying disks	Chapter 7
Adding directories to the search path	Chapters 6, 8
Creating aliases	Chapters 6, 8 Appendix C
Assigning directories and devices	Chapters 6, 8 Appendix C
Using the ASSIGN PATH option	Chapter 6, Appendix C
Making commands resident	Chapter 6, Appendix C
Customizing startup files	Appendix D
Creating new directories	Chapter 6
Setting CrossDOS options	Chapter 7
Setting up function keys	Chapter 7
Preparing PCMCIA memory cards for use	Chapters 6, 7
Preloading resources into memory	Chapter 6 Appendix C
Making room on your Workbench disk	Appendix C
Using mount files and MountLists	Chapter 6 Appendix B

Learning AmigaDOS and the Shell	Where to find it
Deciding when to use AmigaDOS	Chapter 1
Opening and closing Shell windows	Chapters 2, 6, 8
Activating a Shell window	Chapter 2
Understanding the basics of a command	Chapter 6
Understanding the command line format	Chapter 6
Understanding the command template	Chapter 6
Understanding special characters in AmigaDOS	Chapter 3
Using pattern matching and wildcard characters	Chapter 3

Learning AmigaDOS and the Shell (cont'd)	Where to find it (cont'd)
---	--------------------------------------

Using command line editing	Chapter 2
Using command history	Chapter 2
Revealing previous command output	Chapters 2, 8
Using copy and paste	Chapter 2
Specifying paths	Chapters 3, 8
Working with a single Shell	Chapter 8
Understanding disk-based and internal commands	Chapter 3
Customizing the Shell window	Chapters 6, 8 Appendix D
Knowing the standard directory structure	Appendix B
Naming and renaming disks, files, and directories	Chapters 3, 6

Displaying information	Where to find it
-------------------------------	-------------------------

Determining the current directory	Chapters 3, 8
Listing the contents of a directory (example)	Chapters 6, 8
Displaying a command's template	Chapter 3
Listing information about files and directories	Chapters 6, 8
Using LIST LFORMAT (example)	Chapter 6
Displaying or setting the date and time	Chapter 7
Displaying graphics, text, and animation files	Chapters 6, 7
Using an on-screen calculator	Chapter 7
Using an on-screen clock	Chapter 7
Displaying the keyboard keymap	Chapter 7
Displaying software version numbers	Chapter 6
Changing the Shell window font	Chapter 6
Getting information about file systems	Chapter 6
Using escape sequences	Appendix D
Listing information about Shell processes	Chapter 6

Running programs **Where to find it**

Running programs from the Shell	Chapters 3, 8
Using the correct path	Chapters 3, 8
Changing the current directory	Chapter 8
Redirecting command input and output	Chapter 3
Stopping a program (example)	Chapter 8
Executing commands as background processes	Chapters 6, 8
Using ICONX (example)	Chapter 8

Using scripts **Where to find it**

Running scripts	Chapters 5, 6
Exiting from a script	Chapter 6
Editing text or scripts	Chapter 4
Creating and modifying a User-startup file	Chapter 8 Appendix D
Using scripting characters	Chapter 5
Setting the s protection bit	Chapters 5, 8
Creating scripts automatically	Chapters 5, 6, 8
Using PCD	Chapter 8 Appendix B
Preventing screen output with >NIL:	Chapter 8
Using script commands	Chapter 5
Debugging scripts	Chapter 5
Using environment variables	Chapter 5
Creating loops using EVAL (example)	Chapter 8
Creating a Move command	Chapter 8

Text editors**Where to find it**

Using the ED editor	Chapter 4
Accessing expanded ED menus	Chapter 8
Using the MEmacs editor	Chapter 4
Using the EDIT editor	Chapter 4

Manipulating files**Where to find it**

Copying files or directories	Chapters 6, 8
Copying disks	Chapter 7
Deleting files or directories	Chapters 6, 8
Deleting files with icons	Chapter 8
Deleting with interactive DIR	Chapter 8
Sorting and joining files	Chapter 8
Testing commands	Chapter 8
Using the Ram Disk	Appendix C
Removing unused fonts and libraries from memory	Chapter 8
Using pipes (example)	Chapter 8, Appendix B
Evaluating simple expressions	Chapter 6
Locating specified text strings	Chapter 6
Alphabetically sorting the lines of a file	Chapter 6
Redirecting printer output	Chapter 7
Updating .font files	Chapter 7

Workbench-specific tasks	Where to find it
Specifying Workbench parameters	Chapter 7
Editing icons	Chapter 7
Attaching icons to files	Chapter 8
Creating a custom Ram Disk icon	Chapter 8
Using ICONX	Chapter 8
Using an on-screen calculator	Chapter 7
Using an on-screen clock	Chapter 7
Printing a screen	Chapter 7
Creating Workbench background patterns	Chapter 7
Blanking the monitor screen	Chapter 7
Blanking the mouse pointer	Chapter 7
Changing Workbench colors	Chapter 7
Changing the mouse pointer	Chapter 7
Setting input options	Chapter 7
Specifying fonts	Chapter 7
Selecting display modes	Chapter 7
Starting Workbench from the Shell	Chapter 6

Chapter 2

Understanding the AmigaDOS Shell

An AmigaDOS Shell is a special window on the Workbench screen that accepts text input, allowing you to communicate with AmigaDOS. The Shell is a type of Command Line Interface or CLI. This chapter describes the following:

- About the Shell
- Opening and closing Shell windows
- Using the Shell

About the Shell

You can communicate directly with AmigaDOS through a Shell console window, a text-only interface that accepts input entered from the keyboard. The Shell window looks and acts like a Workbench window with these exceptions:

- Icons cannot be dragged into the Shell window.
- The mouse can only be used for copy and paste operations, except within the ED and MEmacs text editors.
- Scroll gadgets do not appear.
- The AmigaDOS Shell window uses only a non-proportional font, normally the System Default Text font (Topaz or Courier) specified by the Font Preferences editor.
- Any Workbench background patterns set in WBPatten do not appear in Shell windows.

Figure 2-1 illustrates a Shell window opened on the Workbench screen.

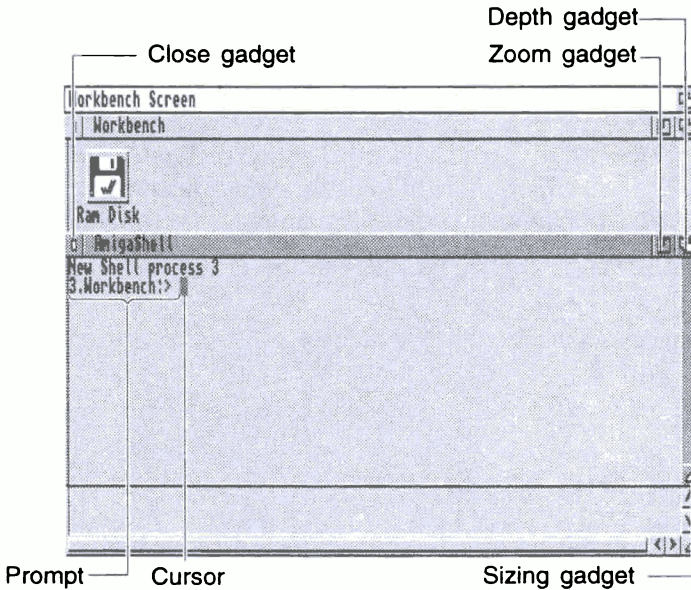


Figure 2-1. Shell Window

Like Workbench, several independent Shell windows can be open at the same time. While commands entered in one Shell are being executed, you can enter and execute different commands in another Shell window.

Opening Shell Windows

Shell windows can be opened in one of two ways:

- Click on the Shell icon in the Workbench System drawer.
- Use the NEWSHELL command described in Chapter 6.

When a Shell window is opened:

- The window is highlighted, indicating that it is the current window
- A prompt appears, such as **1.SYS:>**
- To the right of the prompt is a cursor, a small highlighted rectangle

Like Workbench, only the currently selected window can receive input. To enter information in a different window, click in it to make it the current window. While a Shell window is the current window, no menus are available in the Workbench title bar.

Closing Shell Windows

Use one of the following three ways to close a Shell window:

- Select the close gadget
- Enter the ENDSHELL command
- Press Ctrl+\

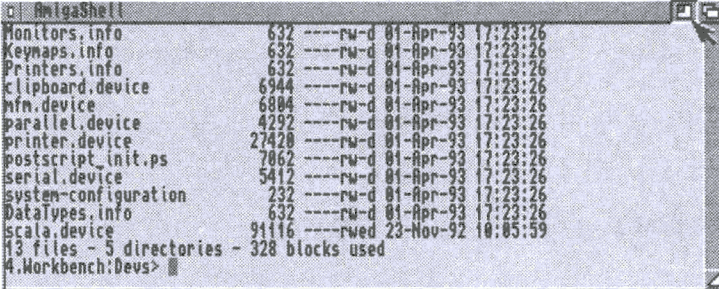
We recommend closing Shell windows when you are finished with them. Any open window uses memory.

All non-detached programs that run from a Shell must be finished before you can close the window. You can tell that a program is still active if pressing Return does not produce a Shell prompt in the window. Although you can still enter commands into such a window, AmigaDOS does not respond to the commands until the running program is exited.

Using the Shell

Enter AmigaDOS commands at the Shell's text prompt. Include with the command any necessary information, such as file names or command options. Press Return at the end of each command line to execute the command. The Shell prompt reappears when the command is finished executing.

To see command output that has scrolled out of the Shell window, enlarge the window by selecting the Shell zoom gadget or using the sizing gadget. This reveals as much of the previous contents of the window as fits. Figure 2-2 illustrates a Shell window before and after using the zoom gadget to display the entire output of a LIST command.

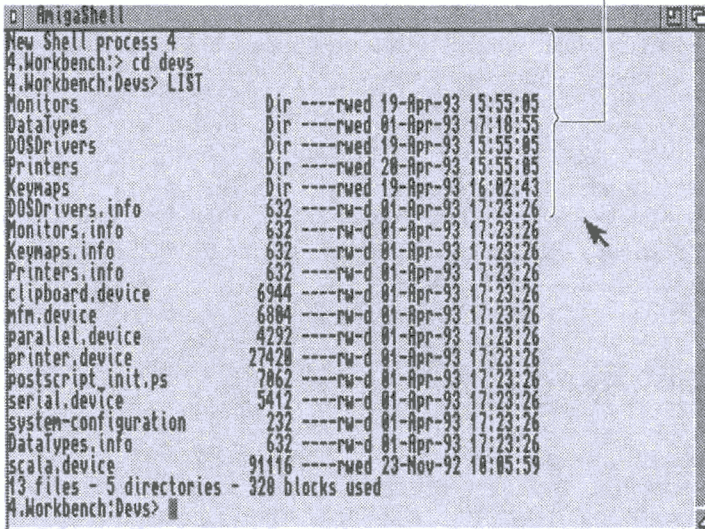


```

C:\ AmigaShell
Monitors.info          632 ----rw-d 01-Apr-93 17:23:26
Keymaps.info          632 ----rw-d 01-Apr-93 17:23:26
Printers.info         632 ----rw-d 01-Apr-93 17:23:26
clipboard.device     6944 ----rw-d 01-Apr-93 17:23:26
nfn.device           6884 ----rw-d 01-Apr-93 17:23:26
parallel.device      4292 ----rw-d 01-Apr-93 17:23:26
printer.device       27420 ----rw-d 01-Apr-93 17:23:26
postscript_init.ps   7862 ----rw-d 01-Apr-93 17:23:26
serial.device        5412 ----rw-d 01-Apr-93 17:23:26
system-configuration 232 ----rw-d 01-Apr-93 17:23:26
DataTypes.info       632 ----rw-d 01-Apr-93 17:23:26
scala.device         91116 ----rwed 23-Nov-92 10:05:59
13 files - 5 directories - 328 blocks used
A:\Workbench:Devs>

```

Previous Output Revealed



```

C:\ AmigaShell
New Shell process 4
A:\Workbench:> cd devs
A:\Workbench:Devs> LIST
Monitors          Dir ----rwed 19-Apr-93 15:55:05
DataTypes         Dir ----rwed 01-Apr-93 17:10:55
DOSDrivers        Dir ----rwed 19-Apr-93 15:55:05
Printers          Dir ----rwed 20-Apr-93 15:55:05
Keymaps           Dir ----rwed 19-Apr-93 16:02:43
DOSDrivers.info   632 ----rw-d 01-Apr-93 17:23:26
Monitors.info     632 ----rw-d 01-Apr-93 17:23:26
Keymaps.info      632 ----rw-d 01-Apr-93 17:23:26
Printers.info     632 ----rw-d 01-Apr-93 17:23:26
clipboard.device  6944 ----rw-d 01-Apr-93 17:23:26
nfn.device        6884 ----rw-d 01-Apr-93 17:23:26
parallel.device   4292 ----rw-d 01-Apr-93 17:23:26
printer.device    27420 ----rw-d 01-Apr-93 17:23:26
postscript_init.ps 7862 ----rw-d 01-Apr-93 17:23:26
serial.device     5412 ----rw-d 01-Apr-93 17:23:26
system-configuration 232 ----rw-d 01-Apr-93 17:23:26
DataTypes.info    632 ----rw-d 01-Apr-93 17:23:26
scala.device      91116 ----rwed 23-Nov-92 10:05:59
13 files - 5 directories - 328 blocks used
A:\Workbench:Devs>

```

Figure 2-2. Revealing Previous Output with the Zoom Gadget

Command Line Editing and Control

To simplify entering and editing command line text, the AmigaDOS Shell provides the following editing key and key combination options:

left arrow	Moves cursor one character to the left.
right arrow	Moves cursor one character to the right.
Shift+left arrow	Moves cursor to the beginning of the line.
Shift+right arrow	Moves cursor to the end of the line.
Backspace	Deletes the character to the left of the cursor.
Del	Deletes the character highlighted by the cursor.
Ctrl+H	Deletes the last character (same as Backspace).
Ctrl+M	Processes the command line (same as Return).
Ctrl+J	Adds a line feed.
Ctrl+W	Deletes the word to the left of the cursor.
Ctrl+X	Deletes the current line.
Ctrl+K	Deletes everything from the cursor forward to the end of the line.
Ctrl+Y	Replaces the characters deleted with Ctrl+K.
Ctrl+U	Deletes everything from the cursor backward to the start of the line.

In addition, the Shell supports the following keys and key combinations:

Space bar (or any printable character)	Suspends output (stops scrolling).
Backspace	Resumes output (continues scrolling).
Ctrl+C	Sends a BREAK command to the current process (halts the process).
Ctrl+D	Sends a BREAK command to the current script (halts the script).
Ctrl+F	Activates and brings Workbench program windows to the front.

Ctrl+S	Suspends output.
Ctrl+Q	Resumes output if it was suspended with Ctrl+S.
Ctrl+\	Closes the Shell window. When console I/O is redirected to another device with * restores normal I/O.

The Shell allows you to enter a command or other information while listing output. However, this stops the output until you press the Return key. The new command executes after the output is finished listing.

If you enter a new command or text and then choose to delete it, the original output resumes scrolling as soon as the last character is erased.

Using the Command History

The Shell uses a 2 KB command line buffer to retain command lines, which provides a command history. Using this history you can recall previously entered command lines, edit them, and re-execute them. This lets you easily repeat a command or enter several similar commands. Figure 2-3 illustrates a series of commands stored in the command history buffer.

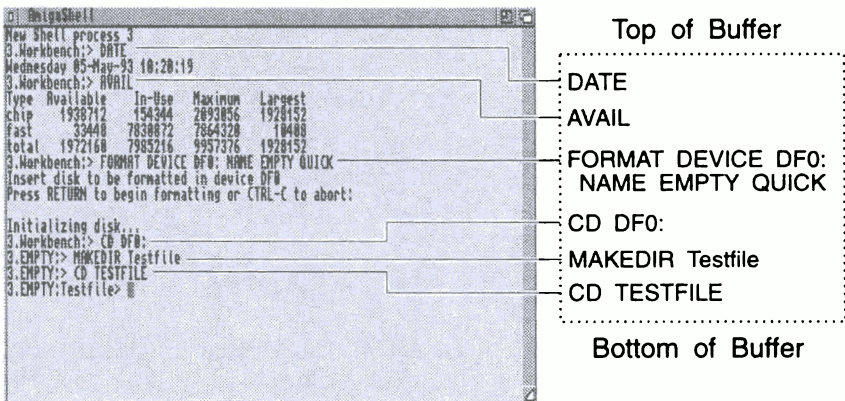


Figure 2-3. Command History Buffer

The exact number of lines retained in the command line buffer varies depending on the length of the lines actually stored. When the buffer is full, the oldest lines are removed. You can access lines in the buffer with the up and down arrow keys:

- up arrow** Moves backward in the history buffer (earlier lines).
- down arrow** Moves forward in the history buffer (later lines).

For example, you can copy several .info files from one directory to another by entering the full command line with the complete path only once and then recalling the line as many times as necessary, changing only the file name.

You can also search for the most recent occurrence of a specific command by entering the command line, or the beginning of it, and pressing Shift+up arrow (or Ctrl+R). For example, if you enter **DIR** and press Shift+up arrow, you are returned to the last command entered to perform a DIR of any directory. Pressing Shift+down arrow goes to the bottom of the command history buffer, leaving the cursor on a blank line.

Copying and Pasting

You can copy and paste information from one console window, such as a Shell or ED window, to the same or another window. This is the only Workbench-style mouse operation performed in Shell windows, except for within the ED and MEmacs text editors. Figure 2-4 illustrates copying and pasting from the Shell window to the ED window.

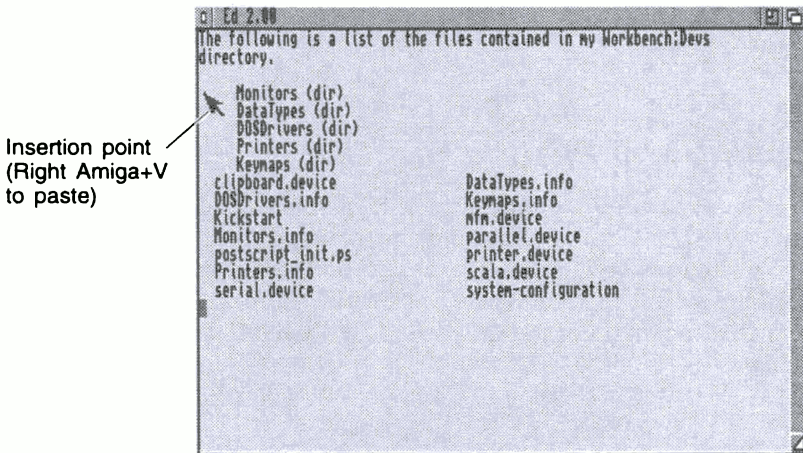
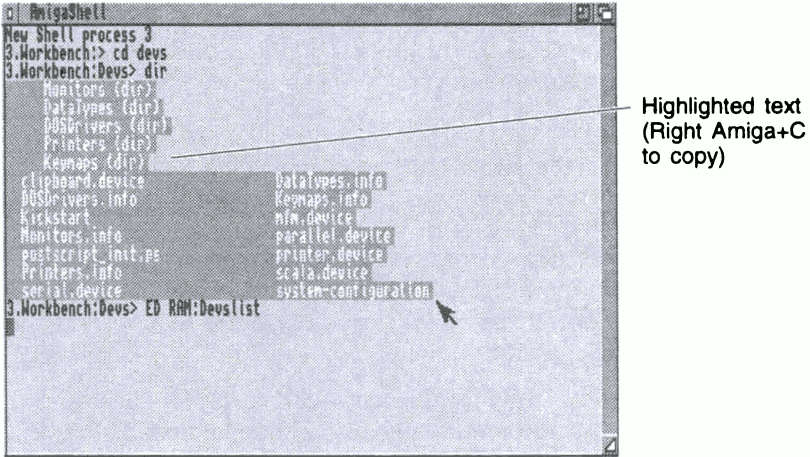


Figure 2-4. Copy and Paste

Use the mouse to highlight the area of text to be copied and pasted. Highlight the text to be copied by moving the pointer to the beginning of the text area, holding down the selection button, and dragging the mouse pointer to the end of the desired text. Release the selection button and press Right Amiga+C. The highlighted area is copied into the Clipboard and the area is unhighlighted. The text you copied can

be repeatedly pasted into any application window that supports reading text from the Clipboard, such as the Shell, ED, and MEMacs.

To position the cursor where you want to paste the text, move the mouse pointer to that location and click. Press right Amiga+V to paste the text.

Note If a block of text is pasted into a Shell window, the Shell attempts to execute each line of the text as a command. This can have unpredictable results if the block of text has embedded returns and is not an AmigaDOS script.

Working with the Shell

The following are tips for speeding your work with the Shell.

- **Use command history and command line editing**

It sometimes takes several attempts using the same command before getting it right, especially when you are first learning how to use AmigaDOS. Use the arrow keys to recall a previous command and change only the part of the line that causes the problem to eliminate the need to retype the entire line.

- **Use aliases**

Defining short aliases for commands you use often is another time-saver. It also eliminates the need to remember a long and/or complex series of options. For complete instructions, see the ALIAS command in Chapter 6.

- **Omit unnecessary keywords**

For clarity, AmigaDOS command names and keywords throughout this book are often shown although they are optional. When you learn a command's format, however, you seldom need to include optional keywords.

- **Do not use capital letters**

Command names, keywords, and assigned directories are shown in all upper case letters throughout this manual even though AmigaDOS is case-indifferent. This is done to distinguish the keywords from the file names and other information on the example command line. There is no need to use capitalization, except in commands that create a file or directory whose name you want to appear capitalized.

- **Use implied CD**

This allows you to leave out the CD command, saving three keystrokes. Enter the only directory name, path, colon, or slashes at the prompt to change directories. For more information about changing directories, see the CD command in Chapter 6.

Chapter 3

Working With AmigaDOS

AmigaDOS stores information in the same hierarchical structure as Workbench. AmigaDOS commands have specific rules that you must follow when creating scripts and programs to run on your Amiga. You must be familiar with the terms specific to the file system and with AmigaDOS command concepts to successfully use AmigaDOS. This chapter describes the following:

- Managing files, directories, and disks
- Command line basics
- Types of commands
- Command structure
- Special characters
- Running programs
- Refining your AmigaDOS environment

Specific commands are fully described in Chapters 6 and 7 of this manual.

Managing Files, Directories, and Disks

In order to use AmigaDOS to access information, you must know where that information is located. On an Amiga, all information is stored in a system of directories and files. This is the same system used by the Workbench, only the method of working with it is different. Most notably, you do not use icons to manipulate the files and directories. See the *Workbench User's Guide* for detailed information about the Amiga file system and the use of common

commands. Use this section to review the following AmigaDOS basic concepts:

- File system terms
- File management
- Naming conventions
- Keywords

File System Terms

The following are the main elements of the AmigaDOS file system:

Device	A physical device, such as a disk drive or printer, or a software (logical) device, such as RAM: or the printer device PRT:.
Partition	A hard disk or part of a hard disk that AmigaDOS treats as a separate device.
Volume	A particular disk or subdivision of a hard disk that AmigaDOS treats as a separate device. Floppy disks and hard disk partitions are volumes.
Directory	Equivalent to a drawer in Workbench.
Root Directory	The top of the filing system for a given volume; the directory that contains all other directories.
Subdirectory	A directory that is contained within another directory.
File	A named collection of data.
Path	The series of device, directory, and subdirectory names that uniquely specifies a particular file and its location.

File Management

AmigaDOS stores information on a device in a file system, which is an organization of directories, subdirectories, and files. Directories and files are arranged in a hierarchical system often referred to as a tree. The branches are directories, which can include subdirectories. At the ends of the branches are the files, unless the directory is empty. Figure 3-1 illustrates a directory tree.

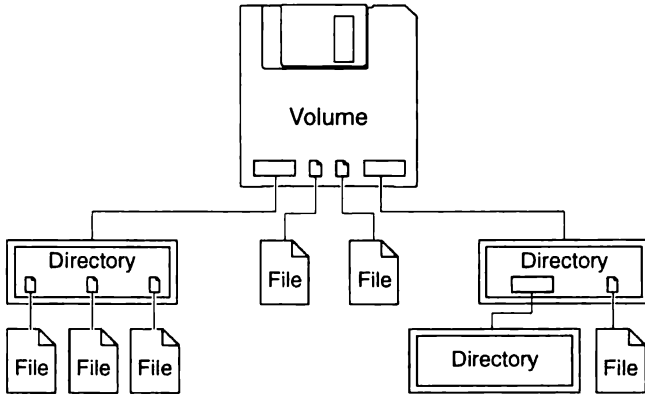


Figure 3-1. Example Directory Tree

Devices

Devices include logical devices and the hardware associated with your Amiga, such as floppy disk drives, hard disk drives, the Ram Disk, RAD:, and peripheral devices. Information stored on these devices can be accessed using a variety of names.

To access files on a particular volume, you can refer to the volume by its volume name, such as `Workbench:`, or its device name, such as `DF0:`. Use the names interchangeably; however, you must always include the colon (`:`) after the name. When you refer to a disk by volume name, the system searches all the available drives for the volume. If it cannot find a volume of that name, a requester asks you to insert the volume. When you refer to the disk by a particular device name, the system uses whatever volume is inserted in that device.

AmigaDOS has standard names assigned to peripheral devices that are attached to the various ports, as well as to various logical (software) devices. Generally, these devices are used for output, such as copying a file to a printer. The standard device names are:

- SYS:** Represents the volume on which the Amiga looks for its basic disk-based resources, such as C: and LIBS:.
- PAR:** Represents any device, usually a printer, that is connected to the parallel port. If you copy a file to PAR:, it is sent to the device attached to the parallel port. Output directed here is not modified by any driver software.
- SER:** Represents any device connected to the serial port, such as a printer or a modem. Output directed here is not modified by any driver software.
- PRT:** Represents the printer. Output to PRT: goes through the selected printer driver and to the serial or parallel port as specified in the Printer editor in the Prefs drawer.
- CON:** Represents a console, which uses a window to accept typed input and display text output. The Shell window is one kind of console window.
- CONSOLE:** Represents the current console window. An asterisk (*) can also be used in place of CONSOLE:.
- NIL:** Represents a dummy device commonly used to prevent output from appearing on the screen. All output sent to NIL: is discarded.
- RAM:** Represents the Ram Disk, which is a portion of the Amiga's internal memory that can be used as a storage device. All information in RAM: is lost if the Amiga is rebooted or turned off.
- RAD:** Represents a special kind of Ram Disk that is only lost if the system is turned off, but not when rebooted. Refer to Appendix C for detailed information.
- DF0:** Represents the Amiga's main internal floppy drive from which the Amiga attempts to boot if there is no other bootable device available.

Directories

Directories are the AmigaDOS equivalent to drawers in Workbench. They allow you to group and classify related files. Each file on a disk is located in a directory. An empty, formatted disk contains one directory, the root directory. If you create a file on an empty disk, that file resides in the root directory. If the file has an icon attached to it, the icon appears in the disk window.

Directories can contain other directories, called subdirectories. The Amiga supports an arbitrary number of nested directories (directories within directories).

Files

A file, the basic unit of storage on a computer, is an organized collection of information. All the programs and any permanent data that a program uses or produces are files. Project icons represent data files. Data files contain the information created or used by a program, such as text, graphic, or spreadsheet files.

.Info Files

Another type of file used by the Amiga is a .info file (pronounced dot info file). The .info files contain the icons that appear on the Workbench screen. Every file or directory that has an icon also has a corresponding .info file. In addition to storing the graphics and position data for the icon image, a .info file contains any Default Tool or Tool Type information entered into the icon's Information window.

When working through the Shell, AmigaDOS does not automatically associate .info files with the corresponding files or directories. For example, if you use the COPY command to copy the Clock file from the Utilities directory to the System directory, the Clock.info file is not copied with it like it is when you drag the Clock icon from one drawer to another in Workbench. In AmigaDOS, to be sure the Clock icon appears in the System drawer, you must also copy the Clock.info file.

When you change icon images by copying .info files, you need to copy an icon of the same type as the item it represents: Tool, Project, Drawer, Disk, or Trashcan. If the icon's type does not match the type

of file it represents, it may not open from the Workbench. Icon type is displayed in the icon's Information window and can be changed with the IconEdit program.

Each disk icon has a corresponding disk.info file. If you delete the disk.info file, a default disk icon automatically replaces the previous icon.

Naming Conventions

The following naming conventions apply to file and directory names:

- Names can be up to 30 characters long and can contain upper case letters and any punctuation marks that are not reserved. Workbench file and drawer names can only be up to 25 characters long to accommodate a possible .info extension.
- Colons (:) and slashes (/) are reserved and cannot be used in file or directory names. Semicolons (;), asterisks (*), parentheses (()), question marks (?), back apostrophes (`), number or pound signs (#), square brackets ([]), angle brackets (< >), tildes (~), vertical bars (|), dollar signs (\$), double quotation marks ("), and percent signs (%) are not reserved; however, we recommend that you do not use these characters in your file or directory names because they have special meaning in AmigaDOS.
- Capitalization used in file names is preserved even though AmigaDOS is not case-sensitive. The name is recognized by the characters; for example, TextFile is treated the same as textfile.
- Spaces in names are allowed, but not recommended when working through AmigaDOS. If you do use names with spaces, the entire path containing the name must be enclosed in double quotation marks. We recommend using an underscore (_) as a separator rather than a space.

Note If you use spaces in file names, do not place one at the beginning or end of the name. This space is invisible when displayed and easily overlooked as part of the file name. AmigaDOS does not recognize the name if such a space is not entered.

Note If you use spaces in file names, do not place one at the beginning or end of the name. This space is invisible when displayed and easily overlooked as part of the file name. AmigaDOS does not recognize the name if such a space is not entered.

Keywords

A keyword is a special word recognized by an AmigaDOS command. AmigaDOS commands use keywords to identify arguments or to specify options. If there is a conflict between a name and a command keyword, enclosing the name in quotation marks ensures that it is interpreted as a name. For example, if you have a directory named Files and you want to display information about all of its files and subdirectories, you might use the command **LIST Files**. However, this is ambiguous because LIST has the keyword FILES. To avoid this, enter:

```
LIST "Files"
```

Command Line Basics

Effectively using a Command Line Interface, such as the Amiga Shell, requires that you understand concepts unique to this method of working with your computer. These include:

- The distinctions between files, programs, commands, and scripts
- The search path
- The current directory

Files, Programs, Commands, and Scripts

Files, programs, commands, and scripts are named collections of data that can be stored in the computer's memory or on a disk drive. These concepts can be confusing because the meanings of the terms often overlap.

Files

Programs, commands, and scripts are all files. Files can be stored on disk or in the Amiga's memory, although certain kinds of files are customarily stored in specific locations.

Programs

A program is a file that the computer executes to accomplish some task. Software that you buy for the Amiga are mostly programs. Workbench programs are called tools, utilities, or editors. A file that is not a program is typically a data file, which contains information a program can use, such as text or graphics. Programs can be stored anywhere.

Commands

A command is a type of program. The term command usually refers to programs that are executed through a command line such as the Shell, especially those programs that come with a computer as part of the operating system and perform some basic function. The programs detailed in Chapter 6 of this book are the AmigaDOS commands. AmigaDOS commands that are not Internal (built into the Shell) are stored in the C: directory.

The term command can also refer to a specific instance of that program's invocation, including its arguments, if any. In this manual, the term command line is used to indicate a command program's invocation; for example, "The command line **TYPE S:User-startup** is an example of the TYPE command." The command must always be the first thing on the command line.

Scripts

A script is another type of program that is a text file containing a series of commands comprising the program. You can view and edit a script with a text editor. A script typically performs some simple task that can be modified by editing the script.

In this manual, the term script refers to scripts of AmigaDOS commands. AmigaDOS scripts are customarily stored in the S:

directory. ARexx programs are also called scripts, although they can be referred to as macros or programs; these scripts are also stored in the S: directory using the assignment REXX:. Some computer systems refer to scripts as batch files.

Search Path

When using the Shell, the Amiga must know where to look for the commands you want to use. The Shell has a search path, which allows you to enter commands without providing the full path. The search path is a series of directories that AmigaDOS searches to find commands that are entered without paths.

The default search path includes the current directory, C:, and several other directories specified in the standard Startup-sequence. You can add other directories in which you keep frequently-used programs by using the PATH command or by using multiple assignments with the ASSIGN command. There is, however, a significant difference between these two methods. Directories added to the search path with the PATH command are local to the Shell in which you added them and to any sub-Shells launched from that Shell. Directories added through multiple assignments with the ASSIGN command are global to the whole system.

When you enter something in the Shell, AmigaDOS looks through the directories in the search path for a command of that name. It searches the directories in the order they appear in the path until it either finds the command or reaches the end of the path list. When a command cannot be found in any of the search path directories, the Shell displays an **Unknown command** message, as illustrated in Figure 3-2.

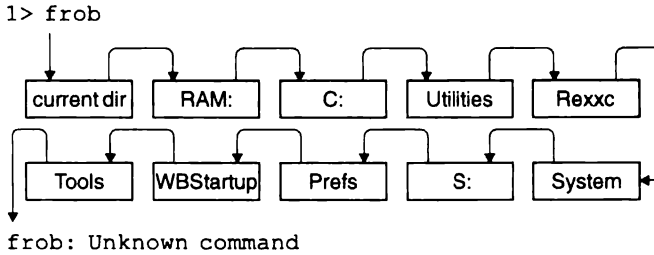


Figure 3-2. Search Path

Note AmigaDOS uses the search path to find commands only. You must include a full path to any files referred to in a command's arguments.

Current Directory

The current directory is the Shell's current location in the filing system hierarchy, similar to the Workbench's current window. The name of the current directory is displayed in the default Shell prompt so that you always know where you are. The following properties apply to the current directory:

- Each Shell has its own independent current directory.
- A Shell has only one current directory.
- The current directory is always the first directory in the search path.
- The path up to and including the current directory is assumed and does not need be included in a path to a particular command that is within the current directory.
- The current directory is the default directory, the directory on which a command operates, if no other directory is specified.

Changing the current directory, like adding directories to the search path, is a way to reduce the amount of typing necessary to specify a command. Often you need to perform several operations within a certain directory, such as copying, renaming, and deleting files. You

can avoid entering the full path for each file by changing the current directory to that directory in which most of the files are located.

Types of Commands

AmigaDOS has both disk-based and internal commands.

Disk-based commands must be loaded from a disk before execution. On systems with hard disks, the disk-based commands are always accessible to the system since they are automatically loaded when invoked. On a floppy-only system, these commands are read from a floppy disk that must be inserted whenever they are called.

Internal commands reside in the Shell, which is in ROM (Read Only Memory). The system accesses internal commands immediately.

Some AmigaDOS commands are essentially the same as menu items or programs on the Workbench. These commands and the corresponding Workbench equivalent are shown in the following table:

Command	Function	Workbench Counterpart
CD	Change the current directory	Select another window/icon
COPY	Copy a file, directory, or disk	Copy menu item
DATE	Set the correct date and time	Prefs/Time editor
DELETE	Delete a file or directory	Delete menu item
DIR	Show files in a directory	Show All Files menu item
DISKCOPY	Copy a disk	Copy menu item
ENDSHELL	Close a Shell window	Select Shell window close gadget
FORMAT	Format a disk	Format Disk menu item
INFO	Show information on all disks	Observe disk window title bars

Command (cont'd)	Function (cont'd)	Workbench Counterpart (cont'd)
LIST	Show files, with sizes, etc.	View By Name menu item
MAKEDIR	Make a new directory	New Drawer menu item
NEWSHELL	Open a new Shell window	Open Shell icon
RELABEL	Rename the disk volume in the specified drive with the specified name	Rename menu item
RENAME	Rename a file or directory	Rename menu item
SETCLOCK	Save the date and time	Prefs/Time editor
TYPE	Display the contents of a text file	MultiView program

AmigaDOS Command Structure

Every AmigaDOS command has a specific format and syntax that must be used for the system to accept and act on the command. The general rules for working with AmigaDOS commands are few, but absolute:

- A legal command or program name must appear first on the command line. The full path to the command is not necessary if the command is in a directory on the search path.
- Arguments are separated from the command and from each other by spaces; a single space is sufficient, but additional spaces are allowed. No punctuation, other than that specifically needed in the command, should be used.
- AmigaDOS is not case-sensitive. Any mixture of upper and lower case can be used on the command line, however, case is ignored. Capitalization given in file and directory names is preserved.

- Except where noted, when a path or string argument contains a space, the entire path or string must be enclosed in double quotation marks ("). For example:


```
1> ECHO comment TO Adisk:Text/Comment
1> ECHO "A comment" TO "My Disk:Text/Comment"
```
- The maximum length of a standard Shell command line is 512 characters.

A sample of the structure of an AmigaDOS command line is illustrated in Figure 3-3. It consists of the COPY command followed by two arguments.

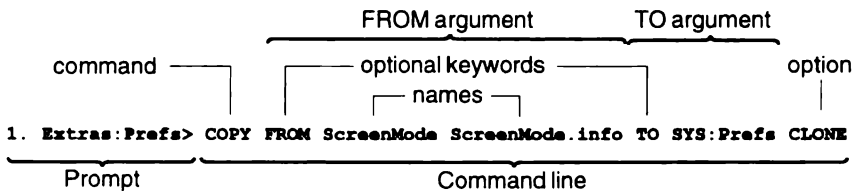


Figure 3-3. Example Command Line

An argument is an additional piece of information the command uses, such as a file name or option. Command arguments are similar to icon Tool Types on the Workbench. Depending on the command, arguments can be optional or required. Figure 3-3 illustrates the following points about arguments.

- Keywords for AmigaDOS commands are generally full words or simple abbreviations, in this example, FROM and TO.
- An argument can consist of more than one term; in the example, a file name argument accepts two names at the same time: Screenmode and Screenmode.info.
- Some arguments have an identifying keyword, which can be optional or required.
- When optional keywords are omitted in a multiple-argument command, the arguments must appear in the order shown by the command template.

Special AmigaDOS Characters

There are several characters that have special meanings when used in AmigaDOS. The functions of special characters include the following:

- Specifying paths
- Pattern matching
- Redirecting command input and output

It is important when using AmigaDOS to remember the various special functions that characters can have. Note that in different contexts the same character can have different effects or have no special effect. If a command that appears correct produces unexpected results, check to see if any character in the command has a special function.

Command Line Characters

The colon and slash characters are reserved by AmigaDOS for specifying paths. In file requesters, on the command line, or in scripts these characters are used only to separate components on the path line.

Colon (:)

Colons are used to designate device names (DF0:), volume names (Workbench:), and assigned directories (SYS:). There are no spaces before the colon, unless it is the first character in the path, or between the colon and subsequent file or directory names in a path. Used by itself, the colon represents the root directory of the current volume. The following are examples of legal uses of the colon:

```
1> DIR DEVS:
1> DIR DF0:Pictures
1> DIR :Prefs
1> DIR :
```


Slash (/)

Slashes are used within paths to separate directories and file names. For example:

```
1> LIST Reports/Salesreps/Eastern
```

The three directory levels are separated by slashes; this example lists the Eastern subdirectory.

Entering a single slash moves the current directory structure up one level. For example, if Reports/Salesreps/Eastern is the current path, entering the following:

```
1> CD /
```

changes the directory path to Reports/Salesreps. Using two slashes moves up two levels, and so on.

Double quotation mark (")

The double quotation mark by itself does not have any special meaning. However, you may occasionally need to use double quotation marks around a command argument for the command to work properly. Since AmigaDOS uses spaces as an argument separator, you must enclose an argument (such as path) that contains spaces in double quotation marks to keep the Shell from interpreting the parts of the argument as separate arguments. For example, the following is incorrect:

```
1> COPY Ram Disk:foo TO SYS:Otherdir
```

It produces an error message because there is a space in the path. The command indicates that there are two items to be copied, when only one is intended. Enclosing the path in quotation marks forces it to be treated as a single argument:

```
1> COPY "Ram Disk:foo" TO SYS:Otherdir
```

Using double quotation marks with nothing between them is a quick way to reference the current directory. For example:

```
1> COPY DF0:public.library TO ""
```

If the current directory is LIBS:, the public.library file is copied there.

Plus (+)

The plus sign, when entered with the RUN command, concatenates several commands entered on subsequent lines into a single command line. For further information and examples, see the RUN command in Chapter 6.

Question mark (?)

One of the special uses of the question mark is to display a command's template. The template is an online reminder of the command's arguments. To display a specific command's template, enter the command name followed by a space and then the question mark, with no other arguments:

```
1> TYPE ?  
FROM/A/M, TO/K, OPT/K, HEX/S, NUMBER/S :
```

The Shell displays the template. It also allows you to enter the arguments for the command with which you have used the question mark. Enter arguments for the command after the colon. Be sure to enter only the arguments and keywords needed before pressing Return.

Pattern Matching

You can work on several files or directories with one command using pattern matching. Special wildcard characters are used in command arguments to match characters in the file names. For example, use a wildcard character in a single command for copying or renaming all the files beginning with a specific letter, ending with the same extension, or residing in the same directory.

Wildcard Characters

The following list shows each wildcard character and the type of match it makes. In the list, a <p> indicates that either a single or multiple character string immediately adjacent to the wildcard is matched. To match a literal wildcard character, you must escape its wildcard meaning by prefacing it with an apostrophe ('). For example, '?', matches ?, and '' (two single apostrophes) matches '.

?	Matches any single character.
#<p>	Matches zero or more occurrences of <p>.
<p1> <p2>	Matches if either <p1> or <p2> matches.
~<p>	Matches everything but <p>.
(<p1><p2>...)	Parentheses group items together.
[<p>-<p>]	Square brackets delimit a character range.
%	Matches the null string (no characters).
'<p>	When <p> is wildcard character, matches that character.

The following examples indicate the matches that can be made using the entry in the left column.

A?B	Matches any three character names beginning with A and ending with B, such as AcB, AzB, and a3b.
A#BC	Matches any name beginning with A, ending with C, and having any number of Bs in between, such as AC, ABC, ABBC, ABBBC.
ABC#?	Matches any name beginning with ABC, regardless of what follows, such as ABCD, ABCDEF.info, or ABCXYZ.
#?XYZ	Matches any name ending in XYZ, regardless of what precedes it, such as ABCXYZ and ABCDEFXYZ.
A(BIC)D	Matches ABD or ACD.
~(XYZ)	Matches anything but XYZ.
~(#?XYZ)	Matches anything not ending in XYZ.
A#(BC)	Matches any name beginning with A followed by any number of BC combinations, such as ABC, ABCBC, and ABCBCBC.
A(BIDI%)#C	Matches ABC, ADC, AC (% is the null string), ABCC, ADCC, ACCC, and so forth.
[A-D]#?	Matches any name beginning with A, B, C, or D.
#?XYZ'?	Matches any name ending with XYZ?

The combination of `#?` matches any characters and is used most often. `#?` is equivalent to the `*` wildcard used by other computer systems. For example, to delete all the `.info` files in the `Picture` directory, enter:

```
1> DELETE Picture/#?.info
```

Caution **Be careful not to accidentally delete the contents of a disk when using `#?`.**

Redirection

Redirection can change input or output to a specific file or device (such as a printer, modem, or logical device). When working in the Shell, the keyboard is the source of command input and the current Shell window is the destination for output. You can redirect input and output using the left angle bracket, right angle bracket, and asterisk characters.

Angle Brackets

A redirection argument consists of either the `<` or `>` symbol followed by a file name or device name. The angle bracket must be preceded by a space, but no trailing space is necessary.

For some commands, the redirection characters can replace the keywords `TO` and `FROM`, depending on the command's syntax.

You can only redirect input or output on the command line in which the redirection characters appear. AmigaDOS applies the default input and output sources for any subsequent commands without redirection.

Right Angle Bracket (>)

The right angle bracket redirects the console output of a command to the destination pointed to by the bracket. The console output is the text that the command prints in the Shell window when executed. For example,

```
1> DIR >Testfile DF0:
```

sends a directory listing of DF0: to a file in the current directory called Testfile. Testfile is created if it does not already exist and it contains the directory listing as ASCII text. The directory listing is not displayed on the screen.

Only the console output of a command is redirected, not the data on which the command works. For example,

```
1> COPY >Log Picdir TO PicsArchive: ALL
```

copies all the files in the Picdir directory to the PicsArchive disk, sending a list of the copied file names to the Log file.

Left Angle Bracket (<)

To change the source of a command's input from the keyboard to a file, use the < symbol. However, a question mark (?) must also be used as a separate argument on the command line. The question mark instructs the command to accept input; it is not a wildcard character in this context. The following example creates a file and then uses the contents of the file as the argument for a command:

```
1> ECHO tomorrow TO Datefile
1> DATE ? <Datefile
```

The ECHO command creates a file called Datefile containing the word "tomorrow". The DATE command accepts the contents of Datefile (the word "tomorrow") as if it were entered at the keyboard. This sets the system date 24 hours ahead.

Double Right Angle Brackets (>>)

Redirect output and append material to an existing file using two output symbols (>>) with no spaces between them. For example:

```
1> Postscript >>Laser/Letter
```

executes the program Postscript, adding its output to the end of the Laser/Letter file.

Asterisk (*)

An asterisk refers to the current Shell window. However, to avoid confusion with other uses of the asterisk, we recommend using CONSOLE:, which is the synonym for *. The asterisk can be used as

a FROM or TO argument or as a redirection file name (the source of input or the output destination).

Pressing Ctrl+\ restores input/output to the default source. For example:

```
1> COPY * TO Screenfile
or
1> COPY CONSOLE: TO Screenfile
```

copies all subsequent text typed in the current window to the file called Screenfile until you press Ctrl+\.

Ctrl+\ is also used to close a Shell window. Be careful not to press this key combination twice when you want to end the redirection since it also closes the Shell window.

Running Programs

Most programs can be run from both the Workbench and the Shell. To run a program from the Shell, you usually enter the program name at the Shell prompt. (If the program file is not in the search path, you must specify the complete path to the file.) This tells AmigaDOS to load and execute the program.

Most programs allow you to specify additional information on the command line after the program name, such as the name of a file to load or startup options. These additional items are its arguments. Refer to a program's documentation to determine the arguments it allows and how they should be entered.

For example:

```
1> MEmacs
```

loads and runs the MEmacs editor. Adding an argument:

```
1> MEmacs S:User-startup
```

loads and runs MEmacs, automatically opening the User-startup file in the S: directory as the file to begin editing.

```
1> CLOCK WIDTH 200 HEIGHT 100 SECONDS
```

loads the Clock with a specified size of 200 pixels by 100 pixels and the SECONDS option turned on.

Often this argument-passing ability is provided as a convenience, allowing you to specify directly on the command line what might otherwise require several menu operations. However, many programs, especially those that can only be run from a Shell, require that file names or other arguments be specified on the command line with the program name.

Running Programs in the Background

Another way to enter a program name is with the RUN command. RUN loads and runs a program in the background. The Shell prompt returns after the program is opened.

For example, entering:

```
1> MEMacs
```

opens the MEMacs editor, but you cannot enter any additional commands or close the Shell window until you exit MEMacs.

However, entering:

```
1> RUN MEMacs
```

opens the MEMacs editor and returns the Shell prompt so that you can enter additional commands.

When a program is invoked with RUN, a process number is assigned to it and a message indicating the new process number is displayed, such as **[CLI 2]**.

Any output that the program generates appears in the originating Shell window.

You cannot close the Shell window if any programs launched from that window are still running. For example, if you open MEMacs through the Shell, you cannot close the Shell window until you exit MEMacs. Avoid this by using the NIL: device. See Chapter 8 for an example.

Refining Your AmigaDOS Environment

The following tips help you to set up your AmigaDOS environment to suit your particular needs.

- **Customize your Shell prompt**

Changing the color, for example, of the prompt string with escape codes makes the prompt easier to distinguish from the rest of the command line and the output that your commands produce. This helps you keep track of the process number and current directory that are normally part of the prompt. For more information on how to change your Shell prompt, see the PROMPT command in Chapter 6 and the example in Chapter 8.

- **Create a logical directory structure and use meaningful names**

Because gaining access to something requires knowing where it is, you should organize your disks and directories in a logical way, with names that reflect their contents. However, do not create directory structures that are heavily nested without good reason.

- **Avoid using spaces and other special characters in names**

Characters with special meaning in AmigaDOS, such as # and ~, are allowed in names, but can cause problems when used on the command line. Use a period (.), underscore (_), or capital letters instead of spaces to separate words in a name: "Anim.file," "Anim_file," or "AnimFile" rather than "Anim file."

- **Name related files consistently**

Giving related file names a common extension or sequential numbering simplifies using pattern matching when processing files.

- **Use assigned names in paths**

Assigned names allow you to type a short, easy-to-remember name rather than a long path. For example, it is quicker to type **ENVARC:** than **SYS:Prefs/Env-archive**. Make your own assigned names for directories you use often and for deeply nested directories.

- **Extend the search path**

If you have a variety of often-used commands or programs, adding their directories to the search path with **PATH** or **ASSIGN** makes accessing them easier.

- **Experiment**

The best way to learn how AmigaDOS works is to experiment. Provided that you use caution with potentially destructive commands, such as pattern-matching **DELETES**, you can experiment freely.

Chapter 4

Using the Editors

A text editor or word processing program is necessary for creating or editing text files and script files. Amiga Workbench software comes with three text editors. This chapter describes them in the following order:

- ED
- MEmacs
- EDIT

Each of the Amiga editors can be used separately for editing AmigaDOS scripts and programs; ED and MEmacs can be used for creating these files. If you are comfortable with the UNIX Emacs editor, you may prefer using the MEmacs editor. If you need to edit files containing binary code or you need to edit files too large to fit into memory, use EDIT. If you are not familiar with any of the editors, we recommend that you use the ED editor.

Each editor has the basic functionality of a word processor, however, none of these editors support style formatting options, such as italics, page numbering, or different fonts. If you need these features, you can purchase third party word processing software containing such features for your Amiga.

ED

ED is a full screen ASCII text editor that uses menus and function keys to access its features. It is easy to use and is suitable for editing scripts, startup files, MountLists, and other simple files. Use either a mouse or the keyboard to perform operations with ED. Although

ED's menus are preprogrammed, when you are familiar with the program, you can reconfigure them as needed.

Note ED does not accept files containing binary code. To edit this type of file, use EDIT or MEMacs.

The bottom line of the ED window is the status line used for displaying messages, prompts, and commands. Error messages displayed on the status line remain until you enter another ED command. Figure 4-1 illustrates the ED window showing the status line.

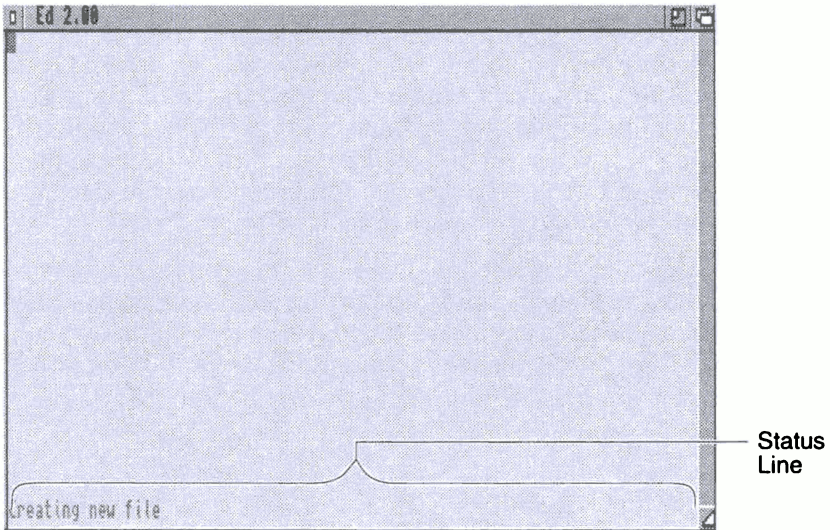


Figure 4-1. ED Window with Status Line

The format for ED is the following:

```
ED [FROM] <filename> [SIZE <n>] [WITH <filename>]
  [WINDOW <window specification>] [TABS <n>] [WIDTH | COLS
  <n>] [HEIGHT | ROWS <n>]
```

The FROM argument specifies the source file to be edited. This argument is required, but the FROM keyword is optional.

The **SIZE** argument changes the ED buffer size. ED has a default text buffer size of 40,000 bytes. For example:

```
1> ED Script SIZE 55000
```

increases the size of the buffer to 55,000 bytes.

The **WITH** argument specifies an ED command file that can contain any sequence of ED extended mode commands. When **WITH** is specified, ED executes the commands contained in the command file. The **WITH** argument's keyword is required if you use **WITH**.

The **WINDOW** argument specifies the console type, such as **RAW:0/0/640/256/EdWindow** or **CONSOLE:**. The **WINDOW** argument's keyword is required if you use **WINDOW**.

TABS sets the tab stop interval, which is the number of spaces to the right that the cursor moves when you press the Tab key. The default value for **TABS** is 3.

The **WIDTH** and **HEIGHT** arguments adjust the size of the ED window by specifying the number of characters to display horizontally and vertically. By default the ED window is 640 x 200 pixels or approximately 88 characters wide by 21 lines high.

Starting ED

Start ED from a Shell or with the Workbench Execute Command menu item. Open ED at the prompt by entering **ED** and a new or existing file name, as follows:

```
1> ED <filename>
```

where <filename> is the name of an existing file or a new file to be used for saving your work. If the file name specified cannot be found in the current directory, ED opens a blank window and displays the message **Creating new file.**

Using ED

All ED commands have key sequences and many are available through menus. You can enter ED commands in either of the following ways:

1. By choosing the command's menu item.
2. By typing in the command's key sequence and pressing Return.

In addition, you can use the mouse to perform some commands, such as those that control cursor movement.

You can work on files in ED with the following two command modes:

- Immediate** Commands are executed as soon as typed. ED opens in immediate mode.
- Extended** Commands are not executed until you press Return or Esc.

Immediate Commands

In immediate mode, ED executes commands right away. Specify an immediate command by pressing a single key or Ctrl+key combination or by using the mouse. All immediate commands have corresponding extended versions.

Immediate commands control the following:

- Cursor movement
- Text scrolling
- Text insertion
- Text deletion
- Repetition of commands

Moving the Cursor in Immediate Mode

The cursor can be positioned anywhere in text by moving the pointer to the desired spot and clicking the selection button. To move the cursor with the keyboard, use the arrow keys, Tab, and Ctrl+key combinations.

Note In ED, the Tab key only moves the cursor. It does not insert Tab characters or spaces in a line.

Move the cursor one position in any direction by pressing the appropriate arrow key. If the cursor is on the right edge of the screen, ED scrolls the text to the left to display the rest of the line. ED scrolls the text vertically one line at a time and horizontally ten characters at a time. You cannot move the cursor beyond the left edge of the line. If you try to move the cursor beyond the top or bottom of the file, ED displays a **Top of File** or **Bottom of File** message.

Additional key combinations that control cursor movement are:

Shift+up arrow	Top of the file.
Shift+down arrow	Bottom of the file.
Shift+left arrow	Left edge of the ED window (regardless of the margin setting).
Shift+right arrow	End of the current line.
Ctrl+]]	Right edge of current line (if cursor is already there, it is moved to the left edge).
Ctrl+E	Start of the first line on the screen (if cursor is already there, it is moved to the end of the last line on the screen).
Ctrl+T	Start of the next word.
Ctrl+R	Space following the previous word.
Tab	The next tab position (multiple of the TABS value; 3 by default).

If your file has more lines than can fit in the ED window, you can scroll through the file vertically. Scroll one line at a time by pressing the up or down cursor key to move in the corresponding direction. Move the text in jumps by pressing:

Ctrl+D	Moves 12 lines down through the file.
Ctrl+U	Moves 12 lines up through the file.

These commands do not move the cursor position in the window; they redraw the text in the window with the new line at the cursor position.

If something disturbs your screen, such as an alert from another program appearing in the ED window or message remarks in the status line, press:

Ctrl+V Refreshes the window display.

Inserting Text in Immediate Mode

Any characters typed in immediate mode are inserted at the current cursor position and the cursor is shifted to the right. Any characters to the right of the cursor are shifted to make room for new text. If the line is wider than the width of the window, the window scrolls to the right to show what you are typing. If you move the cursor beyond the end of the line, ED inserts spaces between the end of the line and any new characters inserted.

There is maximum limit of 255 characters in a line. If you add more characters, ED displays a **Line Too Long** message.

To split the current line at the cursor, press Return. Any text to the left of the cursor remains on the original line. All text under and to the right of the cursor moves down onto a new line. Pressing Return at the end of the line creates a new blank line.

Deleting Text in Immediate Mode

ED has no type over mode. To replace a word or line, you must delete the existing words and insert new information with the following keys and key combinations:

Backspace	Deletes the character to the left of the cursor.
Del	Deletes the character highlighted by the cursor.
Ctrl+O	If the cursor is over a space, all spaces up to the next character are deleted. If the cursor is over a character, all characters up to the next space are deleted.
Ctrl+Y	Deletes all characters from the cursor to the end of the line.

When text is deleted, any characters remaining on the line shift to the left and any text beyond the right edge of the screen becomes visible.

Changing Case in Immediate Mode

You can change the case of text by positioning the cursor and pressing Ctrl+F. If the letter is lower case, it becomes upper case and vice versa. Ctrl+F does not change non-alphabetic characters or symbols.

After you press Ctrl+F, the cursor moves to the right. You can hold down Ctrl+F to repeat the command until you change all the letters on the line.

Extended Commands

In extended mode, commands are displayed on the command line— or status line— at the bottom of the window. ED does not execute these commands until you press Return or Esc. If you use Esc to execute extended commands, ED remains in extended mode. If you use Return to execute extended commands, ED returns to immediate mode.

Extended commands manage the following:

- Program control
- Cursor movement
- Text modification
- Block control
- Searching and exchanging text

To enter extended mode, press Esc. An asterisk appears as a prompt in the status line. Extended commands consist of one or two characters. Multiple extended commands can be typed on a single command line by separating them with a semicolon. Commands can be grouped together for ED to repeat automatically. Use Backspace to correct mistakes.

You can also execute commands through the programmable menu and function keys. Reconfigure the menus and functions keys by assigning a command to the key or menu item of your choice as described on page 4-21.

Using String Delimiters

In some cases, commands require arguments, such as a number or a text string. A string argument for an ED command must be enclosed in a pair of identical delimiter characters. In unambiguous situations you may omit the trailing delimiter. Valid delimiters include " , / , \ , ! , ; , + , - , and % . You cannot use the same delimiter character inside your string. Invalid delimiter characters include letters, numbers, spaces, semicolons, question marks, brackets, and control characters.

Using a File Requester

You can also ask ED to use a file requester, allowing you to view the contents of the drives and directories in your system.

To invoke a file requester for a load or save command, you must place a question mark (?) before the required string argument. Be sure to include a space before the question mark (for example, **sa ?/Text/**). Normally, when a command is followed by a string, ED treats the string as the file to be loaded or saved and attempts the operation immediately. However, the question mark indicates that you want to specify the file through a file requester. You must still specify a string after the question mark, but the string becomes the text that appears in the file requester title bar.

ED Menus

ED has two sets of command menu assignments: default and expanded. The default menu assignments, as illustrated in Figure 4-2, are set up by the S:Ed-startup file, which is automatically executed each time you run ED. The S:Ed-startup file is a command file of ED extended mode commands, without the Escape characters. You can edit this file to set up custom menus, as described on page 4-21, or define preprogrammed function key assignments with the Set FN Key menu item.

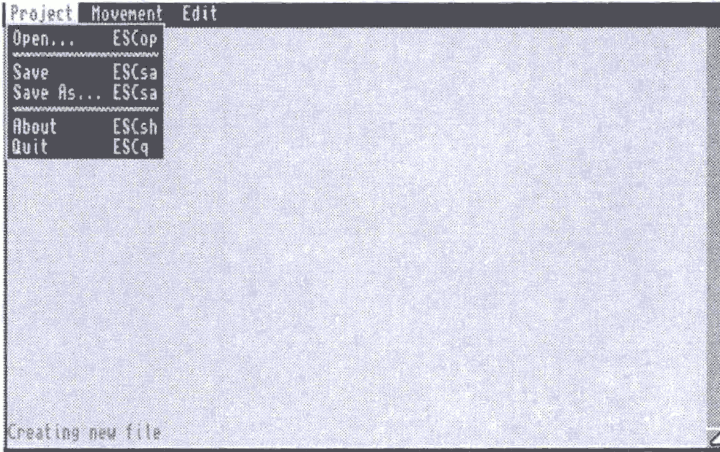


Figure 4-2. Default Menu Assignments

Enabling Expanded Menus

The expanded command menu assignments, as illustrated in Figure 4-3, can be enabled by renaming or deleting the default S:Ed-startup file. If ED cannot find a file named S:Ed-startup, it opens with the expanded set of menus, providing more options.

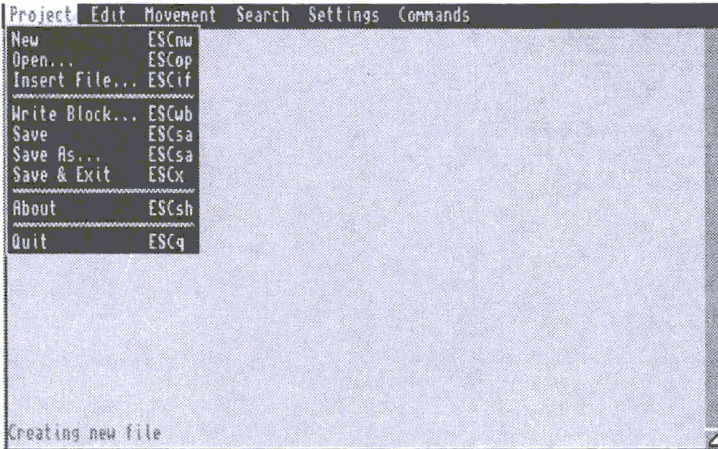


Figure 4-3. Expanded Menu Assignments

Rather than deleting your S:Ed-startup file, we recommend that you rename it as follows:

1. In the Workbench window, go to the Window menu and select Show All Files.
2. Double-click on the S drawer icon.
3. Click on the Ed-startup icon.
4. Go to the Icons menu and choose Rename.
5. Delete the name in the Rename requester's New Name text gadget and enter a new name for Ed-startup.
6. Select OK.

You can also create your own customized file of startup options. Avoid including Quit commands in the S:Ed-startup file since they can cause ED to quit immediately after opening.

The menu items in both the default and expanded menus have the same function regardless of which set you use. All of the ED commands are available through the keyboard using extended mode commands even if they do not appear in any menu.

The following sections describe the menu items found in the expanded menus and their corresponding extended and immediate mode commands. An ellipsis (...) indicates that an argument is required or that a menu item opens a requester or prompt.

Project Menu

The following are the expanded Project menu items:

New	Esc,N,W	Creates a new file, replacing the existing file. The message Edits will be lost-type Y to confirm: is displayed. Press any key (except Y) to abort the command.
Open...	Esc,O,P...	Opens a file. Specify the file by entering the path to the file as a properly delimited string. (If slashes appear in the path to a file, do not use the slash as a delimiter.) The message Edits will be lost-type Y to confirm: reminds you that you are replacing the current file.
Insert File...	Esc,I,F...	Inserts a file into the current file. ED reads into memory the specified file at the point immediately following the current line.
Write Block...	Esc,W,B...	Writes the currently marked block to a specified file. ED overwrites any other files with that name and copies the block to the file.
Save	Esc,S,A	Saves the text to the current file, overwriting the existing text in the file. Use Save As to save to a different file. SA followed by Q is equivalent to the X command.
Save As...	Esc,S,A...	Saves the text to the specified file name.
Save & Exit	Esc,X	Exits ED, saving the current file to the designated file name. ED writes the text it is holding in memory to the file that was specified when ED was opened and then terminates.

About	Esc,S,H	Shows the current state of the editor. The screen displays information, such as the value of tab stops, current margins, block marks, and the name of the file being edited.
Quit	Esc,Q	Exits ED without saving changes. If you made any changes to the file, ED asks if you want to quit. If you press Y, ED terminates immediately without saving the changes to the file.

Edit Menu

The following are commands used for editing:

Undo Line	Esc,U	Reverses changes made to the current line. However, ED cannot undo a line deletion. Once you have moved from the current line, the U command cannot undo a change.
Star Block	Esc,B,S	Identify the beginning and end of a block of text. To specify a block of text to be moved, inserted, or deleted, place the cursor on the first line that you want in the block and enter the BS command. Move the cursor to the last line that you want in the block and enter the BE command. You cannot start or finish a block in the middle of a line.
End Block	Esc,B,E	
Show Block	Esc,S,B	Redraws the display so the block is at the top of the screen.
Insert Block	Esc,I,B	Inserts a copy of the block after the current line. The block remains defined until you change the text. Use IB to insert copies of the block throughout the document.
Delete Block	Esc,D,B	Deletes a block.
Delete Line	Esc,D Ctrl+B	Deletes the entire line.

Movement Menu

The following commands move the cursor around the screen:

Top	Esc,T	Top of the file; first line of the file is brought to the top of the window.
Bottom	Esc,B	Bottom of the file; last line of the file is brought to the bottom of the window.
Go To Line...	Esc,M...	Move the cursor to the specified line. Enter the line number on the status line and press Return. The line specified is brought to the top of the window. If no number is given, the cursor goes to the top of the window.
Next Page	Esc,P,D Ctrl+D	Go to next page.
Previous Page	Esc,P,U Ctrl+U	Go to previous page.

Search Menu

The following commands let you search through the file for specific instances of text. You can substitute one pattern of text with another (search and replace) and have ED request confirmation of (query) each replace. If the specified text is not found or there are no more instances of the text, the message **Search failed** is displayed. When using the Find and Replace menu commands, ED prompts for the text strings. Enter the text without delimiters. When using extended mode, include delimited strings with the command.

Find... Find Next	Esc,s...	Finds the next occurrence of the specified string of text. The search starts one character beyond the current cursor position and continues forward through the file. If the string is found, the cursor moves to the start of the located string. The search is case-sensitive, unless the Ignore Case command is used. Find Next repeats the command.
------------------------------------	----------	---

Reverse Find... Reverse Find Next	Esc,B,F...	Searches backwards through the file for the specified string. This command finds the last occurrence of the string before the current cursor position. The search continues through to the beginning of the file. Reverse Find Next repeats the command.
Replace...	Esc,E...	Exchanges one occurrence of text with another. In extended mode, enter the strings enclosed by three delimiters. For example, to replace the word to with too, enter "to"too". Specify empty strings by typing two delimiters with nothing between them. If the first string is empty, ED inserts the second string at the current cursor position. If the second string is empty, ED searches for the next occurrence of the first string and then deletes it. Note that ED ignores margin settings when exchanging text.
Global Replace...	Esc,R,P, E...	Exchanges all occurrences of text.
Query- Replace...	Esc,E,Q...	Searches for the text to be exchanged and requests verification by displaying Exchange? . Enter Y to exchange or another other key to abort.
Global Query- Replace...	Esc,R,P, E,Q...	Searches for all occurrences of the text to be exchanged and requests verification for each. Enter Y to exchange or any other key to abort.

Settings Menu

The following commands are used for setting up your ED environment:

Set FN Key...	Esc,S,F...	Defines the function keys and other programmable keys. Defining function key and Ctrl+key commands is similar to defining menu items. See page 4-15 for instructions for defining function keys and an example of the Set FN Key command.
Show FN Key...	Esc,D,F <key>	Displays the setting for the function key specified by <key>. Enter a space and a key slot number for <key>.
Reset Keys	Esc,R,K	Resets the key definitions to the default. See page 4-16 for a table of special key mappings.
Right Margin...	Esc,S, R...	Sets the right margin. Use the SR command followed by a number indicating the column position.
Left Margin...	Esc,S,L...	Sets the left margin. Use the SL command followed by a number indicating the column position. The left margin should not be set beyond the right edge of the screen.
Ignore Case	Esc,U,C	Specifies a case-insensitive search. UC instructs all subsequent searches not to make any distinction between upper and lower case text. To make searches case-sensitive again, use the LC command.
Case Sensitive	Esc,L,C	Specifies a case-sensitive search.

Set FN Key

Set FN Key is used to define function keys and other programmable keys. There are 57 immediate command key slots ranging from 1 to 57. Any slot number can be redefined and any numbers within the range that do not appear in the special key mappings on page 4-16 are not defined.

The following is the syntax for the Set FN Key command:

```
SF <slot number> /command string/
```

Define Ctrl+key combinations by substituting a caret (^) and the other character for the slot number.

Example Script

This example script assigns function keys to cursor control commands. You can also enter these as a series of extended mode commands. The Top of File, Bottom of File, End of Page, Next Page, Next Line, and Previous Line commands are assigned to the F1 through F6 keys, respectively. Quotation marks are used as delimiters.

```
SF 1 "t"
SF 2 "b"
SF 3 "ep"
SF 4 "pd"
SF 5 "n"
SF 6 "p"
```

Special Key Mappings

The following table shows the default key definitions used in the Reset Keys command:

Slot #	Key/Key Sequence	Function
1-10	F1 through F10	Not defined
11-20	Shift+F1 to Shift+F10	Not defined
21	Shift+left arrow	Move to beginning of line
22	Shift+right arrow	Move to end of line
23	Shift+up arrow	Move to top of document
24	Shift+down arrow	Move to bottom of document
25	Del	Delete character at cursor
26	Not defined	Not defined
27	Ctrl+A	Insert line
28	Ctrl+B	Delete line
29	Ctrl+C	Not defined
30	Ctrl+D	Move down 12 lines

Slot # (cont'd)	Key/Key Sequence (cont'd)	Function (cont'd)
31	Ctrl+E	Move to top or bottom of screen
32	Ctrl+F	Change case
33	Ctrl+G	Repeat last extended command line
34	Ctrl+H	Delete character left of cursor
35	Ctrl+I	Move cursor to next tab position
36	Ctrl+J	Not defined
37	Ctrl+K	Not defined
38	Ctrl+L	Not defined
39	Ctrl+M	Return
40	Ctrl+N	Not defined
41	Ctrl+O	Delete word or spaces
42	Ctrl+P	Not defined
43	Ctrl+Q	Not defined
44	Ctrl+R	Move to end of previous word
45	Ctrl+S	Not defined
46	Ctrl+T	Move to start of next word
47	Ctrl+U	Move up 12 lines
48	Ctrl+V	Redisplay window
49	Ctrl+W	Not defined
50	Ctrl+X	Not defined
51	Ctrl+Y	Delete to end of line
52	Ctrl+Z	Not defined
53	Ctrl+[Esc (enter extended command mode)
54	Not defined	Not defined
55	Ctrl+]]	Move to end or start of line, depending on cursor position
56	Not defined	Not defined
57	Not defined	Not defined

Command Menu

The following commands are for manipulating files:

Extended Command...	Esc,C,M...	Enters extended command mode; equivalent to pressing Ctrl+[or Esc.
Repeat Last	Esc,R,E	Attempts to repeat the last command.
Run File...	Esc,R,F...	Loads and executes a command file of extended mode commands.
ARexx Command...	Esc,R,X...	Runs the specified ARexx program.
Redisplay	Esc,V,W	Redraws the ED window and clears the status line; equivalent to pressing Ctrl+V.

Other ED Commands

There are also ED commands that do not appear in menus. These commands are listed here in functional groups. Use them in extended mode by entering the following key sequences.

Program Control

The following are program control commands:

Extend Margins	Esc,E,X	Extends the margins for the current line. Once you enter the EX command, ED ignores the right margin on the current line.
Status Line Message	Esc,S,M...	Prints a given string on the status line.
Exit with Query	Esc,X,Q	Exits ED unless changes were made to the file. If changes have been made, the message File has been changed—type Y to save and exit: is displayed. Press any key (except Y) to abort the exit. XQ is equivalent to clicking the close gadget on the ED window.

Cursor Control

The following commands are used for controlling the cursor:

End Page	Esc,E,P	End of a page.
Previous	Esc,P	Start of the previous line.
Character Left	Esc,C,L	One place to the left.
Character Right	Esc,C,R	One place to the right.
Current End	Esc,C,E	End of the current line.
Current Start	Esc,C,S	Start of the current line.
Tab	Esc,T,B	Next tab position.
Word Next	Esc,W,N	Start of the next word.
Word Previous	Esc,W,P	Space after previous word.

Modifying Text

The following commands edit text on the screen:

Insert Before	Esc,I	Inserts the specified string on the line before the cursor. Specify a new line's string after the I command to insert text before the current line containing the cursor.
Insert After	Esc,A	Inserts the specified string on the line after the cursor. This command works in the same way as I, except that the string is inserted on a new line beneath the current cursor position.
Split	Esc,S	Splits the current line at the cursor position.
Join	Esc,J	Joins the next line to the end of the current line.
Delete	Esc,D	Deletes the current line.
Delete Character	Esc,D,C	Deletes the character under the cursor.
Delete Left	Esc,D,L	Deletes the character to the left of the cursor.
Delete Word	Esc,D,W	Deletes to the end of the current word.
End Line	Esc,E,L	Deletes to the end of the current line.

Flip Case	Esc,F,C	Switches the case of the selected letters, one at a time.
Set Tab	Esc,S,T	Sets the tab stop. To change the current setting of tabs, use the ST command followed by a number.
Next	Esc,N	Start of the next line.

Repeating Commands in Extended Mode

Pressing Ctrl+G repeats a command line. You can set up and execute complex sets of editing commands many times.

You can repeat a command a specified number of times by entering the number before the command. For example:

```
4 E/rename/copy/
```

exchanges the next four occurrences of "rename" to "copy".

Use the RP (Repeat) extended command to repeat a command until ED returns an error, such as reaching the end of the file. For example:

```
T;RP E/rename/copy/
```

moves the cursor to the top of the file, then exchanges all occurrences of "rename" with "copy". The T command (Top of File) changes all occurrences of Rename in the whole file. Otherwise, only the occurrences after the current cursor position are changed.

To execute command groups repeatedly, you can group the commands together in parentheses. You can also nest command groups. For example:

```
RP (F/Workbench/;3A//)
```

inserts three blank lines (the null string //) after every line containing Workbench.

To interrupt any sequence of extended commands, press any key during execution. If an error occurs, ED abandons the command sequence.

Customizing ED

You can customize ED with commands that change the menus and function key setup. These commands can be entered individually within ED. They can also be saved as a script, such as S:Ed-startup, or as a file specified using the WITH argument. To execute the file from within ED, use the Run File (Esc,R,F) extended command. For information about changing the function keys, see page 4-15.

Set Menu Item	Esc,S,I	Defines the menu headings and items. There are 120 menu item slots ranging from 0 to 119. The slot type identifies the contents of the slot and is a number from 0 to 4. The 0 slot type must be the last defined slot. Do not create a menu without items; if you specify a menu heading, include menu items after it. See below for the syntax of the Set Menu Item command and a table of the slot types.
Enable Menu	Esc,E,M	Enables menus. You must follow the Set Menu Item commands with EM to enable the menu commands. See page 4-22 for an example script using the Enable Menu command.

Set Menu Item

The following is the syntax for the Set Menu Item command:

```
SI <slot number> <slot type> /string1/string2/
```

The following table shows the slot types and functions used with the Set Menu Item command:

Type	Function	String Input
0	End of Menu	No arguments
1	Menu Heading	String1 = heading name
2	Menu Item	String1 = item name String2 = command string

Type (cont'd)	Function (cont'd)	String Input (cont'd)
3	Submenu Heading	String1 = heading name String2 = command string
4	Separator bar	No arguments

Example Script

The following is an example script using the Set Menu Item and Enable Menu commands. Quotation marks are used as the delimiters.

```
SI 0 1 "Project"
SI 1 2 "Open ... " "op ? /Open file:/"
SI 2 2 "Save ... " "sa"
SI 3 4
SI 4 2 "Quit!" "q"
SI 5 1 "Move"
SI 6 2 "Top" "t"
SI 7 2 "Bottom" "b"
SI 8 0
EM
```


This script produces the menus illustrated in Figure 4-4:

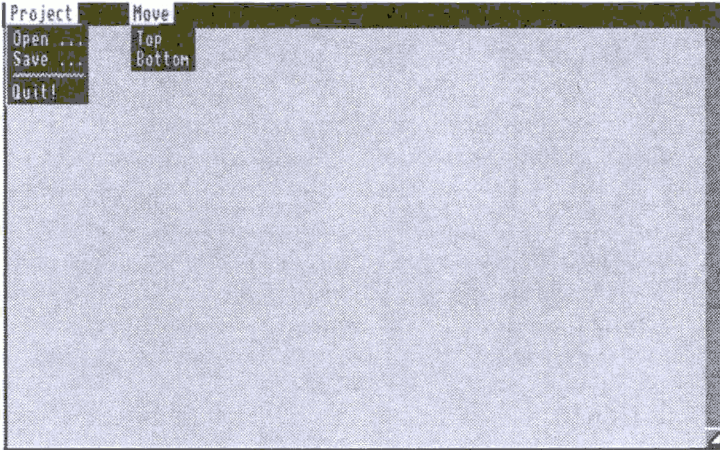


Figure 4-4. ED Custom Menu Example

Printing From ED

Use the following steps to print a file that is open in the current ED window:

1. Choose the Save As menu item to display a file requester.
2. Enter **prt:** in the Drawer field.
3. Select OK.

This prints the file, but does not save it to disk. If you wish to save the file you must select the Save menu item or Save As and a file name.

Quitting ED

You can exit ED in one of the following three ways:

- **Esc,X**. This method exits ED and saves the current file to a designated file name that was specified when ED was opened.
- **Esc,X,Q** or click on the ED window's close gadget. This method exits ED if no changes were made to the file. If changes were made to the file, ED lets you save the changes or exit without saving.
- **Esc,Q** or select the **Quit** item in the Project menu. This method exits ED without saving any changes if you confirm the status line warning that changes will be lost.

ARexx Support

You can also control ED from ARexx by sending and receiving commands through ED's ARexx port. Each copy of ED running concurrently has an individual ARexx port name that must be specified to handle information for the correct session. The ARexx port names are assigned as follows:

- The first session's port name is **Ed**
- The second session's is **Ed_1**
- The third session's is **Ed_2** and so on

Many of ED's extended commands can be used from ARexx. By using ED's **RV** command in ARexx programs, you can send information from ED to ARexx. This gives information about the status of ED, such as the current line number or the name of the file being edited.

The **RV** command accepts the name of the ARexx stem variable to store its argument information. For example, in ARexx:

```
address 'Ed' 'RV /stem/'
```

assigns values to the following variables:

stem.LEFT	Current left margin
stem.RIGHT	Current right margin
stem.TABSTOP	Current tab stop setting

stem.LMAX	Maximum number of lines visible on screen
stem.WIDTH	Width of the screen in characters
stem.X	Cursor X position in the ED window (1 is the left edge)
stem.Y	Cursor Y position in the ED window (1 is the top line)
stem.BASE	Window base (normally 0, but non-zero when the screen is shifted to the right)
stem.EXTEND	Extended margin value (Extend Margins command)
stem.FORCECASE	Case sensitivity flag (Ignore Case = 1, Case Sensitive = 0)
stem.LINE	Current line number in the file (1 is the first line)
stem.FILENAME	Name of the file being edited
stem.CURRENT	Text of the current line
stem.LASTCMD	Last extended command issued
stem.SEARCH	Last string searched for

Any valid ARexx symbol can be substituted for "stem." Enclose the name in proper delimiters. These variables can be treated as ordinary ARexx stem variables.

ED/ARexx Example Program

The example program, *Transpose.ed*, illustrates the use of several extended commands from ARexx. This program transposes two characters when launched from ED. For example, if a line contains the string 123 and the cursor is highlighting the 3, *Transpose.ed* changes the string to 213.

Enter this program and save it as `REXX:Transpose.ed`. Then, open ED and edit an existing file or create a new one. Place the cursor one character to the right of the ones to be transposed, press Esc, and enter:

```
RX /transpose.ed/
```

The program executes and the characters are transposed if ARexx is running and everything is entered correctly. The entire file name, including the extension, must be specified to run the program.

Sample Program

```

/*Transpose.ed: An example program to transpose two characters.      */
/* Given string '123', if cursor is on 3, this macro converts      */
/* string into '213'.                                              */
HOST = address ()          /* find out which ED session invoked this program */
address VALUE HOST        /*...and talk to that session      */
'rv' '/CURR/'             /* Ask ED to store info in stem variable CURR */
                           /* Obtain two pieces of information:      */
currpos = CURR.X           /* 1. position of cursor on line      */
currlin = CURR.CURRENT    /* 2. contents of current line       */
if (currpos > 2) then     /* Work only on the current line     */
  currpos = currpos - 1
else do                    /* Otherwise, report error and exit  */
'sm /Cursor must be at position 2 or further to the right/'
exit 10
end
/* Next the code needs to reverse the CURRPOSTh and CURRPOSTh-1  */
/* characters and then replace the current line with the new one.  */
/* drop CURR. CURR is no longer needed; dropping it saves some   */
/* memory. */
'd'                        /* Tell ED to delete current line    */
currlin = swapch (currpos,currlin) /* Swap the two characters          */
'i' /!|currlin!/'        /* Insert modified line              */
do i = 1 to currpos      /* Place cursor back where it started */
  'cr'                    /* ED's 'cursor right' command      */
end
exit                       /* Program has finished              */
/* Function to swap two characters                                  */
swapch: procedure
parse arg cpos,clin
  ch1 = substr(clin,cpos,1) /* Get character                      */
  clin = delstr(clin,cpos,1) /* Delete it from string              */
  clin = insert(ch1,clin,cpos-2,1) /* Insert to create transposition    */
return clin                /* Return modified string             */

```

MEMacs

MEMacs (MicroEmacs), which is similar to the UNIX-based Emacs editor, is a screen-oriented editor in which you can edit multiple files at the same time. MEMacs performs all operations on memory-resident text, requiring that entire text files be able to fit into memory at once.

Line length, generally 80 characters long, is limited to the right edge of the screen. You can enter characters beyond the limit, however, they are not displayed. To see these characters, break the line or delete some of the displayed characters. A dollar sign (\$) at the right edge of the screen indicates that there are characters beyond what is displayed.

The format for the MEMacs command is the following:

```
MEMACS [<filename>] [GOTO <n>] [OPT W]
```

The <filename> argument is optional.

The GOTO <n> option specifies the line on which the cursor is to appear when the file is opened.

Specifying OPT W opens MEMacs in a Workbench window rather than on its own screen, which saves memory.

Starting MEMacs

MEMacs can be run from either the Workbench or the Shell. From the Workbench, double-click on the MEMacs icon in the Tools window of the Extras disk. If you have a hard disk, the Tools drawer is in your Workbench window.

From the Shell, enter:

```
MEMacs <filename>
```

where <filename> specifies the file to read into MEMacs. If a file with that name does not previously exist, a new file is created when you save your work.

MEMacs Commands

The line at the bottom of the MEMacs screen identifies either the current file name or the name of the current buffer if no file name is specified. Figure 4-5 illustrates the MEMacs opening screen.

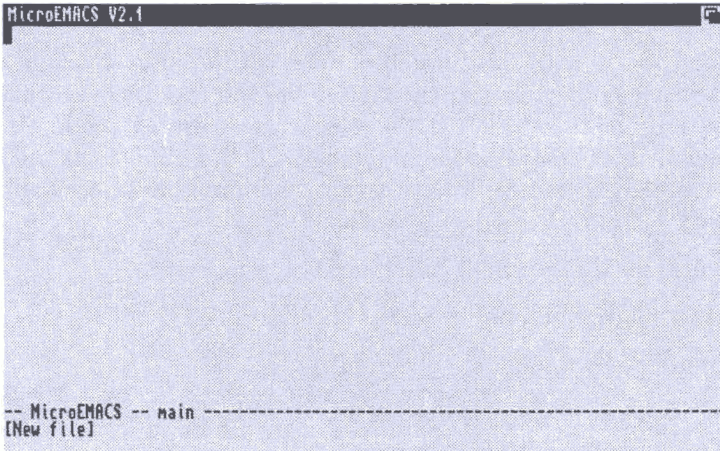


Figure 4-5. MEMacs Opening Screen

Several buffers can be in use at the same time and one or more can be displayed on the screen simultaneously. Menu options switch between them. At all times, the screen displays what is actually in the buffer.

MEMacs has two conditions of operation:

- Normal** When you enter and manipulate text directly in the file without special functions.
- Command** When you enter a command through a menu selection or the keyboard shortcut for it. In the command condition, the cursor jumps to the bottom line of the display and waits for you to supply additional information following the prompt. You cannot return to the normal condition until you satisfy or cancel the command by pressing Return.

In the MEMacs normal condition, you can:

- Move the cursor using the arrow keys.
- Move the cursor to the edge of the window by holding down Shift and pressing the appropriate arrow key.
- Move the cursor by clicking the left mouse button at the desired place on the screen.
- Insert characters at the current cursor position by typing them.
- Delete the character at the current cursor position by pressing Del.
- Delete the character to the left of the cursor by pressing Backspace.
- Perform other special menu and command functions.

When using MEMacs, you should be familiar with the following special terms:

Buffer A memory area that MEMacs controls. There is always at least one buffer used by MEMacs containing zero or more text characters.

Dot The current cursor position.

Mark A specified cursor position. (Each buffer has its own dot and mark.) The Set-mark menu item marks the current cursor position (described on page 4-33). You can move forward or backward in the file, adding or deleting text. To return to the marked place, select the Swap-dot&mark menu item (described on page 4-36).

You can also set a mark to indicate the beginning of a block of text that you want to duplicate, move, or delete. The block encompasses all the characters starting with the mark and continuing to the current cursor position.

Kill Kill commands remove text from the screen to save in a kill buffer. This text can be retrieved and inserted into your document by using the Yank command. Issuing successive Kill commands (without selecting Yank in between) adds each block of text to the existing text in the kill buffer. If you select Yank, the next block of killed text overwrites the current block.

Window MEMacs screens can be split into multiple layers for editing and displaying more than one buffer or two or more portions of the same buffer. Each layer is a MEMacs window.

Modified Buffers Buffers are marked as modified when any changes are made. The modified status is removed when the buffer is saved.

To see modified buffers, use the List-buffers command (described on page 4-33); modified buffers are identified with an asterisk (*). If you exit MEMacs without saving any changes, a prompt tells you that modified buffers exist and asks if you really want to quit.

Menu Commands

MEMacs has the following menus:

Project	Contains system and file-oriented items.
Edit	Contains buffer editing commands.
Window	Controls the characteristics of the MEMacs windows.
Move	Controls the placement of the cursor.
Line	Controls line-oriented operations.
Word	Controls word-oriented operations.
Search	Controls search and search/replace options.
Extras	Controls the numerical value of arguments and lets you execute a series of operations as though it were a single special command.

Figure 4-6 illustrates the MEMacs expanded menu bar.

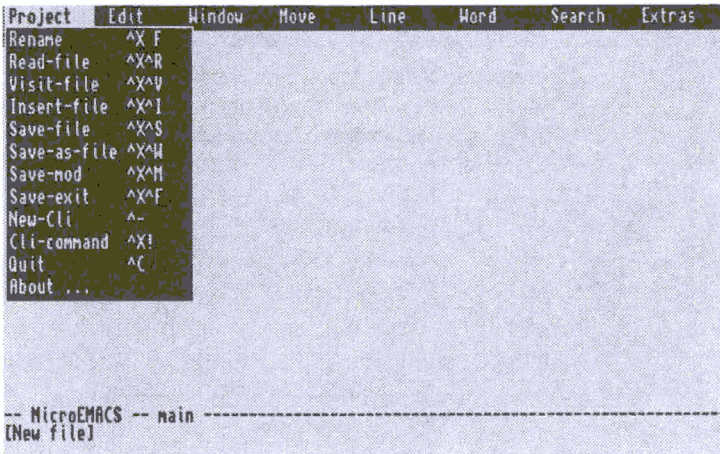


Figure 4-6. MEMacs Expanded Menus

Project Menu

The commands in the Project menu, except for Visit-file, affect the buffer associated with the current cursor position.

Rename	Ctrl+X,F	Changes the name of the file associated with the current buffer. Pressing Return without specifying a file name disassociates the buffer from any file name.
Read-file	Ctrl+X, Ctrl+R	Replaces the contents of the current buffer with the contents of a file. Enter a complete file path. Press Return without specifying a file name to ignore the request and return to normal mode.
Visit-file	Ctrl+X, Ctrl+V	Allows you to work with additional files other than the one you are currently editing. Enter the complete file path.

Insert-file	Ctrl+X, Ctrl+I	Inserts the contents of a file into the current buffer at a point one line above the current cursor position. Enter the complete file path.
Save-file	Ctrl+X, Ctrl+S	Writes the contents of the current buffer to the file name associated with that buffer. Issues the file's line count following a successful save. MEMacs does not save the file if no name is provided; it displays this error message: No file name.
Save-as-file	Ctrl+X, Ctrl+W	Allows you to specify the name and path of a file associated with a buffer.
Save-mod	Ctrl+X, Ctrl+M	Writes the contents of all modified buffers to the disk. Do not accidentally modify a buffer that you did not intend to change.
Save-exit	Ctrl+X, Ctrl+F	Saves all modified buffers and exits MEMacs.
New-Cli	Ctrl+-	Opens a new Shell window known as a Spawn Window. Enter AmigaDOS commands in the Spawn Window without interfering with MEMacs. Close the window with ENDSHELL.
Cli-Command	Ctrl+X,!	Lets you execute an AmigaDOS command while in MEMacs. Enter a command following the ! prompt at the bottom of the screen. Command output is placed in the spawn.output buffer.
Quit	Ctrl+C	Exits MEMacs. You are given an opportunity to save modified buffers or quit without saving. Alternative keyboard shortcuts: Ctrl+X, Ctrl+C Esc, Ctrl+C.
About...		Gives program copyright information.

Edit Menu

The commands in the Edit menu affect the editing of your buffers and their associated files.

Kill-region	Ctrl+W	Deletes blocks of text from the current buffer and saves them in a kill buffer. Text can be retrieved with the Yank command. Make copies of a block by immediately selecting Yank without changing the cursor position after killing the block. This restores the block to its position and leaves a copy in the kill buffer.
Yank	Ctrl+Y	Copies the contents of the kill buffer to the current cursor location on the current line. Reverses the action of Kill-region without changing the contents of the kill buffer. Used with Kill-region for moving text or for repeatedly copying a single block of text.
Set-mark	Ctrl+@	Marks the cursor position in a buffer. The subsequent cursor position is referred to as a dot. Move between the mark and the dot using the Swap-dot&mark command in the Move menu. Used for marking blocks of text. Alternative keyboard shortcut: Esc,-.
Copy-region	Esc,W	Copies the contents of the marked region to the kill buffer without deleting it, replacing any previous contents.
Upper-region	Ctrl+X, Ctrl+U	Changes the text of the entire marked region to upper case.
Lower-region	Ctrl+X, Ctrl+L	Changes the text of the entire marked region to lower case.
List-buffers	Ctrl+X, Ctrl+B	Splits the current buffer's window and displays a list of the buffers MEMacs is maintaining. To redisplay the current buffer, select the One-window command or press Ctrl+X,1. The List-Buffer fields are: C Displays an asterisk if the buffer has been modified since it was last saved to a file. (Stands for changed.)

		Size	Shows how many characters are in the buffer.
		Buffer	Shows the name given to the buffer. If you read in a file, this is the name of the file without the full path.
		File	Shows the full path to the file. This is the file to which MEMacs writes the buffer if you Save-file or Save-exit while the cursor is in that buffer.
Select-buffer	Ctrl+X,B		Allows you to select the buffer to edit in the current window. Replaces the contents of the window with the selected buffer or new buffer.
Insert-buffer	Esc, Ctrl+Y		Inserts the contents of a named buffer into the current buffer at the line above the current cursor position.
Kill-buffer	Ctrl+X,K		Deletes the contents of one or more chosen buffers, returning the memory to the memory manager to reuse. You must specify the buffer to be deleted; a buffer cannot be killed if its contents are currently displayed.
Justify-buffer	Ctrl+X,J		Removes all blank spaces and tabs from the left edge of all the lines in the current buffer. The text realigns with the current margins.
Redisplay	Ctrl+L		Redraws the screen.
Quote-char	Ctrl+Q		Allows insertion of any literal character in the text file, except the Tab key. Alternative keyboard shortcut: Ctrl+X,Q.
Indent	Ctrl+J		Moves the cursor to the next line, automatically indenting the same amount of spaces as the previous line. Alternative keyboard shortcuts: Help or Enter on the numeric keypad.

Transpose	Ctrl+T	Swaps the positions of two adjacent characters. Place the cursor on the right-most of the two characters and execute the command.
Cancel	Ctrl+G	Ends an ongoing menu command, such as query search and replace.

Window Menu

The Window menu controls how to view your buffers on the screen.

One-window	Ctrl+X,1	Makes the current buffer a single, full-sized window on the MEMacs screen.
Split-window	Ctrl+X,2	Splits the current window in half, positioning the current buffer identically in both windows. Any changes you make in either window affect the whole buffer.
Next-window	Ctrl+X,N	Moves the cursor to the next window and makes that window available for editing.
Prev-window	Ctrl+X,P	Moves the cursor to the previous window and makes that window available for editing.
Expand-window	Ctrl+X,Z	Adds a line to the height of the current window and simultaneously subtracts a line from the height of the adjacent window.
Shrink-window	Ctrl+X, Ctrl+Z	Subtracts a line from the height of the current window and adds a line to the height of the adjacent window.
Next-w-page	Esc, Ctrl+V	Displays the next page of the adjacent window. This does not make the window available for editing.
Prev-w-page	Ctrl+X,V	Displays the previous page of the adjacent window. If only one window is displayed, it displays the previous page of that window.

Move Menu

These commands move the cursor rapidly through the current buffer.

Top-of-buffer	Esc,<	Moves the cursor to the top line of the current buffer.
----------------------	-------	---

End-of-buffer	Esc,>	Moves the cursor to the bottom line of the current buffer.
Top-of-window	Esc,,	Moves the cursor to the top of the current window.
End-of-window	Esc,.	Moves the cursor to the bottom of the current window.
Goto-line	Ctrl+X, Ctrl+G	Moves the cursor to a specified line number. Specifying a line number larger than the total number of lines in a buffer moves the cursor to the last line of the buffer.
Swap-dot&mark	Ctrl+X, Ctrl+X	Marks the current cursor position (dot) and moves the cursor to the previously marked setting.
Next-page	Ctrl+V	Moves the cursor toward the end of the buffer by one full window, less one line.
Prev-page	Esc,V	Moves the cursor towards the beginning of the buffer by one full window, less one line.
Next-word	Esc,F	Moves the cursor forward to the next non-alphabetic character, such as a space or punctuation mark, following the current word.
Previous-word	Esc,B	Moves the cursor back to the first letter of the previous word.
Scroll-up	Ctrl+Z	Moves the text up one line.
Scroll-down	Esc,Z	Moves the text down one line.

Line Menu

These commands move the cursor within or between lines and perform operations involving entire lines.

Open-line	Ctrl+O	Splits the line containing the cursor, forcing the character at the current cursor position to become the first character of the following line. The cursor remains on the original line. Pressing the Del key cancels an accidental Open-line.
Kill-line	Ctrl+X, Ctrl+D	Deletes the line containing the cursor and places the text in the kill buffer.

Kill-to-eol	Ctrl+K	Deletes the text between the current cursor position and the end of the line and places the text in the kill buffer.
Start-of-line	Ctrl+A	Moves the cursor to the first position on a line.
End-of-line	Ctrl+E	Moves the cursor to the last position on a line.
Next-line	Ctrl+N	Moves the cursor down one line.
Previous-line	Ctrl+P	Moves the cursor up one line.
Line-to-top	Esc,!	Moves the line containing the cursor to the top of the window.
Delete-blanks	Ctrl+X, Ctrl+O	Deletes blank lines, proceeding forward from the current cursor position until MEMacs reaches the next line on which there is text. Does not delete single blank lines.
Show-Line#	Ctrl+X,=	Displays information on the present cursor position.

Word Menu

The Word menu contains word-associated operations.

Delete-forw	Esc,D	Deletes the character on which the cursor is positioned and all remaining characters to the right until the next non-alphanumeric character is found, such as a blank space, tab, or punctuation.
Delete-back	Esc,H	Deletes all characters to the left of the cursor until it finds the first character of a word. The character at the cursor position is not deleted. Alternative keyboard shortcut: Esc,Del.
Upper-word	Esc,U	Changes a word to upper case, starting at the cursor position and proceeding to the last character of the word.
Lower-word	Esc,L	Changes a word to lower case, starting at the cursor position and proceeding to the last character of the word.

Cap-word	Esc,C	Changes the character at the cursor position to upper case. Also changes the characters to the right of the cursor, up to the end of the word, to lower case.
Switch-case	Esc,^	Changes the case of a word, starting at the current cursor position and proceeding to the right until it reaches the end of the word. If a word is upper case or has mixed text, it changes upper to lower case and vice versa.

Search Menu

These commands search through the current buffer for specific text strings. The case (upper or lower) of the string is not significant in the search. However, if you are using text substitution (search and replace), the text is replaced in the case of the replacement string.

Search-forward	Ctrl+S	Searches through the text starting at the current cursor position and moving forward to the end of the buffer. At the prompt, enter the character string for the search. Alternative keyboard shortcut: Ctrl+X,S.
Search-backward	Ctrl+R	Searches through the text from the current cursor position backwards to the beginning of the buffer. Alternative keyboard shortcut: Ctrl+X,R.
Search-replace	Esc,R	Searches the same as Search-forward, allowing you to replace the string with different text. At the prompt, enter the replacement string of characters.
Query-s-r	Esc,Q	Operates the same as Search-replace, except it asks for confirmation to replace each time it finds the specified string. The options are Y (yes), N (no), C (change all occurrences), and Ctrl+G (abort).
Fence-match	Esc, Ctrl+F	Finds the closest occurrence of the fence character to match the one at the current cursor position. Fence characters are parenthesis, brackets, braces, and angle brackets.

Extras Menu

These commands are MEMacs operational commands and macro commands. Specific numeric arguments may be required before selecting a command; an * indicates that an argument is required. Macro commands are executed by selecting the Execute-macro menu item.

Set-arg	Ctrl+U	Lets you specify a numeric argument for a command.
Set	Esc,S	Lets you set the following MEMacs parameters:
	Screen	Places the MEMacs display in a Workbench window or back onto a custom screen.
	Interlace	Turns the interlace mode on or off.
	Mode	Results in a second prompt "Mode:"; you can enter Cmode (for editing C programs) or Wrap (to enable automatic word-wrap when the text reaches a set cursor position). Cmode provides automatic fence matching. Use +mode or -mode to add or subtract a mode.
	Left	Determines the left margin. Prompts for a numerical argument if not provided with the entry.
	Right	Determines the right margin. Prompts for a numerical argument if not provided with the entry.
	Tab	Sets the increment for tab spacing. Prompts for a numerical argument if not provided with the entry.

	Indent	Determines how far to indent each level of nesting (used in Cmode). Prompts for a numerical argument if not provided with the entry.
	Case	Turns case-sensitive searches on or off; default is off.
	Backup	Turns the MEMacs backup function on or off. Your options are: ON (renames the current file <filename>.bak and saves that backup file to the T: directory); SAFE (this option checks to see if a file already exists for the buffer — if so, it displays an error and does not overwrite the existing file, Ctrl+X clears the display); and OFF (this is the default option — MEMacs does not perform any backup).
Start-macro	Ctrl+X,(Tells MEMacs to start recording any subsequent keystrokes. Used with the Stop-macro and Execute-macro commands.
Stop-macro	Ctrl+X,)	Tells MEMacs to stop recording keystrokes.
Execute-macro	Ctrl+X,E	Repeats keystrokes that were entered between Start-macro and Stop-macro.
Set-key	Ctrl+X, Ctrl+K	Allows you to redefine all of the function keys, the Shifted function keys, the Help key, or any key on the numeric keypad as keyboard macros. You cannot use the menu shortcut of Ctrl+@ to insert the Set-mark command into any keyboard macro definitions.
Reset-keys	Esc,K	Returns any keys defined by Set-keys to their original default state.
Execute-file	Esc,E	Allows you to execute a program file within MEMacs.

Execute-line	Ctrl+[, Ctrl+[Sets MEMacs to command mode. At the prompt enter any menu command and its parameters. Alternate keyboard shortcut: Esc,Esc.
---------------------	-------------------	---

The following table contains the default values of the Set-key/function keys when used in macro commands.

Key	Assignment	Key Sequence
F1	Clone line	Ctrl+A,Ctrl+K,Ctrl+Y,Ctrl+M,Ctrl+Y
F2	Delete line	Ctrl+X,Ctrl+D
F3	Execute keyboard macro	Ctrl+X,E
F4	Next screen	Ctrl+V
F5	Previous screen	Esc,V
F6	Split window	Ctrl+X,2
F7	One window	Ctrl+X,1
F8	Scroll window up	Ctrl+Z
F9	Scroll window down	Esc,Z
F10	Save file and exit	Ctrl+X,Ctrl+F
Help	Insert line	Ctrl+J
Enter (keypad)	Insert line	Ctrl+J

Commands Not in Menus

The following commands are only accessible through the keyboard.

Keys are bound when they can be used to perform a function. For example, any key or key sequence that can be used as a shortcut for a menu item is bound to that menu item.

Describe Key	Esc, Ctrl+D	Tells you if any functions are bound to a key or key sequence. At the prompt enter the specific key or key sequence.
---------------------	----------------	--

Bind Key	Esc, Ctrl+B	Allows you to bind a key to a function. At the prompt, enter the key or key sequence.
-----------------	----------------	---

Unbind Key	Esc, Ctrl+U	Allows you to return a bound key to an unbound state. At the prompt, enter the key or key sequence. Standard bound keys cannot be unbound.
Echo	Esc, Ctrl+E	Displays the string entered in the command line.
Move to Edge of Window	Shift+ Arrow	Moves the cursor to the top, bottom, left, or right edge of the screen.
Delete the Next Character	Ctrl+D	Deletes the character at the current position. Same as pressing Del.
Delete the Previous Character	Ctrl+H	Deletes the character to the left of the current cursor position. Same as pressing Backspace.
Move to Next Line	Ctrl+M	Inserts a newline character after the current cursor position and moves the cursor to the start of the new line.
Move x number of Characters	Ctrl+F, Ctrl+B	Allows you to move the cursor forward or backward a specified number of spaces. Providing no value moves the cursor only one character.

Customizing MEMacs

MEMacs looks for an Emacs_pro file when it is opened to see if there are any commands or local files that it should automatically execute. You can customize the Emacs_pro file by adding commands to it that you use often, command sequences, or text strings. If an Emacs_pro file does not already exist, you can create one.

To create a global file of commands, place the Emacs_pro file in the S: directory. Local files can be put in any directory. If that directory is the current directory when MEMacs is opened, the commands in that particular local file are executed.

When both local and global Emacs_pro files are present, the local file overrides the global file.

For example:

```
Set Case On
Set-Key F11 "Dear Sirs:"
```

```
Set-Key F12 "^S Workbench"  
Set-Key F13 "^X^B"
```

makes the following assignments:

Shift+F1	Enter the text string "Dear Sirs:".
Shift+F2	Search forward for the next occurrence of the word Workbench. (The Set Case On commands make any text searches case-sensitive.)
Shift+F3	Display the list of buffers.

You must use **Ctrl+Q** to enter a **Ctrl+key** sequence. For example, to enter the **^S** character shown in the example, press **Ctrl+Q**, **Ctrl+S**.

Quitting MEmacs

You can exit MEmacs by selecting the Quit menu item in the Project menu or by entering **Ctrl+C**. MEmacs lets you save any modified buffers or quit without saving.

EDIT

EDIT is a line editor designed for the automated editing of files, particularly binary files or files that are larger than available memory. You cannot create a new file with EDIT.

EDIT processes files line by line. As EDIT moves through the input, or source file, each line is passed after alteration to a sequential output file, the destination file.

EDIT processes the lines in files in a forward direction; however, you can move backward a limited number of lines. EDIT holds the lines in an output queue before writing them to the destination file. The size of this queue depends on the amount of memory available. You can increase the size of the queue with the **OPT P** and **OPT W** options.

The format for EDIT is the following:

```
EDIT [FROM] <filename> [[TO] <filename>] [WITH <filename>]
[VER <filename>] [OPT P <lines> | W <chars> |
P<lines>W<chars>] [WIDTH <chars>] [PREVIOUS <lines>]
```

The FROM argument specifies the source file to be edited. You must specify a source file with EDIT, although the FROM keyword is optional.

The TO argument specifies the destination file to which EDIT sends its output, including editing changes. If you omit the TO argument, EDIT uses a temporary file. This temporary file is renamed with the name of the FROM file and overwrites the FROM file when editing is complete.

The WITH keyword specifies a file containing editing commands.

The VER keyword specifies the file to which EDIT sends error messages and line verifications. If the VER argument is not given, EDIT uses the screen.

Use OPT P <n> and OPT W <n> to specify the PREVIOUS and WIDTH options. However, do not use the OPT keyword with PREVIOUS and WIDTH.

You can use the PREVIOUS and WIDTH options to increase or decrease the amount of available memory. The PREVIOUS option sets the number of previous lines available to EDIT to the integer <n>. The WIDTH option sets the maximum number of characters allowed on a line to <n>. EDIT multiplies the number of previous lines by the maximum number of characters (PREVIOUS * WIDTH) to determine the available memory. The default values are PREVIOUS 40 WIDTH 120.

Starting EDIT

Start EDIT through a Shell using the following command:

```
1.> EDIT <filename>
```

Where <filename> is the name of an existing file to be edited.

EDIT Commands

The following list provides background information about EDIT commands:

Current line	Refers to the line that EDIT is working on at any time. Every command entered refers to the current line, all text changes are made to the current line, and new lines are inserted before the current line.
Original lines	The lines of the source file. Lines retain their original number until you renumber them with the REWIND or = commands.
Non-original lines	Any lines that are inserted into the source file or original lines that are split. These are not assigned line numbers.
Line verification	When using commands that change information in a line, EDIT displays the revised line after the command is executed.
Arguments	Strings, qualified strings, numbers, and switch values used with EDIT commands.

Enter commands in one of the following three ways:

- Enter the commands, then press Return
- Enter the final command argument, then press Return
- Enter a semicolon or closing parenthesis

The text conventions used in the command descriptions are the following:

- Command names are shown in upper case, although EDIT is not case-sensitive.
- Angle brackets indicate that information must be substituted. For example, <string> indicates that the command takes a string argument.
- An <n> represents a numeric argument.
- Square brackets indicate that the argument is optional. For example, [<n>] indicates that the command can take an optional numeric argument.

- Slashes are used as delimiters for strings; use one slash between two strings.
- Periods are used as delimiters for file names (slashes cannot be used since they are used to separate strings).

Selecting the Current Line

The following commands let you move through the file and select the current line.

Move to a specific line number	M <n>	Specify a new current line by entering its line number, a period, and an asterisk as M's argument. Only original lines can be accessed by line number.
Move to next line in the source file	N	Move forward one line. Entering a number and N indicates the number of lines to move forward. When used as the last line of the source file, EDIT creates an extra line at the end of the file. If you are already on this extra line, using N causes an error message to be displayed.
Move to the previous line in the source file	P	Moves back one line. Entering P repeatedly moves more than one line. Entering a number + P indicates how many lines to move back. You can only move back to previous lines that have not yet been written to the destination file. The default is 40 lines, which can be changed with the PREVIOUS option.
Find	F<string>	Lets you select a current line by specifying some of its content.
Search Backward	B,F<string>	Looks backward starting from the current line through the source file for a line containing the specified string.

Editing the Current Line

The following commands add new material or replace material on the current line.

Insert <string2> after <string1>	A <string1> <string2>	Inserts <string2> after the first occurrence of <string1>.
Insert <string2> before <string1>	B <string2> <string1>	Inserts <string2> before the first occurrence of <string1>.
Exchange <string2> for <string1>	E <string2> <string1>	Replaces the first occurrence of <string1> with <string2>.

Inserting and Deleting Lines

The following commands insert new material (non-original lines) and delete lines from the source file. You can also insert complete files into the source file.

Insert one or more lines	I [<n>]	<p>If given alone or with a line number, inserts text before the current line.</p> <p>If given with an asterisk, the text is inserted at the end of the file.</p> <p>Indicate the end of the insertion by pressing Return, Z, and Return.</p>
Delete one or more lines	D [<n>]	<p>Deletes the current line if entered with no arguments. Deletes a specific line if entered with a line number. Deletes a set of lines if entered with a range of line numbers; do not use punctuation between the numbers. Deletes everything from the current line through the end of the source file if entered with a period and an asterisk as arguments.</p>
Delete all lines until the specified string is found	D,F<string>	<p>Deletes successive lines from the source file until the line containing the matching string is found. If no argument is specified, it deletes all lines until it finds the last string specified.</p>

Delete existing lines and replace with new text	R [<i><n></i>]	Lets you delete lines and then insert new ones. Entering a line number following R indicates a specific line to replace.
Change the terminator	Z <i><string></i>	Tells EDIT that it has reached the end of any new text that is being inserted. Entering a string after Z changes it from the default.
Show current information about EDIT	S,H,D	Displays saved information values, such as the last string searched for, the last command entered, and the input terminator.
Turn trailing spaces on/off	T,R, +/-	Preserves any blanks that fall at the end of lines.

Editing Line Windows

You can define subsections of the line, called line windows, on which EDIT executes all subsequent commands. In the descriptions of EDIT qualifiers, the beginning of the line always indicates the beginning of the line window.

Whenever EDIT verifies a current line, it indicates the position of the line window by displaying a > character directly beneath the line. EDIT omits the pointer if the line window begins at the start of the line.

The following commands control the position of the character pointer:

>	Moves the pointer one character to the right.
<	Moves the pointer one character to the left.
PR	Resets the pointer to the start of the line.
PA <i><string></i>	Moves the pointer to the first character after the specified string.
<i><string></i>	Moves the pointer to the first character before the specified string.

The following commands change the character at the current pointer, then move the pointer:

- \$** Makes the character at the pointer lower case, then moves the pointer one character to the right.
- %** Makes the character at the pointer upper case, then moves the pointer one character to the right.
- _** The `_` (underscore) command deletes the character at the pointer, turning it into a space, then moves the pointer one character to the right.
- #** Deletes the character at the pointer, then moves the rest of the line one character to the left. To delete several characters, specify a number before the `#`. For example, `5#` deletes the next five characters in the window.

A combination of these commands can be used to edit a line character by character.

The following commands insert and exchange text on the current line, similar to the A, B, and E commands; however, the character pointer is moved on completion.

Insert <string2> after <string1>	A,P <string1> <string2>	Inserts <string2> after the first occurrence of <string1>. The pointer is then positioned after <string2>.
Insert <string2> before <string1>	B,P <string1> <string2>	Inserts <string2> before the first occurrence of <string1>. The pointer is then positioned after <string2>.
Exchange <string1> with <string2>	E,P, <string1> <string2>	Replaces the first occurrence of <string1> with <string2>. The pointer is then positioned after <string2>.
Delete Till After	D,T,A <string>	Deletes all text from the beginning of the line or the character pointer to the end of the specified string.
Delete Till Before	D,T,B <string>	Deletes all text from the beginning of the line or the character pointer; stops before the specified string.
Delete From After	D,F,A <string>	Deletes all text starting after a specified string to the end of the line.
Delete From Before	D,F,B <string>	Deletes all text starting with the specified string to the end of the line.

Splitting and Joining Lines

These commands split a line into more than one line and join together two or more successive lines.

Split line before <string>	S,B <string>	Splits the current line before the specified string. The first part of the line is sent to the output queue; the second part is made into a new non-original current line. Qualifiers can be used to restrict the context of the string.
Split line after <string>	S,A <string>	Splits the current line after the specified string. The first part of the line is sent to the output queue; the remainder of the line becomes the new current line. Qualifiers can be used to restrict the context of the string.
Join two lines	C,L [<string>]	Joins the current line with the next line of the source file. The string argument is optional; however, if a string is specified it is added to the end of the current line and that whole line is joined with the next line in the source file.

Renumbering Lines

These commands renumber the source file's lines to include non-original lines and to update a file that has been edited.

Renumber source lines	= <n>	Sets the current line number to <n>. All subsequent original and non-original lines below <n> are renumbered if you move to them.
Return to the beginning source file	REWIND	Moves back through the source file to make line 1 the current line. EDIT scans the rest of the source file and writes the lines to the destination file. This file is closed and reopened as a new source file. Non-original lines are now recognized as original lines. Can be entered as REWI.

Verifying Lines

These commands describe different ways of verifying lines.

Turn Verification on/off	V + -	Turns line verification on or off. If off, the lines are not displayed on the screen. To turn off, enter V -. To turn on, enter V +.
Verify the current line	?	Verifies the current line by displaying the line number and the contents of the line.
Verify the current line with character indicators	!	Produces two lines of verification. In the first line all nongraphic characters are replaced with the first character of their hexadecimal value. In the second line, a minus sign is displayed under all positions corresponding to upper case letters and the second hexadecimal digit corresponds to nongraphic characters. All other positions contain spaces. In binary files, nongraphic characters are represented with question marks (??).

Inspecting the Source File

These commands advance through the source file, sending the lines it passes to the verification file, as well as to the normal output. They are known as Type commands because they display lines on the screen.

Type <n> lines to the screen	T<n>	Types the specified number of lines to the screen. The first line typed is the current line. Omitting the <n> continues typing to the end of the source file. Interrupt the command with Ctrl+C.
Type the lines in the output queue	T,P	Displays the lines currently held in the output queue.
Type until EDIT has replaced all the lines in the output queue	T,N	Types from the current line forward until all the lines in the output queue are replaced. The previous contents are sent to the destination file.

Type with line numbers	T,L <n>	Similar to the T command. Types a specified number of lines, displaying the line numbers. EDIT displays + + + + for inserted or split lines since they do not have line numbers.
-------------------------------	---------	--

Making Global Changes

These commands start and stop global changes. Global changes take place automatically as EDIT scans the source file in a forward direction. These commands automatically apply an A, B, or E command, as appropriate, to any occurrence of <string1> in a new current line. They also apply to the current line that is in effect when the command is given.

```
GA [qualifier] <string1> <string2>
GB [qualifier] <string1> <string2>
GE [qualifier] <string1> <string2>
```

For example, if you want to change DF0: to DF2: throughout an entire file, enter:

```
GE /DF0:/DF2:/
```

Cancel a global command	CG [<id number>]	Cancels a global command. The identification number set with a GA, GB, or GE command is output to the verification file or the screen if EDIT is interactive. If no argument is specified, all global operations are cancelled. To cancel a specific operation, enter the identification number after the CG command.
Suspend a global command	SG [<id number>]	Suspends a global command. All global operations are suspended if no argument is given. Enter the identification number to suspend a specific operation.
Enable a global command	EG [<id number>]	Resumes global operation that had been suspended with the SG command. Unless a specific identification number is provided, all global commands are resumed.

Show gobal commands	SHG	Displays the current global commands and their identification numbers. Also provides the number of times each global string was matched.
----------------------------	-----	--

Changing Command, Input, and Output Files

These commands change the files set up when you started EDIT from the Shell. These files are:

- the command file — started with the WITH option.
- the input file — the source file specified with FROM.
- the output file — the destination file specified with TO.

Changing the Command File	C <filename>	Reads EDIT commands from a specified file. Delimit the file using a character other than a slash (/) since AmigaDOS uses these characters to separate file names. When all commands in the specified file are executed, the file is closed. You can then enter the commands through the keyboard.
Changing the Input File	FROM <filename>	Reads lines from another source file. EDIT does not close the original source file. Reselect the source file by entering the FROM command without an argument. For examples of the FROM command, see page 4-54.
Closing a File	CF <filename>	Closes the destination file that was originally specified with the TO command. You can then open that file for input. CF can also close a new input file that is open. For examples of the CF command, see page 4-54.

Changing the Output File	TO <filename>	Specifies a different file as the destination file. The TO command writes the existing queue of output lines to the new TO file. The new TO file is used until another file is specified. Reselect the original destination file by using the TO command with no argument. The alternate output file remains open, but unused. For examples of the TO command, see page 4-54.
Stop executing the command file	Q	Stops EDIT from executing the current command file specified with the WITH keyword or with the C command. EDIT reverts to any previous command file. Using Q at the outermost level is equivalent to using the W command.

The FROM, CF, and TO commands are used as follows:

Command	Action
M10	Pass lines 1-9 in the original source file to the output queue.
FROM .XYZ.	Select the XYZ file for new input; line 10 of the original source file remains current.
M6	Pass line 10 from the original file, then pass lines 1-5 from the XYZ file to the output queue. Line 6 of XYZ is the new current line.
FROM	Reselect the original source file.
M14	Pass line 6 from XYZ, then lines 11-13 from the original source file to the output queue. Line 14 of the source file is the new current line.
FROM .XYZ.	Reselect file XYZ. Line 14 of the source file is still the current line.
M*	Pass line 14 of the source file and all remaining lines of file XYZ to the output queue. An extra line is added to the end of file XYZ. That line is the new current line.

Command (cont'd)	Action (cont'd)
FROM	Reselect the original source file. The extra line added to file XYZ is still the current line.
CF .XYZ.	Close file XYZ.
M*	Pass the remaining lines of the source file (lines 15 to the end of the file) to the output queue.
M11	Pass lines 1-10 of the source file to the original destination file.
TO .XYZ.	Make XYZ the new output file.
M21	Pass lines 11-20 to file XYZ.
TO M31	Make the original destination file current, and pass lines 21 to 30 to it.
TO .XYZ.	Make XYZ the current output file.
M41	Pass lines 31 to 40 to XYZ.
TO	Make the original destination file current.
TO .XYZ.	Send the output queue to file XYZ.
1000N	Advance through the next 1000 lines of the source file.
TO	Select the original destination file.
CF .XYZ.	Close the XYZ file.
I2000 .XYZ.	Insert the 1000 lines from the source file that were sent to file XYZ back into the source file above line 2000.

Ending EDIT

These commands exit EDIT.

Exit, saving changes	W	Exits EDIT, saving all changes to the destination file specified by TO. EDIT exits after reaching the end of the source, closing all the files, and relinquishing memory. If you started EDIT without specifying a destination file, it renames the temporary destination with the same name as the original source file, which is renamed .T/Edit-backup. This backup is only available until the next time you run EDIT.
Exit, without saving changes	STOP	Stops EDIT immediately without saving any changes to the source file. Prevents EDIT from overwriting the original source file, ensuring that no changes are made to the original input information.

Chapter 5

Using Scripts

Script files are text files that contain lists of commands for repetitive or complicated tasks or for performing the same operation on multiple files. This chapter includes the following information about script files:

- Understanding scripts
- Scripting characters
- Script commands
- Condition flags
- Debugging script files
- Environment variables

Understanding Scripts

A script is a text file that contains a series of commands. Using scripts is a way to automate complex or repetitive tasks, especially those that you need to perform regularly. A script can perform virtually any operation normally done one command at a time, including working with programs and data files; performing calculations; and operating interactively, accepting and displaying information. Essentially a script is a small program that can be easily edited.

To create a script, follow these steps:

1. Open a text editor, such as ED, that saves files in ASCII format.
2. In the text editor, enter the script commands in the sequence in which they are to be performed.

3. Save the file. The S: directory is usually used for script files, but you can store a script anywhere.

You can then run the script in the Shell, entering the EXECUTE command followed by the full path to the script.

Note You can avoid having to use the EXECUTE command and the full path by setting the script's s protection bit. Enter the command **PROTECT <script> +s**, substituting the path to the script. When the s bit is set, enter only the name of the script to execute it.

It is possible to run a script that has a Workbench project icon by using the ICONX command as the default tool. For more information on the ICONX command, see Chapter 6.

Kinds of Scripts

There are several kinds of scripts that you can use on the Amiga, including scripts of AmigaDOS commands, scripts of ARexx commands, and scripts of ED commands. For details on writing ARexx scripts, see the *ARexx User's Guide*. See the ED section of Chapter 4 for information on ED command scripts.

When to Use ARexx

You can create both AmigaDOS and ARexx scripts on the Amiga. These scripts are not mutually exclusive; under AmigaDOS Release 2 and beyond, ARexx commands can appear in AmigaDOS scripts and vice versa. However, different tasks are appropriate to each.

Unlike ARexx, AmigaDOS is not a general purpose programming language. AmigaDOS is oriented toward basic file management and system configuration tasks. ARexx is meant to be used for accomplishing tasks more complex than simple branching and conditional execution.

Simple Scripts

A script can be as simple as a series of paths, telling the Amiga to run certain programs. A User-startup file is a good example of a simple script. You can add various configuration commands, such as an `ADDBUFFERS` statement, to such a script without worrying about affecting program flow or using error checking. For more information about the User-startup file, see Appendix D.

Automatic Scripts

You can automatically generate scripts using the `LIST` command. `LIST` has an `LFORMAT` option that allows you to modify its output to include any text you wish along with the usual `LIST` output. This text can be a command and command keywords, with the file name `LISTed` appearing as an argument to the command. If you list the contents of a directory this way, redirecting the output to a file, you have a ready-made script that applies a command to the contents of that directory.

When an operation on multiple files is too involved to be done with a single pattern matching command, use an automatically generated script to execute the command for each of the files. For example, an operation that renames a series of files, giving the current file names the same extension, cannot be done with a single pattern-matching `RENAME`. See the `LIST` section in Chapter 6 for information on `LIST`'s `LFORMAT` option and the examples in Chapter 8 for an illustration of this technique.

Special Script Characters

The semicolon, back apostrophe, dollar, double dollar, and question mark are special characters that are used primarily in scripts to add comments, execute commands from within strings, introduce environment variables, reference the current Shell number, and accept input redirection.

Semicolon (;)

Semicolons add comments to command lines. All characters to the right of a semicolon are ignored, allowing you to place descriptive comments on the same line with AmigaDOS commands. For example:

```
ASSIGN T: RAM:t      ;set up T: directory for
                    ;scripts
```

Comments can continue onto additional lines if they are too long to fit on one line. New lines must begin with a semicolon and should be indented to the same level as the previous comment for clarity.

Back Apostrophe (')

Back apostrophes are used to execute commands from within a string. If a string containing a command enclosed in back apostrophes is printed, the enclosed command is executed. For example:

```
1> ECHO "The date and time are: `date`"
```

prints "The date and time are: " followed by the output of the DATE command. When a command such as DIR that produces multiple lines of output is embedded in an ECHO statement, the output is not properly formatted; all of it appears on one line.

Note Commands that refer to the current directory do not work correctly when invoked from within a string with the back apostrophe. Using the back apostrophe automatically sets up a temporary sub-shell for that command only. References to a current directory access the sub-shell's directory.

Dollar (\$)

The dollar sign is used in two ways: as an operator that introduces an environment variable (which also works outside of a script) and in a bracketed statement to separate a variable value from a default value.

For example, with an environment variable:

```
1> ECHO "Current Kickstart version: $Kickstart"
Current Kickstart version: 39.106
```

As a default separator in a script:

```
COPY foo.library TO <LIBS:$userlibdir>
```

You can change the character that provides this function from the dollar sign to something else with the `.DOLLAR` scripting keyword.

Double Dollar (<\$\$>)

A bracketed double dollar sign (<\$\$>) substitutes the current process number. You can reference the current Shell number by the double dollar sign character string <\$\$> including brackets since it always returns the current process number as a string. When you create temporary files in a multitasking environment, it is essential for these files to have unique names so that processes do not interfere with each other. Adding the <\$\$> string to file names creates unique names for temporary files, logical assignments, and PIPEs. A `.KEY` statement is required in any script that uses <\$\$>. `.KEY` is described on page 5-7. To avoid conflict with the redirection arguments, <\$\$>'s angle brackets can be redefined using the `.BRA` and `.KET` commands. `.BRA` and `.KET` are described on page 5-8.

Question Mark (?)

The question mark, when used as a separate argument in a command, instructs the command to accept input redirection.

Script Commands

Any AmigaDOS command can be used in a script, however, there are some commands that are used only in scripts. You can do the following with script commands:

- Parameter substitution
- Enter arguments on the command line
- I/O redirection
- Specify default strings
- Enter comments
- Nest commands within scripts
- Create interactive scripts

- Create scripts that repeat commands

Script-Specific Commands

The following commands are typically used only with scripts:

Script Command	Meaning
ASK	Asks for user input.
ECHO	Prints a string.
ELSE	Allows an alternative in a conditional block.
ENDIF	Terminates an IF block.
ENDSKIP	Terminates a SKIP block.
EXECUTE	Executes a script with optional argument substitution.
FAILAT	Sets the failure condition of the script.
IF	Handles conditional operations.
LAB	Specifies a label; used with SKIP.
QUIT	Specifies a return code and exits a script.
REQUESTCHOICE	Allows AmigaDOS and ARexx scripts to use system requesters to ask the user for feedback.
REQUESTFILE	Allows AmigaDOS and ARexx scripts to use the system file requester.
SKIP	Skips the execution of the script ahead or backwards to the specified label.
WAIT	Waits for the specified time.

Dot Commands

Dot commands are keywords beginning with a period that are used only in scripts. Special dot command lines included in your script can specify places for parameter substitution. Enter these parameters as arguments to the EXECUTE command. The following table lists the dot commands.

Dot Command	Meaning
.KEY	Argument template used to specify the format of arguments; can be abbreviated to .K . Separate arguments for .KEY with commas. Do not use spaces. See page 5-7 for more information on using .KEY .
.DOT <ch>	Change dot character from . to <ch> , where <ch> is the character that is substituted.
.BRA <ch>	Change opening bracket character from < to <ch> .
.KET <ch>	Change closing bracket character from > to <ch> .
.DOLLAR <ch>	Change default character from \$ to <ch> ; can be abbreviated to .DOL .
.DEF <keyword> <value>	Give default to parameter.
.<space>	Comment line. Be sure to include the space following the dot to avoid producing an error. Note that the preferred method for entering comments is with a semicolon (;). Blank comment line. Be sure there is nothing else on the line with the dot to avoid producing an error.

When you EXECUTE a command line, AmigaDOS looks at the first line of the script. If it starts with a dot command, AmigaDOS scans the script looking for the parameter substitutions described above and builds a temporary file in the T: directory. If the file does not start with a dot command, AmigaDOS assumes that no parameter substitution is necessary and starts executing the file immediately without copying it to T:. Because dot commands require extra disk accesses and increase execution time, do not use them if you do not need parameter substitution.

Allowing Arguments

The **.KEY** (or **.K**) keyword specifies both keyword names and positions in the command line. It tells EXECUTE the number of parameters and how to interpret them. Only one **.KEY** statement is allowed per

script; if present, it should be the first line in the file. Any script containing <\$\$> must also contain a .KEY statement.

The arguments on the .KEY line can be given with the /A and /K directives, which work the same as in an AmigaDOS template. (Templates are described in Chapter 6.) Arguments followed by /A are required; arguments followed by /K require the name of that argument as a keyword. Use commas to separate multiple arguments in the .KEY line, not spaces.

For example, if a script starts with .KEY filename/A it indicates that a file name must be given on the EXECUTE command line after the name of the script. This file name is substituted in subsequent lines of the script. For example, if you have a script called Newtext and the first line is:

```
.KEY filename/A,TO/K
```

you must specify a file name variable. The TO variable is optional, but, if specified, the TO keyword must be used. The following is an acceptable way to run Newtext:

```
1> EXECUTE Newtext Textfile TO NewFile
```

Substitution

Before execution, AmigaDOS scans the script for any items enclosed by .BRA and .KET characters (< and > by default). Such items can consist of a keyword or a keyword and a default value. EXECUTE tries to substitute a parameter when it finds a keyword enclosed in angle brackets. However, if you want to use a string in your script file that contains angle brackets or if your script uses angle brackets for redirection of input/output, you must define substitute bracket characters with the .BRA and .KET commands. .BRA <ch> changes the opening bracket character to <ch>, while .KET <ch> changes the closing bracket character to <ch>.

For example, the following script called Demo uses these lines:

```
.KEY filename
ECHO "This line does NOT print <angle> brackets."
.BRA {
.KET }
ECHO "This line DOES print <angle> brackets."
ECHO "The specified filename is {filename}."
```

```
1> EXECUTE Demo TestFile
```

which results in the following output:

```
This line does NOT print brackets.  
This line DOES print <angle> brackets.  
The specified filename is TestFile.
```

The first ECHO statement causes AmigaDOS to look for a variable to substitute for the <angle> parameter. Since no argument named "angle" is given on the EXECUTE command line, the null string is substituted. The .BRA and .KET commands tell the script to use braces instead of angle brackets to enclose parameters. When the second ECHO statement is executed, the angle brackets—with their special meaning removed—are printed as normal text. The third ECHO statement illustrates that the braces now function as the bracket characters.

Redefine <\$\$>'s angle brackets with .BRA and .KET to avoid redirection conflicts.

Defaults

When enclosing a keyword in bracket characters, you can also specify a default string to be used if a variable is not supplied on the command line. Defaults can be specified in two ways: every time you reference a parameter or using the .DEF command.

If you specify the default every time you reference a parameter, you must separate the two strings with a dollar sign (\$).

For example, in the following statement:

```
ECHO "<word1$defword1> is the default for Word1."
```

defword1 is the default value specified for word1. It is printed if no other variable is given for word1. However, if you want to specify this default several times in your script, you must use <word1\$defword1> each time.

Defining a default using the .DEF command allows you to specify a default for each specific keyword. For example:

```
.DEF word1 "defword1"
```

assigns `defword1` as the default for the `word1` parameter throughout the script. The following statement:

```
ECHO "<word1> is the default for Word1."
```

results in the same output as the previous ECHO statement:

```
defword1 is the default for Word1.
```

The `.DOLLAR <ch>` command allows you to change the default character from `$` to `<ch>`. (You can also use `.DOL <ch>`.) For example:

```
.DOL #  
ECHO "<word1#defword1> is the default for Word1."
```

Comments

You can embed comments in a script by including them after a semicolon (;) or by entering a dot (.), followed by a space, then the comment. Blank lines are accepted and ignored within scripts.

Note We recommend that you use the semicolon method for entering comments to avoid errors caused by omitting the space following the dot.

Nesting Commands

AmigaDOS provides a number of commands that can be used in scripts, such as `IF`, `ELSE`, `SKIP`, `LAB`, and `QUIT`. These commands, as well as the `EXECUTE` command, can be nested in a script.

To stop the execution of the current script, press `Ctrl+D`. If you have nested script files, you can stop the set of `EXECUTE` commands by pressing `Ctrl+C`.

Example 1:

Assume the script `Printit` contains the following:

```
.KEY filename  
RUN COPY <filename> TO PRT: +  
ECHO "Printing of <filename> done"
```

The following command:

```
1> EXECUTE Printit Test/Prg
```

responds as though you entered the following commands at the keyboard:

```
1> RUN COPY Test/Prg TO PRT: +  
ECHO "Printing of Test/Prg done"
```

Note the use of the plus sign at the end of the first line. If you press Return after a plus sign, the RUN command also executes the command in the second line when the first command is finished.

Example 2:

Another example, Show, uses more of the features described above:

```
.KEY name/A  
IF EXISTS <name>  
  TYPE <name> NUMBER ;if file is in the given directory  
                        ;type it with line numbers  
ELSE  
  ECHO "<name> is not in this directory"  
ENDIF
```

The command:

```
1> EXECUTE Show Work/Docfile
```

displays the Work/Docfile file, with line numbers on the screen, if Show exists in the current directory. If the file is not there, the screen displays an error message. The /A requires a file name be given on the command line after Show or an error occurs.

Interactive Script Files

You can create scripts that pause to request information from the user before continuing. The REQUESTCHOICE or REQUESTFILE commands for creating standard Amiga requesters let you use the familiar Workbench approach to get a user's response. This can accommodate variable conditions with a single script. For example, the following script copies six files from a hard drive to a floppy disk. When the six files are copied, the script asks for confirmation to continue the copy, allowing time to insert a new floppy disk into the disk drive if required.

```
COPY 2k.eps DF0:
COPY 2m.eps DF0:
COPY 2n.eps DF0:
COPY 2o.eps DF0:
COPY 2t.eps DF0:
COPY 2v.eps DF0:
ECHO "Chapter 2 files copied."
ASK "Continue Copy?"
IF WARN
    COPY 3a.eps DF0:
    COPY 3c.eps DF0:
    COPY 3g.eps DF0:
    COPY 3aa.eps DF0:
    COPY 3bb.eps DF0:
    COPY 3ff.eps DF0:
ENDIF
```

At the "Continue Copy?" prompt, press Y to copy the remaining files to a disk in DF0:. Press N or Return to terminate the copy process.

Repeating Commands

You can create scripts that repeat the same command, substituting a different file name in each command line. For example, to rename eight files in one operation you could create the following script:

```
RENAME section1 chap1.1
RENAME section2 chap1.2
RENAME section3 chap1.3
RENAME section4 chap1.4
RENAME section5 chap1.5
RENAME section6 chap1.6
RENAME section7 chap1.7
RENAME section8 chap1.8
```

This example assumes that the files are in the Shell's current directory. If not, you must specify the complete path to each file.

Scripts such as this can often be generated automatically by using the LFORMAT option of the LIST command.

Ending a Script

In general, scripts end when they have completed all the specified commands. Return codes report whether executed commands

succeeded or failed. You can include a QUIT command in a script to end it, but unless there is a specific condition under which you wish the script to end, including QUIT is not necessary. You can stop a script by pressing Ctrl+D in the window in which it is running.

Condition Flags

Condition flags indicate the condition upon which a particular command stops running. When commands are executed, return codes report if they succeeded or failed. The standard return codes are:

- 0** The command succeeded.
- 5** Represents a caution, usually indicating that some type of error occurred. The error, however, was not serious enough to abort the command. If the command is part of a script, subsequent commands are executed. Several commands set the condition flag to WARN to specify a non-error command outcome.
- 10** Represents an error. A return code of 10 aborts a script, unless a higher limit has been set with the FAILAT command.
- 20** Represents a failure.

Other values may be returned by applications. In such cases, the previously listed values are considered lower limits of the specified condition as follows:

0-4	No Error
5-9	Warn
10-19	Error
20 or above	Failure

Some commands, such as ASK and SEARCH, use the WARN flag to signal certain conditions for testing in scripts.

For example, in the COPY script on page 5-12, the ASK command requests confirmation to continue the copy:

```
ASK "Continue Copy?"  
IF WARN  
    COPY 3a.eps DF0:
```

Pressing Y sets the condition flag to 5 (WARN), executing the IF block. Pressing N or Return sets the condition flag to 0 (NO ERROR), aborting the script because the IF statement did not receive the specified return code.

Debugging Script Files

If a script command fails, you may see one of the following error messages:

```
Unknown command <command>
```

This occurs when you have entered a command that is unrecognizable.

```
<command> failed returncode 20
```

This occurs when a valid command's arguments are entered incorrectly. Enter WHY at the prompt after the error appears for more information about the error.

If an error appears, use your text editor (ED, MEmacs, or a word processor that can save ASCII files) to correct the line containing that command.

Inserting the line **SET ECHO ON** within the script assists in locating errors. This causes all subsequent command lines to be ECHOed to the screen as they are executed. Error messages are printed after the system tries to execute the incorrect command. To disable SET ECHO, enter SET ECHO OFF or delete the SET ECHO ON line.

Using Environment Variables

Environment variables are used in scripts to hold status and string information. Variables can be substituted for strings that are long and tedious to enter. Changing the value of the variable is more convenient than re-editing the script when the value for the string changes.

Environment variables are maintained by AmigaDOS rather than individual applications. These variables can be accessed and used by

different programs or scripts. When a variable name preceded by a dollar sign (\$) is encountered in a script, the variable name is replaced by the value assigned to the variable. The line is then executed as if you had originally entered the value.

For example, AmigaDOS maintains the variables `Workbench` and `Kickstart` that track the current version numbers of your `Workbench` and `Kickstart` software. Running the following line prints the `Workbench` version number:

```
ECHO "Amiga Workbench Disk. Release Version
$Workbench"
```

Some variables, such as the `Workbench` and `Kickstart` variables, have already been created. The Shell responds to the `ECHO` variable and maintains the `PROCESS`, `RC`, and `RESULT2` local variables automatically. These are explained as follows:

ECHO	When the value of this variable is <code>ON</code> , commands are echoed to the screen when they are executed. When it is <code>OFF</code> (the default), commands are not echoed.
PROCESS	Holds the process (CLI) number.
RC	Holds the condition flag return code of the last command executed (0, 5, 10, or 20). This is often used in scripts.
RESULT2	Holds the secondary return code, or error number, that explains why a command failed.

For example, if you include the `SET ECHO ON` command at the beginning of a script, each line of the script is echoed to the screen as it is executed.

When an environment variable is given a numeric value, it can be used in calculations and expressions. For example, if you assign the value 9 to a variable called `nine`, use `$nine` in `EVAL` expressions. For example:

```
1> SETENV nine 9
1> EVAL 5 * $nine
45
```

`EVAL` is an AmigaDOS command that evaluates integer and Boolean expressions. However, it does not work on environment variables that have a fractional numeric value; be sure to use whole numbers when using `EVAL`.

Creating Environment Variables

Environment variables can be created with the SET and SETENV commands.

SET

SET creates local variables, which are recognized only by the Shell in which they are created and any Shells created by that original Shell. For example, if you are creating an environment variable in your Shell window, then execute the NEWSHELL command through the Execute Command menu item, the new Shell does not recognize any of the variables created in your original Shell. However, if you open a second Shell by entering the NEWSHELL command in your original Shell, the new Shell recognizes any variables created in its parent Shell.

Using the GET command displays the value associated with a variable; using the UNSET command removes variables.

SETENV

SETENV creates global variables recognized by all Shells. Global variables are stored as small ASCII files in the ENV: directory. GETENV displays the value associated with global variables and UNSETENV removes global variables. Use global variables only when certain values must be available to other processes.

Some applications use environment variables. For example, the MORE program supports an Editor environment variable. You can use SETENV to specify MEMacs as your editor of choice:

```
1> SETENV Editor Extras:Tools/MEMacs
```

Be sure to specify the complete path to MEMacs.

If you use MORE to view the contents of the User-startup file, pressing Shift+E automatically transfers to a MEMacs screen with the User-startup loaded and ready for editing.

Chapter 6

AmigaDOS Command Reference

The commands in this chapter are executed from the Shell window. They are described in alphabetic order; however, some commands reserved for system use appear together at the end of the chapter.

The following table provides a complete alphabetical reference to all of the commands in this chapter, their purpose, and the page on which they appear:

Command	Purpose	Page
ADDBUFFERS	Instructs the file system to add or display cache buffers for a drive.	6-10
ADDDATATYPES	Builds a list of data types (system command).	6-92
ALIAS	Sets or displays command aliases.	6-11
ASK	Gets yes/no user input during script file execution.	6-12
ASSIGN	Controls assignment of logical devices.	6-13
AVAIL	Reports the amount of memory available.	6-17
BINDDRIVERS	Activates device drivers in Expansion drawer (system command).	6-92
BREAK	Sets attention flags in the specified process.	6-18
CD	Sets or displays the current directory.	6-19
CHANGETASKPRI	Changes priority of Shell processes.	6-21

Command (cont'd)	Purpose (cont'd)	Page (cont'd)
CONCLIP	Moves data between the console windows and the clipboard (system command).	6-93
COPY	Copies files or directories.	6-22
CPU	Sets or displays processor options.	6-24
DATE	Sets or displays the system date and time.	6-26
DELETE	Deletes files or directories.	6-28
DIR	Displays a sorted list of the files in a directory.	6-29
DISKCHANGE	Informs the Amiga that the disk in a disk drive has changed.	6-32
ECHO	Displays a string.	6-32
ED	Edits text files.	6-33
EDIT	Edits text files (line editor).	6-34
ELSE	Specifies an alternative for an IF statement in a script file.	6-34
ENDCLI	Ends a Shell process.	6-35
ENDIF	Terminates an IF block in a script file.	6-35
ENDSHELL	Ends a Shell process.	6-36
ENDSKIP	Terminates a SKIP block in a script file.	6-36
EVAL	Evaluates integer or Boolean expressions.	6-37
EXECUTE	Executes a script with optional argument substitution.	6-38
FAILAT	Instructs a command sequence not to fail unless a given value is returned.	6-39
FAULT	Prints messages for the specified error numbers.	6-40
FILENOTE	Attaches a comment to a file.	6-41
GET	Gets the value of a local variable.	6-42
GETENV	Gets the value of a global variable.	6-43

Command (cont'd)	Purpose (cont'd)	Page (cont'd)
ICONX	Allows execution of a script file from an icon.	6-43
IF	Evaluates conditional operations in script files.	6-45
INFO	Gives information about mounted devices.	6-46
INSTALL	Writes or checks a disk boot block.	6-47
IPREFS	Monitors system Preferences (system command).	6-93
JOIN	Concatenates two or more files into a new file.	6-48
LAB	Specifies a label in a script file.	6-48
LIST	Lists specified information about directories and files.	6-49
LOADRESOURCE	Preloads resources into memory to avoid excessive disk swaps.	6-52
LOADWB	Starts Workbench.	6-53
LOCK	Sets the write-protect status of a device.	6-54
MAGTAPE	Rewinds or skips forward SCSI tapes.	6-55
MAKEDIR	Creates a new directory.	6-55
MAKELINK	Creates a link between file names.	6-56
MOUNT	Makes a device connected to the system available.	6-57
NEWCLI	Opens a new Shell window.	6-58
NEWSHELL	Opens a new Shell window.	6-59
PATH	Controls the directory list that the Shell searches to find commands.	6-62
PROMPT	Changes the prompt string of the current Shell.	6-63
PROTECT	Changes the protection bits of a file or directory.	6-64
QUIT	Exits from a script file with a specified return code.	6-66

Command (cont'd)	Purpose (cont'd)	Page (cont'd)
RELABEL	Changes the volume name of the disk in the given drive to the specified name.	6-67
REMRAD	Removes the recoverable RAM disk RAD:.	6-68
RENAME	Changes the name of or moves a file or directory.	6-68
REQUESTCHOICE	Allows AmigaDOS and ARexx scripts to use custom requesters.	6-69
REQUESTFILE	Allows AmigaDOS and ARexx scripts to use a file requester.	6-70
RESIDENT	Displays and modifies the list of resident commands.	6-72
RUN	Executes commands as background processes.	6-74
SEARCH	Looks for the specified text string in the files of the specified directories.	6-76
SET	Sets a local variable.	6-77
SETCLOCK	Sets or reads the battery backed-up hardware clock.	6-78
SETDATE	Changes the timestamp of a file or directory.	6-79
SETENV	Sets a global variable.	6-79
SETFONT	Sets the font for the Shell.	6-80
SETKEYBOARD	Sets the keymap for the Shell.	6-81
SETPATCH	Makes ROM patches in system software (system command).	6-94
SKIP	Skips to a label when executing script files.	6-82
SORT	Alphabetically sorts the lines of a file.	6-84
STACK	Displays or sets the stack size within the current Shell.	6-85
STATUS	Lists information about Shell processes.	6-85

Command (cont'd)	Purpose (cont'd)	Page (cont'd)
TYPE	Displays the contents of a file.	6-86
UNALIAS	Removes an alias.	6-87
UNSET	Removes a local variable.	6-87
UNSETENV	Removes a global variable.	6-88
VERSION	Displays software version and revision numbers.	6-88
WAIT	Waits for the specified time.	6-89
WHICH	Searches the command path for a particular item.	6-90
WHY	Prints an error message that explains why the previous command failed.	6-91

Command Documentation

Each command documented in this manual is shown with the format, arguments, options, symbols, and abbreviations required for proper use.

This chapter and Chapter 7 provide command specifications for the AmigaDOS commands and the Workbench programs accessible through the Shell using the following standard outline:

Format	All the arguments and options accepted by a command. The special characters that indicate the particular type of argument are described on page 6-6.
Template	An optional on-line reminder of the command's format that is embedded in the program's code. Entering a command followed by a space and a question mark (for example, DIR ?) displays the template. A complete description of the template notation is found on page 6-8.
Location	The directory where the command is normally stored.

Examples A sample use of the command. Examples are displayed in the Courier typeface to distinguish them from normal text. The 1> represents the Shell prompt; do not type it as part of the example command. Lines in the example not prefaced by 1> represent the output of a command. Command names and keywords are shown in all upper case letters and file and directory names usually have the first letter in upper case; however, they do not need to be entered that way. Press Return to execute the command line.

Separate commands and arguments with spaces. Use punctuation only when required in the syntax of specific commands.

Format

The following lists the characters that indicate the type of argument shown in format listings. Do not use these characters as part of the command.

- < > Angle brackets indicate where additional information, such as a file name, must be included. This argument is required if it is not surrounded by square brackets. (For example, [<filename>]; see below.)
- [] Square brackets enclose optional arguments and keywords. Although not required, these arguments and keywords are accepted by the command.
- { } Braces enclose items that can be given once or repeated any number of times. For example, {<args>} indicates that several items can be given for this argument.
- | Vertical bars separate lists of options from which you can choose only one. For example, [OPT RISIRS] indicates a choice of the R option, the S option, or both options.
- <n> A numeric value is expected by the argument.
- KEYWORD** Italics indicate that the argument's keyword is required if you include that argument.

An ellipsis (...) after a string argument indicates that the string must be the final argument on the command line. Including a comment is not allowed. The remainder of the command line is taken as the desired string. Quotation marks are not needed around the string, even if it contains spaces. If you enter quotation marks, they are part of the string. If you specify the keyword, you can put leading and trailing spaces in the string.

command line indentation On command lines that are long enough to wrap to the next line, this manual shows the wrapped lines as indented for documentation purposes only. In practice, the wrapped lines align with the first character of the Shell prompt.

The format for the COPY command illustrates the use of these conventions:

```
COPY [FROM] {<name | pattern>} [TO] <name | pattern> [ALL]
      [QUIET] [BUF/BUFFER=<n>] [CLONE] [DATES] [NOPRO]
      [COM] [NOREQ]
```

The [FROM] keyword is optional. If it is not specified, the command reads the file name or pattern to copy by its position on the command line.

The {<name | pattern>} argument must be provided. You must substitute either a file name or pattern. The braces indicate that more than one name or pattern can be given.

The [TO] keyword is optional. If it is not specified, the command reads the file name or device to copy to by its position on the command line.

The <name | pattern> argument must be provided. You can specify only one destination.

The [ALL], [QUIET], [CLONE], [DATES], [NOPRO], [COM], and [NOREQ] arguments are optional.

The [BUF | BUFFER=<n>] argument is optional. If given, the keyword is required, but you can use either BUF or BUFFER with the numerical argument. For example, both BUF=5 and BUFFER=5 are acceptable. The numerical argument can also be entered without the equals sign; spaces are optional.

Template

The Template is built into the system to serve as an on-line reminder of a command's syntax and to let you run the command from the Template line by providing a prompt at which you enter the command's arguments.

Display the Template by entering a question mark (?) after a command. The Shell assumes that you wish to run the command and it expects you to enter the command's arguments after the colon following the display. For example:

```
1> TYPE ?
FROM/A/M, TO/K, OPT/K, HEX/S, NUMBER/S:
```

Pressing Return executes the command if it does not require any arguments to run properly. Entering the arguments and their respective keywords and then pressing Return also executes the command. If a command requires arguments and you do not supply them or if you enter anything other than the required arguments, pressing Return results in a non-fatal error message. Remember that you do not need to enter the entire format for a command at this prompt, just the required arguments.

The Templates are listed with the arguments separated by commas, followed by a slash (/), and a capital letter indicating the type of argument. These slash/letter combinations are displayed to remind you of the command's particular requirements and are not entered as part of the command. The following table explains the notation:

Template Notation	Format Equivalent	Meaning
argument/A	<name>	The argument is always required.
option/K	KEYWORD	The option's keyword is required if the argument is given.
option/S	[KEYWORD]	The option works as a switch. The name of the option must be entered to specify it. Most options are switches.
value/N	<n>	The argument is numeric.

Template Notation (cont'd)	Format Equivalent (cont'd)	Meaning (cont'd)
argument/M	{<name>}	Multiple items are accepted for this argument. Although there is no limit to the number of possible arguments, they must be provided before the next argument or option.
string/F	argument...	The string must be the final argument on the command line; the remainder of the command line is taken as the desired string.
=	KYWD KEYWORD	Two different forms of the keyword are equivalent and either are accepted. The equals sign is not entered as part of the command.

The Template for the COPY command illustrates the use of arguments:

FROM/M, TO/A, ALL/S, QUIET/S, BUF=BUFFER/K/N,
CLONE/S, DATES/S, NOPRO/S, COM/S, NOREQ/S

FROM/M indicates that the argument is required and more than one argument is acceptable.

TO/A indicates that the argument is required.

ALL/S, QUIET/S, CLONE/S, DATES/S, NOPRO/S, COM/S, and NOREQ/S indicate that the keywords act as switches. If the keyword is present in the line, the option is used.

BUF=BUFFER/K/N indicates that the BUF or BUFFER keyword (/K) is required to specify this numerical (/N) argument. Both BUF and BUFFER are acceptable keywords (=).

Keywords and their arguments can be linked with an equals sign (=) to ensure correct assignments in complex cases. For example, BUF=20.

Command Listing

ADDBUFFERS

Instructs the file system to add or display cache buffers for a drive.

Format ADDBUFFERS <drive> [<n>]

Template DRIVE/A,BUFFERS/N

Location C:

ADDBUFFERS adds <n> buffers to the list of buffers available for <drive>. Although adding buffers speeds disk access, each additional buffer reduces free memory by approximately 512 bytes. The default buffer allocation is 5 for floppy drives and 30 for hard disk partitions.

The amount of extra available memory dictates the number of buffers you can add. There is no fixed upper limit; however, adding too many buffers reduces overall system performance by taking RAM away from other system functions. Specifying a negative number subtracts that many buffers from the current allocation. The minimum number of buffers is one; however, using only one is not recommended.

Twenty buffers are recommended for a floppy drive in a 512 KB system. Use the default value recommended by the HDTtoolbox program for hard disks. (Display this value by selecting the Advanced Options gadget on the Partitioning screen.)

If only the <drive> argument is specified, ADDBUFFERS displays the number of buffers currently allocated for that drive.

Example:

```
1> ADDBUFFERS DF0:
DF0: has 5 buffers
```

A further example of ADDBUFFERS appears in Chapter 8.

ALIAS

Sets or displays command aliases.

Format ALIAS [<name>] [<string...>]

Template NAME,STRING/F

Location Internal

ALIAS creates aliases, or alternative names, for AmigaDOS commands. ALIAS can be used to abbreviate frequently used commands or replace standard command names with different names.

When AmigaDOS encounters <name>, it replaces it with the defined <string>, integrates the result with the rest of the command line, and attempts to interpret and execute the resulting line as an AmigaDOS command. <Name> is the alias for the command and <string> is the command to be substituted for the alias.

An alias must be entered at the beginning of the command line. You can enter arguments after the alias, but you cannot create an alias to represent a series of command arguments. For example, in the following command line:

```
1> NEWSHELL WINDOW=CON:0/250/640/150/2SHELL/CLOSE
```

the WINDOW argument cannot be replaced with an alias.

You can substitute a file name or other instruction within an alias by placing square brackets ([]) with nothing between them in the <string>. Any argument entered after the alias is inserted at the brackets.

ALIAS <name> displays the <string> for that alias. Entering ALIAS alone lists all current aliases.

Aliases are local to the Shell in which they are defined. If you create another Shell with the NEWSHELL command, it shares the same aliases as its parent Shell. However, if you create another Shell with the Execute Command menu item, it does not recognize aliases created in your original Shell. A global alias that is recognized by all Shells can be created by inserting the alias in the Shell-startup file.

To remove an ALIAS, use the UNALIAS command.

Example 1:

```
1> ALIAS d1 DIR DF1:
```

Entering `d1` displays a directory of the contents of the disk in `DF1:`, as if you entered `DIR DF1:`.

Example 2:

```
1> ALIAS hex TYPE [] HEX
```

creates an alias called `HEX` that displays the contents of a specified file in hexadecimal format. The empty brackets indicate where the file name is inserted in this example. Entering:

```
1> hex Myfile
```

displays the contents of `Myfile` in hexadecimal format.

See also: `UNALIAS`. Further examples of using `ALIAS` appear in Chapter 8.

ASK

Gets yes or no user input during script file execution.

Format ASK <prompt>

Template PROMPT/A

Location Internal

`ASK` is used in scripts to write the string specified by <prompt> to the current window and then wait for keyboard input. Valid keyboard responses are `Y` (yes), `N` (no), and `Return` (no). Selecting `Y` sets the condition flag to 5 (`WARN`). Selecting `N` or pressing `Return` sets the condition flag to 0. Check the response using an `IF` statement.

If the <prompt> contains spaces, it must be enclosed in quotation marks.

Example:

Assume a script contained the following commands:

```
ASK Continue?
IF WARN
    ECHO Yes
```

```
ELSE
    ECHO No
ENDIF
```

At the ASK command, **Continue?** is displayed on the screen. If Y is pressed, **Yes** is displayed on the screen. If N or a Return alone is pressed, **No** is displayed.

See also: IF, ELSE, ENDIF, REQUESTCHOICE, WARN

ASSIGN

Controls assignment of logical device names to files or directories.

Format ASSIGN [<name>:] [{<target>}] [LIST] [EXISTS]
 [DISMOUNT] [DEFER] [PATH] [ADD] [REMOVE]
 [VOLS] [DIRS] [DEVICES]

Template NAME,TARGET/M,LIST/S,EXISTS/S,
 DISMOUNT/S,DEFER/S,PATH/S,ADD/S,
 REMOVE/S,VOLS/S,DIRS/S,DEVICES/S

Location C:

ASSIGN allows references to files or directories with short, convenient logical device names, rather than their usual names or complete paths. The ASSIGN command can create assignments, remove assignments, or list some or all current assignments.

If the <name> and {<target>} arguments are given, ASSIGN assigns the given logical name to the specified target. Each time the assigned logical device name is referred to, AmigaDOS accesses the specified target. If the <name> given is already assigned to a file or directory, the new target replaces the previous one. A colon must be included after the <name> argument.

If only the <name> argument is given, any existing ASSIGN of a file or directory to that logical device name is cancelled.

You can assign several logical device names to the same target by using multiple ASSIGN commands.

You can assign one logical device name to several targets by specifying each file or directory after the <name> argument or by using several ASSIGN commands with the ADD option. Specifying

the ADD option does not replace any existing target assigned to <name>. This target is added to the ASSIGN list and the system searches for all the targets when <name> is encountered. If the first target is not available, ASSIGN uses the next target added.

The REMOVE option deletes a target name from the ASSIGN list.

If no arguments are given with ASSIGN or if the LIST keyword is used, a list of all current assignments is displayed. If the VOLS, DIRS, or DEVICES switch is specified, ASSIGN limits the display to volumes, directories, or devices.

When the EXISTS keyword is entered with a logical device name, AmigaDOS searches the ASSIGN list for that name and displays the volume and directory assigned to that device. If the device name is not found, the condition flag is set to 5 (WARN).

When the {<target>} argument is given, AmigaDOS immediately looks for that file or directory. If the ASSIGN commands are part of the User-startup, the targets must be present on a mounted disk during the boot procedure. If an assigned target cannot be found, a requester asks for the volume containing it. However, using the DEFER and PATH options make the system wait until the target is needed before searching for it.

Note The assigned name does not have to retain the name of the file or directory and it does not have to be in upper case. For example, the name CLIPS: or Clips: can be assigned to the Ram Disk:Clipboards directory.

The DEFER option creates a late-binding ASSIGN. This ASSIGN takes effect when the assigned object is first referenced, rather than when the assignment is made. When the DEFER option is used, the disk containing the assigned target is not needed until the object is called. The assignment then remains valid until explicitly changed.

If you ASSIGN FONTS: to DF0:Fonts with the DEFER option, the system associates FONTS: with the disk that is in DF0: when FONTS: is referred to. For example, if you have a Workbench disk in DF0: at the time the FONTS: directory is needed, the system associates FONTS: with that particular Workbench disk. If you later remove that Workbench disk and insert another disk containing a

Fonts directory, the system specifically requests the original Workbench disk the next time FONTS: is needed.

The PATH option creates a non-binding ASSIGN. A non-binding ASSIGN acts like a DEFERred ASSIGN, except that it is re-evaluated each time the assigned name is referenced. For example, if you assign FONTS: to DF0:Fonts with the PATH option, any disk in DF0: is searched when FONTS: is referenced. As long as the disk contains a Fonts directory, it satisfies the ASSIGN. You cannot assign multiple directories with the PATH option.

Floppy disk only system users can find that using the PATH option eliminates the need to reinsert the original Workbench disk used to boot the system. As long as the drive you assigned with the PATH option contains a disk with the assigned directory name, the system uses that disk.

The DISMOUNT option disconnects a volume or device from the list of mounted devices. You must provide the device name in the argument. DISMOUNT removes the name from the list, but does not free resources. You cannot cancel a DISMOUNT without rebooting. DISMOUNT is meant for use by software developers only and can cause a software failure if not used carefully.

Example 1:

```
1> ASSIGN FONTS: MyFonts:Fontdir
```

assigns the FONTS: directory to Fontdir on MyFonts:.

Example 2:

```
1> ASSIGN LIST
Volumes:
Ram Disk [Mounted]
Workbench [Mounted]
MyFonts [Mounted]

Directories:
LOCALE      Workbench:Locale
KEYMAPS     Workbench:Devs/Keymaps
PRINTERS    Workbench:Devs/Printers
REXX        Workbench:S
CLIPS       Ram Disk:Clipboards
ENV         Ram Disk:Env
T           Ram Disk:T
ENVARC      Workbench:Prefs/Env-Archive
```

```

SYS          Workbench:
C            Workbench:C
S            Workbench:S
L            Workbench:L
FONTs       MyFonts:Fontdir
DEVS        Workbench:Devs
LIBS        Workbench:Libs
            + Workbench:Classes

```

```

Devices:
PIPE AUX RAM CON
RAW PAR SER PRT DF0

```

shows a typical list of all current assignments. The plus sign indicates any additional directories with the same assignment.

Example 3:

```

1> ASSIGN FONTs: EXISTS
FONTs: MyFonts:FontDir

```

is an inquiry into the assignment of FONTs:. AmigaDOS responds by showing that FONTs: is assigned to the FontDir directory of the MyFonts volume. The return code is set to 0 if it exists or to 5 if it does not.

Example 4:

```

1> ASSIGN LIBS: SYS:Libs BigAssem:Libs ADD

```

is a multiple-directory assignment that creates a search path containing two Libs directories. Specifying ADD keeps the standard SYS:Classes assignment from being removed. These directories are searched in sequence each time LIBS: is invoked.

Example 5:

```

1> ASSIGN DEVS:

```

removes the DEVS: assignment from the system.

Example 6:

```
1> ASSIGN WorkDisk: DF0: DEFER
1> ASSIGN WorkDisk: EXISTS
WorkDisk <DF0:>
```

sets up a late-binding assignment of the logical device WorkDisk:. Until the first time you refer to the name WorkDisk:, you do not need to insert it in DF0: ASSIGN shows DF0: enclosed in angle brackets to indicate that it is DEFERred. After the first reference to WorkDisk:, the volume name of the disk that was in DF0: replaces <DF0:>.

Example 7:

```
1> ASSIGN C: DF0:C PATH
1> ASSIGN C: EXISTS
C    [DF0:C]
```

references the C directory of the disk that is in DF0: when a command is searched for. ASSIGN shows DF0:C in square brackets to indicate that it is a non-binding ASSIGN.

Example 8:

```
1> ASSIGN LIBS: ZCad:Libs ADD
```

adds ZCad:Libs to the list of directories assigned as LIBS:.

Example 9:

```
1> ASSIGN LIBS: ZCad:Libs REMOVE
```

removes ZCad:Libs from the list of directories assigned as LIBS:.

For more examples using ASSIGN, see Chapter 8.

AVAIL

Reports the amount of Chip and Fast memory available.

Format AVAIL [CHIP|FAST|TOTAL] [FLUSH]

Template CHIP/S,FAST/S,TOTAL/S,FLUSH/S

Location C:

AVAIL gives a summary of the system RAM, both Chip and Fast. For each memory type, AVAIL reports the total amount of memory, how much is available, how much is currently in use, and the largest contiguous memory block not yet allocated.

Unless you want a complete summary, use the CHIP, FAST, and/or TOTAL options to have AVAIL display only the number of free bytes of Chip, Fast, or Total RAM available.

The FLUSH option frees memory by removing all unused libraries, devices, fonts, catalogs.

Example 1:

```
1> AVAIL
Type   Available   In-Use   Maximum   Largest
chip   233592        282272   515864    76792
fast   341384        182896   524280    197360
total  574976        465168   1040144   197360
```

A complete summary of system RAM is displayed.

Example 2:

```
1> AVAIL CHIP
233592
```

The number of free bytes of Chip RAM is displayed.

See Chapter 8 for more examples using AVAIL.

BREAK

Sets attention flags in the specified process.

Format BREAK <process> [ALL|C|D|E|F]

Template PROCESS/A/N,ALL/S,C/S,D/S,E/S,F/S

Location C:

BREAK sets the specified attention flags in the <process> indicated. Use the STATUS command to display the current process numbers. C sets the Ctrl+C flag, D sets the Ctrl+D flag, and so on. ALL sets all the flags from Ctrl+C to Ctrl+F. By default, only the Ctrl+C flag is set.

BREAK acts the same as selecting the relevant process by clicking in its window and pressing the appropriate Ctrl+key combinations.

Ctrl+C is the default for sending a BREAK signal to halt a process. A process that has been aborted this way displays *****Break** in the Shell window. Ctrl+D halts execution of a script file. The STATUS command displays the current process numbers. Ctrl+E is undefined.

Ctrl+F is used by programs that open windows to activate their window and bring it to the front of all windows. Not all programs respond to Ctrl+F.

Example 1:

```
1> BREAK 7
```

sets the Ctrl+C attention flag of process 7. This is the same as selecting process 7 and pressing Ctrl+C.

Example 2:

```
1> BREAK 5 D
```

sets the Ctrl+D attention flag of process 5.

See also: STATUS

CD

Sets or displays the current directory.

Format CD [<dir|pattern>]

Template DIR

Location Internal

CD with no arguments displays the name of the current directory. When a valid directory name is given, CD makes the named directory the current directory.

You must specify a complete path to the directory since CD does not search through the disk for it. If CD cannot find the specified directory in the current directory or in the given path, a **Can't find <directory>** message is displayed.

To move up a level in the filing hierarchy to the parent directory of the current directory, enter `CD` followed by a space and a single slash (`/`). You can move to another directory in the parent at the same time by including its name after the slash. If the current directory is a root directory, `CD /` has no effect. Use multiple slashes with no spaces between them to refer to additional higher levels.

To move directly to the root directory of the current device, use `CD` followed by a space and a colon; for example, `CD :`.

AmigaDOS supports an implied `CD` so that the `CD` command itself can often be left out. Enter the directory name, path, colon, or slashes at the prompt.

`CD` also supports pattern matching. When a directory matching the specified pattern is found, it becomes the current directory. If more than one directory matches the given pattern, an error message is displayed. You cannot use pattern matching with implied `CD`. For more information on pattern matching, see Chapter 3.

Example 1:

```
1> CD DF1:Work
```

sets the current directory to the `Work` directory on the disk in drive `DF1:`.

Example 2:

```
1> CD SYS:Com/Basic
```

makes the subdirectory `Basic` in the `Com` directory the current directory.

Example 3:

```
1> //
```

using the implied `CD`, moves up two levels in the directory structure.

Example 4:

```
1> CD SYS:Li#?
```

uses the `#?` pattern to match with the `LIBS:` directory.

For more examples using the `CD` command, see Chapter 8.

CHANGETASKPRI

Changes the priority of a currently running process.

Format CHANGETASKPRI <priority> [*PROCESS* <process number>]

Template PRI=PRIORITY/A/N,PROCESS/K/N

Location C:

CHANGETASKPRI changes the priority of the specified Shell process. If no process is specified, the current Shell process is assumed. Any shell process started from <process number> inherits its priority.

Use the STATUS command to display the current process numbers.

The range of acceptable values for <priority> is the integers from -128 to 127, with higher values yielding a higher priority (a greater proportion of CPU time is allocated). However, do not enter values above +10 to avoid disrupting important system tasks.

Example:

```
1> CHANGETASKPRI 4 Process 2
```

The priority of Process 2 is changed to 4. Any shell process started from this Shell also has a priority of 4. They have priority over any other user tasks created without using CHANGETASKPRI (those tasks have a priority of 0).

See also: STATUS. For another example for using CHANGETASKPRI, see Chapter 8.

COPY

Copies files or directories.

Format COPY [FROM] {<name | pattern>} [TO] <name>
[ALL] [QUIET] [BUF | BUFFER=<n>] [CLONE]
[DATES] [NOPRO] [COM] [NOREQ]

Template FROM/M,TO/A,ALL/S,QUIET/S,
BUF=BUFFER/K/N,CLONE/S,DATES/S,
NOPRO/S,COM/S,NOREQ/S

Location C:

COPY copies the file or directory specified with the FROM argument to the file or directory specified by the TO argument. You can copy several items at once by giving more than one name/pattern in the FROM argument; they should be separated by spaces. If the FROM argument is a pattern or consists of multiple names, the TO argument must be a directory.

If a TO file name already exists, COPY overwrites the TO file with the FROM file. You can use a pair of double quotation marks (""") to refer to the current directory. When used as the FROM argument, "" copies all the files in the current directory. Do not put any spaces between the double quotation marks.

If the FROM argument is a directory, only the directory's files are copied; its subdirectories are not copied. Use the ALL option to copy the complete directory, including its files, subdirectories, and the subdirectories' files. It is possible to create a directory as you copy if you are copying more than one file. To give the new directory a name, specify the directory name as the last component in the TO argument's path. This can be any name, including the same name as the original if it is on a different path.

COPY prints to the screen the name of each file as it is copied. This can be overridden by the QUIET option.

The BUF= option is used to set the number of 512-byte buffers used during the copy. (Default is 128 buffers, 64 KB of RAM.) Limit the number of buffers when copying to RAM: BUF=0 uses a buffer the same size as the file to be copied.

By default, COPY gives a TO file the timestamp of when the copy was made, rather than that of the original file. Also by default, comments attached to the original FROM file are not copied and the protection bits of the FROM file are copied to the TO file. You can override these defaults using the following:

CLONE	The timestamp, comments, and protection bits of the FROM file are copied to the TO file.
DATES	The timestamp of the FROM file is copied to the TO file.
COM	Any comment attached to the FROM file is copied to the TO file.
NOPRO	The protection bits of the FROM file are not copied to the TO file. The TO file is given standard protection bits of r, w, e, and d.

COPY displays a requester if the COPY cannot continue. When the NOREQ option is given, all requesters are suppressed. Use this in scripts to prevent a COPY failure from stopping the script to wait for a response. With the NOREQ option, the COPY command is aborted and the script continues.

Example 1:

```
1> COPY File1 TO :Work/File2
```

copies File1 in the current directory to the Work directory in the root of the current device, renaming it File2.

Example 2:

```
1> COPY Chapter#? TO DF1:Backup
```

copies all the files whose names start with Chapter in the current directory to the Backup directory on the disk in DF1. The Backup directory is created if it does not already exist.

Example 3:

```
1> COPY Work:Test TO ""
```

copies the files in the Test directory on Work to the current directory; subdirectories in Test are not copied.

Example 4:

```
1> COPY Work:Test TO DF0:Test ALL
```

copies all the files and any subdirectories of the Test directory on Work to the Test directory on DF0:. If a Test directory does not already exist on DF0:, COPY creates one.

Example 5:

```
1> COPY DF0: TO DF1: ALL QUIET
```

copies all files and directories on the disk in DF0: to DF1:, without displaying on the screen any file/directory names as they are copied. (For disks less than half full, this can be faster than DiskCopy.)

For more examples using COPY, see Chapter 8.

CPU

Sets or displays processor options.

Format CPU [CACHE | NOCACHE] [BURST | NOBURST]
 [DATACACHE | NODATACACHE]
 [DATABURST | NODATABURST]
 [INSTCACHE | NOINSTCACHE]
 [INSTBURST | NOINSTBURST]
 [FASTROM | NOFASTROM] [TRAP | NOTRAP]
 [COPYBACK | NOCOPYBACK]
 [EXTERNALCACHE | NOEXTERNALCACHE]
 [NOMMUTEST] [CHECK 68010 | 68020 | 68030 |
 68040 | 68881 | 68882 | 68851 | MMU | FPU]

Template CACHE/S,BURST/S,NOCACHE/S,NOBURST/S,
 DATACACHE/S,NODATACACHE/S,
 DATABURST/S,NODATABURST/S,
 INSTCACHE/S,NOINSTCACHE/S,
 INSTBURST/S,NOINSTBURST/S,COPYBACK/S,
 NOCOPYBACK/S,EXTERNALCACHE/S,
 NOEXTERNALCACHE/S,FASTROM/S,
 NOFASTROM/S,TRAP/S,NOTRAP/S,
 NOMMUTEST/S,CHECK/K

Location C:

CPU adjusts various options of the microprocessor installed in your Amiga. CPU also shows the processor and options that are currently enabled.

Many options only work with certain members of the 680x0 processor family. The 68020 has a special type of memory known as instruction cache. When instruction cache is used, instructions are executed more quickly. The 68030 and 68040 have two types of cache memory: instruction and data.

If mutually exclusive options are specified, the safest option is used. Availability of the following options depends on the type of microprocessor present.

CACHE	Turns on all caches.
NOCACHE	Turns off all caches.
BURST	Turns on burst mode for both data and instructions.
NOBURST	Turns off burst mode for data and instructions.
DATACACHE	Turns on data cache.
NODATACACHE	Turns off data cache.
DATABURST	Turns on burst mode for data.
NODATABURST	Turns off burst mode for data.
INSTCACHE	Turns on instruction cache.
NOINSTCACHE	Turns off instruction cache.
INSTBURST	Turns on burst mode for instructions.
NOINSTBURST	Turns off burst mode for instructions.
FASTROM	With a processor having a supported MMU, copies the system ROM into 32-bit RAM, making access to operating system functions significantly faster. CPU then write-protects the RAM area so that the data cannot be changed.
NOFASTROM	Turns off FASTROM.
TRAP	This option is for developers only.
NOTRAP	This option is for developers only.
COPYBACK	Turns on 68040 copyback cache.
NOCOPYBACK	Turns off 68040 copyback cache.

EXTERNALCACHE	Turns on external cache.
NOEXTERNALCACHE	Turns off external cache.
NOMMUTEST	Allows the MMU settings to be changed without checking to see if an MMU is currently in use.

The **CHECK** option, when given with a keyword (68010, 68020, 68030, 68040, 68881, 68882, or 68851, MMU, FPU) checks for the presence of the processor indicated by the keyword.

Examples:

```
1> CPU
System: 68030 68881 (INST: Cache Burst) (DATA:
Cache NoBurst)

1> CPU NODATACACHE FASTROM
System: 68030 68881 FastROM (INST: Cache Burst)
(DATA: NoCache NoBurst)

1> CPU NOBURST DATACACHE NOINSTCACHE
System: 68030 68881 (INST: NoCache NoBurst) (DATA:
Cache NoBurst)
```

DATE

Displays or sets the system date and/or time.

Format DATE [<day>] [<date>] [<time>] [*TO* | *VER*
<filename>]

Template DAY,DATE,TIME,TO=VER/K

Location C:

DATE with no argument displays the currently set system time and date, including the day of the week. Time is displayed using a 24-hour clock.

DATE <date> sets only the date. The format for entry and display of <date> is DD-*MMM*-YY (day-month-year). The hyphens between the arguments are required. A leading zero in the date is not necessary. The number or the first three letters of the month (in English) must be used, as well as the last two digits of the year.

If the date is already set, you can reset it by specifying a day name. You can also use tomorrow or yesterday as the <day> argument. You cannot specify a day name to change the date to more than seven days into the future.

DATE <time> sets the time. The format for entry and display of <time> is HH:MM:SS (hours:minutes:seconds). Seconds is optional.

If your Amiga does not have a battery backed-up hardware clock and you do not set the date, when the system boots it sets the date to the date of the most recently created file on the boot disk.

If you specify the TO or VER option, followed by a file name, the output of the DATE command is sent to that file, overwriting any existing contents.

Adjustments made with DATE only change the software clock and do not survive powering off the system. To set the battery backed-up hardware clock from the Shell, you must set the date and use SETCLOCK SAVE.

Although DATE accepts and displays the date and time in a single format, programs such as Clock display the date and time according to your Locale country setting.

Example 1:

```
1> DATE
6-Sep-92
```

Example 2:

```
1> DATE 6-sep-92
```

sets the date to September 6, 1992. The time is not reset.

Example 3:

```
1> DATE tomorrow
```

resets the date to one day ahead.

Example 4:

```
1> DATE TO Fred
```

sends the current date to the file Fred.

Example 5:

```
1> DATE 23:00
```

sets the current time to 11:00 p.m.

Example 6:

```
1> DATE 1-jan-02
```

sets the date to January 1st, 2002. The earliest date you can set is January 1, 1978.

DELETE

Deletes files or directories.

Format DELETE {<name | pattern>} [ALL] [QUIET]
 [FORCE]

Template FILE/M/A,ALL/S,QUIET/S,FORCE/S

Location C:

DELETE attempts to erase the specified items. You can delete multiple items at the same time by listing them individually or by using a wildcard to delete a specific set of files matching a pattern. The pattern can specify directory levels, as well as names. To abort a multiple-item DELETE, press Ctrl+C. A multiple-item DELETE aborts if and when it finds something that cannot be removed; for example, a file is delete-protected or in use. A pattern matching DELETE removes everything it can and lists the items that it did not delete, if any.

Note AmigaDOS does not request confirmation of deletions. Do not use pattern matching to delete things if you are not familiar with the procedure; deleted items cannot be recovered, unless you have an up-to-date backup of the items deleted.

An error message warns you that you cannot delete directories that still contain files. Override this using the ALL option. DELETE ALL deletes the named directory, its subdirectories, and all files.

File names are displayed on the screen as they are deleted. To suppress the screen output, use the QUIET option.

If the d (deletable) protection bit of a file or directory has been cleared, that item cannot be deleted unless the FORCE option is used.

Example 1:

```
1> DELETE Old-file
```

deletes the file named Old-file in the current directory.

Example 2:

```
1> DELETE Work/Prog1 Work/Prog2 Work
```

deletes the files Prog1 and Prog2 in the Work directory and then deletes the Work directory if it contains no other files.

Example 3:

```
1> DELETE T#?/#?(1|2)
```

deletes all the files that end in 1 or 2 in directories that start with T.

Example 4:

```
1> DELETE DF1:#? ALL FORCE
```

deletes all the files on DF1:, even those set as not deletable.

See also: PROTECT. For more examples using DELETE, see Chapter 8.

DIR

Displays a sorted list of the files in a directory.

Format DIR [<dir | pattern>] [OPT A | I | AI | D | F] [ALL] [DIRS] [FILES] [INTER]

Template DIR,OPT/K,ALL/S,DIRS/S,FILES/S,INTER/S

Location C:

DIR displays the file and directory names contained in the specified directory or the current directory. Directories are listed first, followed

by an alphabetical list of the files in two columns. Pressing Ctrl+C aborts a directory listing.

The options are:

ALL	Displays all subdirectories and their files.
DIRS	Displays only directories.
FILES	Displays only files.
INTER	Enters an interactive listing mode.

The ALL, DIRS, FILES, and INTER keywords supersede the OPT A, D, F, and I options, respectively. The older keywords are retained for compatibility with earlier versions of AmigaDOS. Do not use OPT with the full keywords — ALL, DIRS, FILES, or INTER.

Interactive listing mode stops after each name to display a question mark at which you can enter commands. The acceptable responses are shown below:

Press Return	Displays the next name on the list.
E	Enters a directory; the files in that directory are displayed.
B	Goes back one directory level.
DEL or DELETE	Deletes a file or empty directory. DEL does not refer to the Del key; enter the letters D, E, and L.
T	Types the contents of a file.
C or COMMAND	Allows you to enter additional AmigaDOS commands.
Q	Quits interactive editing.
?	Displays a list of the available interactive-mode commands.

The COMMAND option allows almost any AmigaDOS command to be executed during the interactive directory list. To issue a command, enter C (or COMMAND) at the question mark prompt. DIR asks you for the command. Enter the desired command, then press Return. The command is executed and DIR continues. You can also combine the C and the command on one line by putting the command in quotation marks following the C.

For example,

```
? C "type prefs.info hex"
```

is equivalent to pressing **Q** to exit interactive listing mode and return to a regular Shell prompt, then entering:

```
1> TYPE Prefs.info HEX
```

to display the Prefs.info file on the screen in hexadecimal format.

Formatting a disk from the DIR interactive mode is not recommended since the format takes place immediately, without any confirmation requesters appearing. Do not start another interactive DIR from interactive mode since it results in garbled output.

Example 1:

```
1> DIR Workbench:
```

displays a list of the directories and files on the Workbench disk.

Example 2:

```
1> DIR MyDisk:#?.memo
```

displays all the directories and files on MyDisk that end in .memo.

Example 3:

```
1> DIR Extras: ALL
```

displays the complete contents of the Extras drawer: all directories, all subdirectories, and all files, including those in the subdirectories.

Example 4:

```
1> DIR Workbench: DIRS
```

displays only the directories on Workbench.

Example 5:

```
1> DIR Workbench: INTER
```

begins an interactive list of the contents of the Workbench disk.

For more examples using DIR, see Chapter 8.

DISKCHANGE

Notifies the Amiga that you have changed a disk in a disk drive.

Format DISKCHANGE <device>

Template DRIVE/A

Location C:

You must use the DISKCHANGE command to inform the system when you change disks or cartridges in 5.25 inch floppy disk drives or removable media drives without automatic diskchange hardware.

Example:

If a requester asks you to insert a new disk into your 5.25 inch drive, known as DF2:, you must insert the disk and then enter:

```
1> DISKCHANGE DF2:
```

AmigaDOS then recognizes the new disk and you can proceed.

ECHO

Displays a string.

Format ECHO [<string>] [NOLINE] [FIRST <n>] [LEN <n>]
 [TO <filename>]

Template /M,NOLINE/S,FIRST/K/N,LEN/K/N,TO/K

Location Internal

ECHO writes the specified string to the current output window or device. By default the string is sent to the screen, but if you use the TO option, you can send the string to any specified device or file.

When the NOLINE option is specified, ECHO does not automatically move the cursor to the next line after printing the string.

The FIRST and LEN options allow the echoing of a substring. FIRST <n> indicates the character position from which to begin the echo; LEN <n> indicates the number of characters of the substring to echo, beginning with the FIRST character. If the FIRST option is omitted and only the LEN keyword is given, the substring printed consists of

the rightmost <n> characters of the main string. For example, if your string is 20 characters long and you specify LEN 4, the 17th, 18th, 19th, and 20th characters of the string are echoed.

Examples:

```
1> ECHO "hello out there!"  
hello out there!  
  
1> ECHO "hello out there!" NOLINE FIRST 0 LEN 5  
hello1>
```

For further examples using the ECHO command, see Chapter 8.

ED

Edits text files (a screen editor).

Format ED [FROM] <filename> [SIZE <n>] [WITH
<filename>] [WINDOW <window specification>]
[TABS <n>] [WIDTH | COLS <n>] [HEIGHT | ROWS
<n>]

Template FROM/A,SIZE/N,WITH/K,WINDOW/K,
TABS/N,WIDTH=COLS/N,HEIGHT=ROWS/N

Location C:

For more information on ED, see Chapter 4. See Chapter 8 for an example using ED.

EDIT

Edits text files by processing the source file sequentially (a line editor).

Format EDIT [FROM] <filename> [[TO] <filename>] [WITH <filename>] [VER <filename>][OPT P <lines> | W <chars> | P<lines>W<chars>] [WIDTH <chars>] [PREVIOUS <lines>]

Template FROM/A,TO,WITH/K,VER/K,OPT/K,WIDTH/N,PREVIOUS/N

Location C:

For more information on EDIT, see Chapter 4.

ELSE

Specifies an alternative for an IF statement in a script file.

Format ELSE

Template (none)

Location Internal

ELSE must be used in conjunction with the IF command. ELSE is used in an IF block of a script to specify an alternative action if the IF condition is not true. If the IF condition is not true, execution of the script jumps from the IF line to the line after ELSE; all intervening commands are skipped. If the IF condition is true, the commands immediately following the IF statement are executed up to the ELSE. Then, execution skips to the ENDIF statement that concludes the IF block.

Example:

Assume a script, called Display, contains the following block:

```
IF exists picfile
    MultiView picfile
ELSE
    ECHO "picfile is not in this directory"
ENDIF
```

If picfile can be found in the current directory, the MultiView program is executed and picfile is displayed on the screen.

If picfile cannot be found in the current directory, the script skips to the ECHO command. The following message is displayed in the Shell window:

```
picfile is not in this directory
```

See also: IF, ENDIF, EXECUTE

ENDCLI

Ends a Shell process.

Format ENDIF

Template (none)

Location Internal

ENDCLI ends a Shell process.

See also: ENDSHELL

ENDIF

Terminates an IF block in a script file.

Format ENDIF

Template (none)

Location Internal

ENDIF must be used when an IF command is used. ENDIF is used in scripts at the end of an IF block. If the IF condition is not true or if the true-condition commands are executed and an ELSE is encountered, the execution of the script skips to the next ENDIF command. Every IF statement must be terminated by an ENDIF.

The ENDIF applies to the most recent IF or ELSE command.

See also: IF, ELSE. For examples using the ENDIF command, see Chapter 8.

ENDSHELL

Ends a Shell process.

Format ENDSHELL

Template (none)

Location Internal

ENDSHELL ends a Shell process and closes the Shell window.

The Shell process can also be ended by ENDCLI, by clicking on the close gadget, or by pressing CTRL+\.

Use ENDSHELL only when the Workbench or another Shell is running. If you quit the Workbench and you close your only Shell, you cannot communicate with the Amiga without rebooting.

The Shell window cannot close if any processes that were launched from the Shell and not detached are still running. Even though the window stays open, the Shell does not accept new input. You must terminate those processes before the window closes. For example, if you opened an editor from the Shell, the Shell window does not close until you exit the editor.

For examples using the ENDSHELL command, see Chapter 8.

ENDSKIP

Terminates a SKIP block in a script file.

Format ENDSKIP

Template (none)

Location Internal

ENDSKIP is used in scripts to terminate the execution of a SKIP block. A SKIP block allows you to jump over intervening commands if a certain condition is met. When an ENDSKIP is encountered, execution of the script resumes at the line following the ENDSKIP. The condition flag is set to 5 (WARN).

See also: SKIP

EVAL

Evaluates integer or Boolean expressions.

Format EVAL <value1> {[<operation>] [<value2>]} [TO
<file>] [LFORMAT=<string>]

Template VALUE1/A,OP,VALUE2/M,TO/K,LFORMAT/K

Location C:

EVAL is used to evaluate and print the answer of an integer expression. The fractional portion of input values and final results, if any, is truncated (cut off). If a non-integer is given as an input value, evaluation stops at the decimal point.

<Value1> and <value2> can be decimal (the default), hexadecimal, or octal numbers. Hexadecimal numbers are indicated by either a leading 0x or #x. Octal numbers are indicated by either a leading 0 or a leading #. Alphabetical characters are indicated by a leading single quotation mark (') and are evaluated as their ASCII equivalent.

The LFORMAT keyword specifies the formatting string used to print the answer. You can use %X (hexadecimal), %O (octal), %N (decimal), or %C (character). The %X and %O options require a number of digits specification (for example, %X8 gives 8 digits of hex output). When using the LFORMAT keyword, you can specify to print a new line by including *N in your string.

The supported operations and their corresponding symbols are shown in the following table.

addition	+
subtraction	-
multiplication	*
division	/
modulo	mod, M, m, or %
bitwise AND	&
bitwise OR	
bitwise NOT	~
left shift	lsh, L, or l

right shift	rsh, R, or r
negation	-
exclusive OR	xor, X, or x
bitwise equivalence	eqv, E, or e

EVAL can be used in scripts as a counter for loops. In that case, use the TO option to send the output of EVAL to a file.

Parentheses can be used in the expressions.

Example 1:

```
1> EVAL 64 / 8 + 2
10
```

Example 2:

```
1> EVAL 0x5f / 010 LFORMAT="The answer is %X4*N"
The answer is 000B
1>
```

This divides hexadecimal 5f (95) by octal 10 (8), yielding 000B, the integer portion of the decimal answer 11.875. (The 1> prompt appears immediately after the 000B if *N is not specified in the LFORMAT string.)

For more examples using the EVAL command, see Chapter 8.

EXECUTE

Executes a script with optional argument substitution.

Format EXECUTE <script> [{<arguments>}]

Template FILE/A

Location C:

EXECUTE is used to run scripts of AmigaDOS commands. The lines in the script are executed as if they had been entered at a Shell prompt. If the s protection bit of a file is set and the file is in the search path, enter only the file name; the EXECUTE command is not needed.

You can use parameter substitution in scripts by including special keywords in the script. When these keywords are used, you can pass variables to the script by including the variable in the EXECUTE command line. Before the script is executed, AmigaDOS checks the parameter names in the script against any arguments given on the command line. If any match, AmigaDOS substitutes the values specified on the command line for the parameter name in the script. You can also specify default values for AmigaDOS to use if no variables are given. If you have not specified a variable and there is no default specified in the script, then the value of the parameter is empty (no substitution is made).

The allowable keywords for parameter substitution are explained in Chapter 5. Each keyword command line must be prefaced with a dot character (.).

See also: IF, SKIP, FAILAT, LAB, ECHO, RUN, QUIT. For examples using the EXECUTE command, see Chapter 8.

FAILAT

Instructs a command sequence not to fail unless a given error condition is returned.

Format FAILAT [<n>]

Template RCLIM/N

Location Internal

Commands indicate that they have failed by setting a nonzero return code. The return code, normally 5, 10, or 20, indicates the severity of the error. A return code greater than or equal to a certain limit, the fail limit, terminates a sequence of non-interactive commands (commands specified after RUN or in a script).

Use the FAILAT command to alter the fail limit RCLIM (Return Code Limit) from its initial value of 10. If you increase the limit, you indicate that certain classes of error should not be regarded as fatal and that execution of subsequent commands can proceed after the error. The argument must be a positive number. The fail limit is reset to the initial value of 10 on exit from the command sequence.

If the argument is omitted, the current fail limit is displayed.

Example:

Assume a script contains the following lines:

```
COPY DF0:MyFile to RAM:
ECHO "MyFile being copied."
```

If MyFile cannot be found, the script is aborted and the following message appears in the Shell window:

```
COPY: object not found
COPY failed returncode 20:
```

However, if you changed the return code limit to higher than 20, the script continues even if the COPY command fails. For example, if you changed the script to read:

```
FAILAT 21
COPY DF0:MyFile to RAM:
ECHO "MyFile being copied."
```

even if MyFile cannot be found, the script continues. The following message appears in the Shell window:

```
COPY: object not found
MyFile being copied.
```

See also: ECHO, EXECUTE.

FAULT

Prints the messages for the specified error numbers.

Format FAULT {<n>}

Template /N/M

Location Internal

FAULT prints the messages corresponding to the error numbers supplied. As many error numbers, separated by spaces, as you want can be specified to print at the same time.

Example:

If you receive the error message:

```
Error when opening DF1:TestFile 205
```

and need more information, enter:

```
1> FAULT 205
FAULT 205: object not found
```

This tells you that the error occurred because TestFile could not be found on DF1:.

A complete list of error messages appears in Appendix A.

FILENOTE

Attaches a comment to a file.

Format FILENOTE [FILE] <file | pattern> [COMMENT
<comment>] [ALL] [QUIET]

Template FILE/A,COMMENT,ALL/S,QUIET/S

Location C:

FILENOTE attaches an optional comment of up to 79 characters to the specified file or to all files matching the given pattern.

If the <comment> includes spaces, it must be enclosed in double quotation marks. To include double quotation marks in a filenote, each literal quotation mark must be immediately preceded by an asterisk (*) and the entire comment must be enclosed in quotation marks, regardless of whether the comment contains any spaces.

If the <comment> argument is omitted, any existing filenote is deleted from the named file.

Creating a comment with FILENOTE is the same as entering a comment into the Comment gadget of an icon's Information window. Changes made with FILENOTE are reflected in the Information window, and vice versa.

Use the LIST command to view comments made with FILENOTE. If a file has comments, LIST displays them below the file name.

When an existing file is copied to (specified as the TO argument of a COPY command), it is overwritten, but its original comment is retained. Any comment attached to a FROM file is not copied unless the CLONE or COM option of COPY is specified.

If the ALL option is given, FILENOTE adds the <comment> to all the files and subdirectories matching the pattern entered. If the QUIET option is given, screen output is suppressed.

Example 1:

```
1> FILENOTE Sonata "allegro non troppo"
```

attaches the filenote **allegro non troppo** to the Sonata file.

Example 2:

```
1> FILENOTE Toccata "*"presto*"
```

attaches the filenote **"presto"** to the Toccata file.

GET

Gets the value of a local variable.

Format GET <name>

Template NAME/A

Location Internal

GET is used to retrieve and display the value of a local environment variable. The value is displayed in the current window.

Local environment variables are only recognized by the Shell in which they are created or by any Shells created from a NEWSHELL command executed in the original Shell. If you open an additional Shell by opening the Shell icon or by using the Execute Command menu item, previously created local environment variables are not available.

Example:

```
1> GET editor
Extras:Tools/MEmacs
```

See also: SET

GETENV

Gets the value of a global variable.

Format GETENV <name>

Template NAME/A

Location Internal

GETENV is used to retrieve and display the value of a global environment variable. The value is displayed in the current window. Global variables are stored in ENV: and are recognized by all Shells.

Example:

```
1> GETENV editor
Extras:Tools/MEmacs
```

See also: SETENV

ICONX

Note ICONX is used only as a default tool in a project icon and cannot be used as a Shell command.

Allows execution of a script file of AmigaDOS commands from an icon.

Format ICONX

Template (none)

Location C:

To use ICONX, create or copy a project icon for the script. Open the icon's Information window and change the Default Tool of the icon to

C:ICONX and select Save to store the changed .info file. The script can then be executed by double-clicking on the icon.

When the icon is opened, ICONX changes the current directory to the directory containing the project icon before executing the script. A console window can be opened on the Workbench screen if the script produces output.

Several Tool Types can be specified in the script icon. The WINDOW Tool Type provides an alternate window specification for the input/output window. By default, the window's specification is:

```
WINDOW=CON:0/50//80/IconX/AUTO/WAIT/CLOSE
```

The AUTO option opens a window only if there is output created by the script. If a window opens, the WAIT option forces it to remain open after the script terminates until you specifically close it. The CLOSE option gives the window a close gadget.

The WAIT Tool Type (not to be confused with the WAIT option of the WINDOW Tool Type) specifies the number of seconds the input/output window should remain open after the script terminates. If you use this option the default input/output window cannot be closed before the WAIT period has expired. There is also a DELAY Tool Type that works in a very similar way, except that its parameter is in 1/50th of a second, instead of in seconds.

The STACK Tool Type specifies the number of bytes to use for stack during script execution. If this Tool Type is not provided, the default 4096 bytes is used.

Finally, the USERSHELL Tool Type runs the script file using the current User Shell instead of the normal ROM Shell. You must specify USERSHELL=YES. User Shells are third party shells that you can purchase and install in your system to replace the standard Amiga Shell environment that comes with the operating system.

Extended selection passes files that have icons to the script. Their file names appear to the script as keywords. To use this facility, the .KEY keyword must appear at the start of the script. In this case, the AmigaDOS EXECUTE command is used to execute the script file.

See also: EXECUTE. For examples using the ICONX command, see Chapter 8.

IF

Evaluates conditional operations in script files.

Format	IF [NOT] [WARN ERROR FAIL] [<string> EQ GT GE <string>] [VAL] [EXISTS <filename>]
Template	NOT/S,WARN/S,ERROR/S,FAIL/S,EQ/K,GT/K, GE/K,VAL/S,EXISTS/K
Location	Internal

In a script file, IF, when its conditional is true, carries out all the subsequent commands until an ENDIF or ELSE command is found. IF must be used in conjunction with ENDIF, however, ELSE is optional. When the conditional is not true, execution skips directly to the ENDIF or to an ELSE. The conditions and commands in IF and ELSE blocks can span more than one line before their corresponding ENDIFs.

Nested IFs jump to the matching ENDIF.

The additional keywords are as follows:

NOT	Reverses the interpretation of the result.
WARN	True if previous return code is greater than or equal to 5.
ERROR	True if previous return code is greater than or equal to 10; only available if FAILAT is set to greater than 10.
FAIL	True if previous return code is greater than or equal to 20; only available if FAILAT is set to greater than 20.
<a> GT 	True if the test of a is greater than the text of b (disregarding case). Use NOT GT for less than.
<a> GE 	True if the text of a is greater than or equal to the text of b (disregarding case). Use NOT GE for less than or equal to.
<a> EQ 	True if the text of a and b is identical (disregarding case).
VAL	Specifies a numeric comparison.
EXISTS <file>	True if the file exists.

If more than one of the three condition-flag keywords (WARN, ERROR, FAIL) are given, the one with the lowest value is used.

You can use local or global variables with IF by prefacing the variable name with a \$ character.

Example 1:

```
IF EXISTS Work/Prog
  TYPE Work/Prog HEX
ELSE
  ECHO "It's not here"
ENDIF
```

AmigaDOS displays the file `Work/Prog` if it exists in the current directory. Otherwise, AmigaDOS displays the message **It's not here** and continues after the `ENDIF`.

Example 2:

```
IF ERROR
  SKIP errlab
ENDIF
ECHO "No error"
LAB errlab
```

If the previous command produces a return code greater than or equal to 10, AmigaDOS skips over the `ECHO` command to the `errlab` label.

See also: `EXECUTE`, `FAILAT`, `LAB`, `QUIT`, `SKIP`. For more examples using the `IF` command, see Chapter 8.

INFO

Gives information about mounted devices.

Format `INFO [<device>]`

Template `DEVICE`

Location `C:`

`INFO` displays a line of information about each mounted storage device, including floppy disk drive and hard disk partitions. Listed are the unit name, maximum size of the disk, the used and free space in blocks, the percentage of the disk that is full, the number of soft disk errors that have occurred, the status of the disk, and the name of the disk.

With the `<device>` argument, `INFO` provides information on the specified device or volume only.

Example:

```

1>INFO
Unit Size Used Free  Full Errs Status      Name
DF0: 879K 1738  20  98%  0   Read Only  Workbench
DF1: 879K  418 1140  24%  0   Read/Write Text-6

Volumes available:
Workbench [Mounted]
Text-6 [Mounted]

```

INSTALL

Writes or inspects a boot block on a formatted floppy disk or PCMCIA card, specifying whether it should be bootable.

Format INSTALL [DRIVE] <DF0: | DF1: | DF2: | DF3: | CC0:>
 [NOBOOT] [CHECK] [FFS]

Template DRIVE/A,NOBOOT/S,CHECK/S,FFS/S

Location C:

INSTALL clears a floppy disk's or PCMCIA memory card's boot block area and writes a valid boot block onto it. INSTALL does not affect any files or directories on the disk or card. The necessary files and directories must still be present on a device to boot from it successfully.

The NOBOOT option removes the boot block from an AmigaDOS disk or card, making it not bootable.

The CHECK option checks for valid boot code. It reports whether a disk or card is bootable and whether standard Commodore-Amiga boot code is present on the media. This is useful in detecting some viruses.

The FFS switch is ignored. It remains part of the template to ensure compatibility with earlier scripts and programs.

Example 1:

```

1> INSTALL DF0: CHECK
No bootblock installed

```

indicates that there is a non-bootable floppy in DF0:.

Example 2:

```
1> INSTALL DF0:
```

makes the disk in drive DF0: a bootable disk.

Example 3:

```
1> INSTALL DF0: CHECK  
Appears to be FFS bootblock
```

indicates that there is a bootable FFS floppy in DF0:.

JOIN

Concatenates two or more files into a new file.

Format JOIN [FILE] {<file | pattern>} AS | TO <filename>

Template FILE/M/A,AS=TO/K/A

Location C:

JOIN copies all the listed files, in the order given, to one new file. This destination file cannot have the same name as any of the source files. You must supply a destination file name. The original files remain unchanged. Any number of files can be JOINed in one operation.

TO can be used as a synonym for AS.

Example:

```
1> JOIN Part1 Part2 Part3 AS Textfile
```

For another example using JOIN, see Chapter 8.

LAB

Specifies a label in a script file.

Format LAB [<string>]

Template (none)

Location Internal

LAB is used in scripts to define a label that is searched for by the SKIP command. The label <string> can be of any length, but must be alphanumeric. No symbols are allowed. If the <string> contains spaces, it must be enclosed in quotation marks.

See also: SKIP, IF, EXECUTE. For more examples using LAB, see Chapter 8.

LIST

Lists specified information about directories and files.

Format LIST [(*<dir | pattern | filename>*)] [*P | PAT <pattern>*] [*KEYS*] [*DATES*] [*NODATES*] [*TO <name>*] [*SUB <string>*] [*SINCE <date>*] [*UPTO <date>*] [*QUICK*] [*BLOCK*] [*NOHEAD*] [*FILES*] [*DIRS*] [*LFORMAT <string>*] [*ALL*]

Template DIR/M,P=PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,SUB/K,SINCE/K,UPTO/K,QUICK/S,BLOCK/S,NOHEAD/S,FILES/S,DIRS/S,LFORMAT/K,ALL/S

Location C:

LIST displays information about the contents of the current directory. If you specify a <dir>, <pattern>, or <filename> argument, LIST displays information about the specified directory, all directories or files that match the pattern, or the specified file, respectively. The PAT argument lets you specify an additional pattern to match.

Unless other options are specified, LIST displays the following:

name	The name of the file or directory.
size	The size of the file in bytes. If there is nothing in this file, the field reads "empty". For directories, this entry reads "Dir".
protection	The protection bits that are set for this file are shown as letters. The clear (unset) bits are shown as hyphens. Most files show the default protection bits, ---rwd for readable/writable/executable/deletable. See the PROTECT command for more on protection bits.

- date and time** The date and time the file was created or last changed.
- comment** The comment, if any, placed on the file using the FILENOTE command. It is preceded by a colon (:).

LIST uses the following options to change the way the output is displayed:

- KEYS** Displays the block number of each file header or directory.
- DATES** Displays dates. (For example, DD-MMM-YY is the USA default).
- NODATES** Does not display date and time information.
- TO <name>** Specifies an output file or device for LIST; by default, LIST outputs to the current window.
- SUB <string>** Lists only files containing the substring <string>.
- SINCE <date>** Lists only files timestamped on or after the specified date.
- UPTO <date>** Lists only files timestamped on or before the specified date.
- QUICK** Lists only the names of files and directories.
- BLOCK** Displays file sizes in 512-byte blocks, rather than bytes.
- NOHEAD** Suppresses printing of the header and summary information.
- FILES** Lists files only (no directories).
- DIRS** Lists directories only (no files).
- LFORMAT** Defines a string to specially format LIST output.
- ALL** Lists the contents of all directories and subdirectories.

The LFORMAT option modifies the output of LIST and can be used as a quick method of generating script files. When using LFORMAT, specify an output format string; this string is output for each file or directory normally listed. It can contain any text you specify, plus the usual LIST output information. When LFORMAT is specified, the QUICK and NOHEAD options are automatically selected. To save the output, you must redirect it to a file by using the > operator or specifying a TO file. (For examples using the LIST LFORMAT option, see Chapter 8.)

The available substitution operators are:

- %A** Prints file attributes (protection bits).
- %B** Prints size of file in blocks.

%C	Prints any comments attached to the file.
%D	Prints the date associated with the file.
%E	Prints just the file extension.
%K	Prints the file key block number.
%L	Prints the length of the file in bytes.
%M	Prints the file name only, omitting any extension.
%N	Prints the name of the file.
%P	Prints the file parent path.
%S	Superseded by %N and %P; still functional.
%T	Prints the time associated with the file.

You can put a length specifier and/or a justification specifier between the percent sign (%) and the field specifier. To specify left justification, place a minus sign (-) before the length specifier. Otherwise, the information displayed is right justified.

The default output of the LIST command uses the following specification:

```
%-24 %7L %A %D %T
```

Example 1:

```
> LIST Dirs
Prefs      Dir  ----rwed  27-Jun-93   11:43:59
T          Dir  ----rwed  16-Jul-93   11:37:43
Trashcan   Dir  ----rwed  21-Jun-93   17:54:20
```

Only the directories in the current directory, in this case SYS:, are listed. (A shortened version of the typical output is shown above.)

Example 2:

```
1> LIST LI#? TO RAM:Libs.file
```

LIST searches for any directories or files that start with LI. The output of LIST is sent to Libs.file in RAM:.

Example 3:

```
1> LIST DF0:Documents UPTO 09-Oct-90
```

Only the files or directories in the Documents directory of DF0: that have not been changed since October 9, 1990 are listed.

For further examples using the LIST command, see Chapter 8.

LOADRESOURCE

Preloads resources into memory to avoid excessive disk swaps.

Format LOADRESOURCE {<name>} [LOCK|UNLOCK]

Template NAME/M,LOCK/S,UNLOCK/S

Location C:

LOADRESOURCE reduces the need for excessive disk swaps on floppy-only systems by preloading the following types of resources into memory:

- Libraries** Specify the path name to the library.
- Devices** Specify the path name to the device; you cannot LOCK devices into memory.
- Fonts** Specify the path name to the exact font file to be loaded.
- Catalogs** Specify a path name as
 LOCALE:Catalogs/<language>/Sys/<catalog>.

The {<name>} option specifies the paths of the resources to load. The LOCK option tells the command to lock resources, such as libraries, fonts, and catalogs, into memory. This prevents the system from flushing the resource from RAM if memory is low. Although you can preload devices into memory using LOADRESOURCE, you cannot force them to stay in memory using the LOCK option. The UNLOCK option tells the command to unlock the resource from memory, allowing it to be flushed from RAM.

Entering LOADRESOURCE with no options lists all the LOCKed resources in RAM.

Example 1:

```
LOADRESOURCE LIBS:asl.library
```

loads asl.library into memory. The system can flush this library from RAM the next time it runs low on memory, unless the LOCK option is included in the command line.

Example 2:

```
LOADRESOURCE FONTS:topaz/11
```

loads the Topaz 11 font into memory.

Example 3:

```
LOADRESOURCE LOCALE:Catalogs/English/Sys/  
monitors.catalog
```

is a valid path name.

LOADWB

Starts Workbench.

Format LOADWB [-DEBUG] [DELAY] [CLEANUP]
 [NEWPATH]

Template -DEBUG/S,DELAY/S,CLEANUP/S,NEWPATH/S

Location C:

LOADWB starts the Workbench. Normally, this is in the Startup-sequence file that starts Workbench when booting. If you close the Workbench, LOADWB can restart it from a Shell.

The -DEBUG option makes a special developer menu, Debug, available in the Workbench menu bar. If the DELAY option is specified, LOADWB waits three seconds before executing to allow disk activity time to stop. The CLEANUP option automatically performs a cleanup of the initial disk window.

Workbench snapshots the current paths in effect when the LOADWB command is executed. It uses these paths for each Shell started from Workbench. NEWPATH allows you to specify a new path that is snapshot from the current Shell.

Example 1:

If you quit the Workbench and are working through a Shell, enter:

```
1> LOADWB
```

to return the Workbench. Entering LOADWB when the Workbench is already loaded has no effect.

Example 2:

```
1> PATH DF2:bin ADD
1> LOADWB NEWPATH
```

loads the Workbench. Any Shells started from the icon have the same path as the Shell used to run the LOADWB NEWPATH command.

LOCK

Sets the write-protect status of a device.

Format LOCK <drive> [ON | OFF] [<passkey>]

Template DRIVE/A,ON/S,OFF/S,PASSKEY

Location C:

LOCK sets or unsets the write-protect status of a device or partition. The LOCK remains on until the system is rebooted or until the LOCK is turned off with the LOCK OFF command.

An optional passkey can be specified. If the passkey is used to lock a hard disk partition, the same passkey must be specified to unlock the partition. The passkey can be any number of characters long.

Example:

```
1> LOCK Work: ON SecretCode
```

The Work partition is locked. You can read the contents of Work with commands such as DIR, LIST, or MORE, but you cannot alter the contents of the partition. If you try to edit the contents of a file on Work, a requester indicates that Work is write-protected. For example, if you try to create a new directory by entering the following:

```
1> MAKEDIR WORK:Test
```

the following message appears:

```
Can't create directory Work:Test
Disk is write-protected
```


To unlock the partition, enter:

```
1> LOCK Work: OFF SecretCode
```

Locking a device is only good for the duration of the current session. Resetting or turning off the Amiga cancels the lock.

MAGTAPE

Retensions, rewinds, or skips forward SCSI (Small Computer System Interface) tapes.

Format MAGTAPE [*DEVICE* <device name>] [*UNIT* <n>]
 [RET | RETENSION] [REW | REWIND] [*SKIP* <n>]

Template DEVICE/K,UNIT/N/K,RET=RETENSION/S,
 REW=REWIND/S,SKIP/N/K

Location C:

By default, MAGTAPE uses SCSI device unit 4. To change the default, you must use both the *DEVICE* and *UNIT* keywords.

The RET | RETENSION option runs the tape to the end and rewinds it. The REW | REWIND option rewinds the tape. The SKIP <n> option skips <n> files on the tape.

MAGTAPE tests to see if the unit is ready before sending the command. If your tape is not on-line, repeat the command.

Example:

```
1> MAGTAPE DEVICE second_scsi.device UNIT 0 REW
```

MAKEDIR

Creates a new directory.

Format MAKEDIR {<name>}

Template NAME/M

Location C:

MAKEDIR creates new, empty directories with the names you specify. The command works within only one directory level at a time, so any directories on the given paths must already exist. The command fails if a directory or a file of the same name already exists in the directory in which you attempt to create a new one.

MAKEDIR does not create a drawer icon for the new directory.

Example 1:

```
1> MAKEDIR Tests
```

creates a directory called Tests in the current directory.

Example 2:

```
1> MAKEDIR DF1:XYZ
```

creates a directory XYZ in the root directory of the disk in DF1:.

Example 3:

```
1> CD DF0:
1> MAKEDIR Documents Payables Orders
```

creates three directories on the disk in DF0:: Documents, Payables, and Orders.

For more examples using MAKEDIR, see Chapter 8.

MAKELINK

Creates a link between files.

Format MAKELINK [FROM] <file> [TO] <file> [HARD]
 [FORCE]

Template FROM/A,TO/A,HARD/S,FORCE/S

Location C:

MAKELINK creates a FROM file, known as a link, that is a pointer to another file, the TO file, on the disk. When an application or command calls the FROM file, the TO file is used. By default, MAKELINK supports hard links: the FROM file and TO file must be on the same volume.

Soft links, which can link across volumes, are not currently implemented.

Normally, MAKELINK does not support directory links. To create a directory link, you must use the FORCE option. If MAKELINK detects that you are creating a circular link, such as a link to a parent directory, a **Link loop not allowed** message is issued.

MOUNT

Makes a device connected to the system available.

Format MOUNT {device} [*FROM* <filename>]

Template DEVICE/M, FROM/K

Location C:

MOUNT reads a device's configuration parameters from a file. It then uses the parameter information to mount the devices or make them available to the system. Multiple devices can be mounted with a single command. The {device} argument specifies the names of the devices to be mounted.

MOUNT can process either DOSDrivers mount files or a traditional multiple-entry MountList file, depending on which of the following three ways the arguments are specified:

1. Given a device name, MOUNT tries to find a mount file of that name in DEVS:DOSDrivers, then in SYS:Storage/DOSDrivers, and finally as an entry in DEVS:MountList. This method is best if you have only one configuration for that device on your system.
2. Given a path, MOUNT looks for a mount file in that location. Wildcards may be used to mount multiple devices; as in **MOUNT DEVS:DOSDrivers/~(#?.info)**. Use this method when you have mount files stored somewhere other than the DOSDrivers drawers or if you have several mount files to process at once.
3. Given the FROM keyword and a path, MOUNT specifies the location of a MountList file to process. Use this method if you have a MountList stored somewhere other than DEVS: or if you have several MountLists.

Note A mount file's icon Tool Types, if any, override parameters of the same name in the mount file itself.

Example 1:

```
1> MOUNT PIPE:
```

This looks for the mount file DEVS:DOSDrivers/PIPE and processes it if found. If DEVS:DOSDrivers/PIPE does not exist, MOUNT looks for SYS:Storage/DOSDrivers/PIPE. If this also fails, then MOUNT looks for a PIPE: entry in DEVS:MountList.

Example 2:

```
1> MOUNT Work:Devices/PIPE
```

This looks for a PIPE mount file in Work:Devices.

Example 3:

```
1> MOUNT PIPE: FROM SYS:Mydevs/MountList
```

This scans for a PIPE: entry in SYS:Mydevs/MountList.

See Appendix B for further information on MountLists.

NEWCLI

Opens a new Shell window.

Format NEWCLI [<window specification>] [FROM
 <filename>]

Template WINDOW, FROM

Location Internal

NEWCLI starts a new Shell process. It is the same as using the NEWSHELL command.

NEWSHELL

Opens a new Shell window.

Format NEWSHELL [<window specification>] [FROM <filename>]

Template WINDOW, FROM

Location Internal

The new Shell window becomes the currently-selected window and process. The new window has the same current directory, prompt string, path, local environment variables, and stack size as the one from which it is invoked. However, each Shell window is independent, allowing separate input, output, and program execution.

The window can be sized, dragged, zoomed, and depth-adjusted like most other Amiga windows.

To create a custom window, you can include the <window specification> argument. Specify the initial dimensions, location, and title of the window with this <window specification> syntax:

```
CON:x/y/width/height/title/options
```

where:

- x** Is the number of pixels from the left edge of the screen to the left border of the Shell window. Use no value (//) to specify the minimum possible pixels.
- y** Is the number of pixels from the top of the screen to the top of the Shell window. Use no value (//) to specify the minimum possible pixels.
- width** Is the width of the Shell window, in pixels. Use no value (//) to specify the full width of the screen.
- height** Is the height of the Shell window, in pixels. Use no value (//) to specify minimum possible height.
- title** Is the text that appears in the Shell window title bar.

Use slashes to separate the parameters and options. If any spaces appear in the specification argument, the entire argument must be enclosed in double quotation marks ("").

The allowable options are:

- AUTO** The window automatically appears when the program needs input or produces output. With the Shell window, it opens for input immediately. The window can only be closed with the ENDSHELL command. Selecting the Shell's close gadget closes the window, but it re-opens immediately since it is expecting input.
- ALT** The window appears in the specified size and position when the zoom gadget is clicked. The four parameters must be separated with slashes (for example, ALT30/30/200/200).
- BACKDROP** The window appears on the backdrop, behind all the Workbench windows. This Shell window cannot be brought to the front of the screen; you have to resize the Workbench windows to see it.
- CLOSE** The window has all the standard gadgets, including a close gadget. This is the default for Shell windows, but you must specify it to get a standard Shell if you use the WINDOW argument.
- INACTIVE** The window opens, but is not made the active window.
- NOBORDER** The window opens without any left or bottom window border. Only the zoom, depth, and sizing gadgets are available.
- NOCLOSE** The window does not have a close gadget. If you open a console normally, there is no close gadget. If you open a console using the AUTO option, there is automatically a close gadget on the window.
- NODEPTH** The window has no window depth gadget.
- NODRAG** The window cannot be dragged. It has zoom, depth and sizing gadgets, but no close gadget.
- NOSIZE** The window only has a depth gadget.
- SCREEN** The window opens on a public screen. The screen must already exist. You must specify the name of the screen after the SCREEN keyword.
- SIMPLE** If you enlarge the window, the text expands to fill the newly available space, allowing you to see text that had been scrolled out of the window. This is the default for standard Shells.
- SMART** If you enlarge the window, the text does not expand to fill the newly available space. This saves memory.

WAIT The window can only be closed by selecting the close gadget or entering Ctrl+. If WAIT is the only option, there is no close gadget.

NEWSHELL uses the default startup file S:Shell-startup, unless a FROM file name is specified. S:Shell-startup is a standard AmigaDOS script file. For example, you can have several different Shell-startup files, each having different command aliases. You can call such customized Shell environments with FROM.

The NEWCLI command has the same effect as NEWSHELL; it invokes a new Shell process.

Example 1:

```
1> NEWSHELL
```

opens a new Shell window with the default window specification.

Example 2:

```
1> NEWSHELL "CON://640/200/My Shell/CLOSE"
```

A window starting in the upper left corner of the screen and measuring 640 pixels wide and 200 pixels high opens. The window is titled My Shell and it has a close gadget. The entire argument is enclosed in quotation marks because the title contains a space. If you add the command to your User-startup file, a Shell window opens automatically when your Amiga is booted.

Example 3:

```
1> NEWSHELL FROM S:Programming.startup
```

opens a new Shell, but instead of executing the Shell-startup file, the Programming.startup file is executed. You can have aliases and prompt commands in the Programming.startup file that are used only when you are programming.

For more examples using NEWSHELL, see Chapter 8.

PATH

Controls the directory list that the Shell searches to find commands.

Format PATH [{<dir>}] [ADD] [SHOW] [RESET] [REMOVE]
 [QUIET]

Template PATH/M,ADD/S,SHOW/S,RESET/S,REMOVE/S,
 QUIET/S,

Location Internal

PATH lets you see, add to, or change the search path that AmigaDOS follows when looking for a command or program to execute. When a directory is in the search path, you do not need to specify the complete path to any command within that directory. Entering the name alone makes AmigaDOS look through the directories in the search path until it finds the file.

Note The search path is only relevant when AmigaDOS is searching for a command or program to execute. Full path specifications are always necessary in arguments for commands such as COPY and DELETE.

Enter the PATH command alone or with the SHOW option to display directory names in the current search path. Normally, when PATH is displaying the directory names, a requester appears if a volume that is part of the search path cannot be found. For example, if you add a floppy disk to the search path and then remove that disk from the disk drive, a requester asks you to insert the disk.

If you specify the QUIET option, PATH does not display requesters for volumes that are not currently mounted. If PATH encounters an unmounted volume, it displays the message **device (or volume) is not mounted**. The names of any directories on that volume included in the PATH are not displayed.

The ADD option specifies directory names to be added to the current PATH. You can add any number of directories with one PATH ADD command (the ADD keyword is optional); names of the directories must be separated by at least one space. When you issue the PATH command, AmigaDOS searches for each of the ADDED directories.

To replace the existing search path with a new one, use **PATH RESET** followed by the names of the new directories. The existing search path, except for the current directory and **C:**, is erased and the new one is substituted.

The **REMOVE** option eliminates the named directory from the search path.

Example:

```
1> PATH EXTRAS:Tools ADD
```

adds the **Tools** directory in the **Extras** drawer to the search path of the Shell. If the **EXTRAS:** is not in a disk drive, a requester asks you to insert it in any drive.

If you remove **EXTRAS:** from the drive and enter:

```
1> PATH
```

a list of directories in the search path is displayed. A requester asks you to insert **EXTRAS:**. If you enter:

```
1> PATH QUIET
```

the list of directories in the search path is displayed. However, when the path comes to **Extras:Tools**, the error message appears in the list.

See also: **ASSIGN**. For more examples using **PATH**, see Chapter 8.

PROMPT

Changes the prompt string of the current Shell.

Format **PROMPT** [<prompt>]

Template **PROMPT**

Location Internal

PROMPT allows you to customize the prompt string, the text printed by the Shell at the beginning of a command line. The prompt string can contain any characters, including escape sequences.

This manual shows the prompt string as **1>**.

The default prompt string is:

```
"%N.%S> "
```

which displays the Shell number, a period, the current directory, a right angle-bracket, and a space. Entering PROMPT without a string argument resets the prompt to this default.

The substitutions available for the <prompt> string are:

%N	Displays the process number for the Shell.
%S	Displays the current directory.
%R	Displays the return code for the last operation.

A space is not automatically added to the end of the string. If you want a space between the prompt and typed-in text, place it in the string, and enclose the string in double quotation marks.

You can embed commands in the prompt string by enclosing the command in back apostrophes (').

Example 1:

```
1> PROMPT %N
1
```

Only the Shell number is shown. The > is removed from the prompt.

Example 2:

```
1> PROMPT "%N.%S.%R> "
1.Work:Anim.0>
```

The Shell number, current directory, and return code of the previous command are shown. A space is included after the >.

For more examples using the PROMPT command, see Chapter 8.

PROTECT

Changes the protection bits of a file or directory.

Format PROTECT [FILE] <file | pattern> [FLAGS][+|-]
[<flags>] [ADD | SUB] [ALL] [QUIET]

Template FILE/A,FLAGS,ADD/S,SUB/S,ALL/S,QUIET/S

Location C:

All files and directories have a series of protection bits (attributes) stored with them that control their properties. These bits can be altered to indicate the type of file and the operations permitted. PROTECT is used to set or clear the protection bits. For directories, only the d bit is significant.

The protection bits are represented by letters:

- s** The file is a script.
- p** The file is a pure command and can be made resident.
- a** The file has been archived.
- r** The file can be read.
- w** The file can be written to (altered).
- e** The file is executable (a program).
- d** The file or directory can be deleted. (Files within a delete-protected directory can still be deleted.)

Use the LIST command to see the protection bits associated with a file. The protection field is displayed with set (on) bits shown by their letters and clear (off) bits shown by hyphens. For example, a file that is readable, writable, and deletable has ----rw-d in the protection field.

To specify the entire protection field at the same time, enter the letters of the bits you want set as the FLAGS argument without any other keywords. The named bits are set and all the others are cleared.

The symbols + and - (or the equivalent keywords ADD and SUB) are used to control specific bits without affecting the state of unspecified bits. Follow + or - with the letters of the bits to set or clear, respectively, and only those bits are changed. There is no space after the symbol or between the letters. The order of the letters does not matter. ADD and SUB work similarly, but there must be a space between the keyword and the letters. You cannot both set and clear bits in the same command.

The ALL option adds or removes the specified protection bits from all the files and subdirectories matching the pattern entered. The QUIET option suppresses the screen output.

Example 1:

```
1> PROTECT DF0:Memo +rw
```

sets only the protection bits r (readable) and w (writable) of the file Memo on DF0:. No other protection bits are changed.

Example 2:

```
1> PROTECT L:#? e SUB
```

clears the e (executable) protection bit from all the files in the L: directory.

Example 3:

```
1> PROTECT Work:Paint rwd
```

The protection status of Paint becomes "----rwd".

QUIT

Exits from a script file with a specified return code.

Format QUIT [<return code>]

Template RC/N

Location Internal

QUIT stops the execution of the script at the specified return code. The default return code is zero. We recommend you use the standard return code values of 5, 10, and 20.

Example:

```
ASK "Do you want to stop now?"
IF WARN
  QUIT 5
ENDIF
ECHO "OK"
ECHO "The script is continuing."
```

If you press **Y** at the prompt, the script is aborted, since **WARN** is equal to a return code of 5. If you press **N** or press **Return**:

```
OK
The script is continuing.
```

is displayed in the Shell window.

RELABEL

Changes the volume name of the disk in the given drive to the specified name.

Format RELABEL [DRIVE] <drive> [NAME] <name>

Template DRIVE/A,NAME/A

Location C:

Volume names are set when disks are initially formatted. **RELABEL** allows you to change a disk's volume name to any name specified.

On floppy-only systems with one drive, be sure to specify the disks by volume name instead of drive name.

Examples:

```
1> RELABEL Workbench: MyDisk
```

changes the name of the **Workbench** disk to **MyDisk**. No colon is necessary after the second name.

```
1> RELABEL DF2: DataDisk
```

changes the name of the disk in **DF2:** to **DataDisk**.

REMRAD

Removes the recoverable RAM disk.

Format REMRAD [<device>] [FORCE]

Template DEVICE,FORCE/S

Location C:

REMRAD allows you to remove the recoverable RAM disk (usually mounted as RAD:) from memory without powering off the system. If you have mounted more than one recoverable RAM disk, use the DEVICE specification.

REMRAD instructs RAD: to delete all of its files and become inactive. However, the RAD: RAM_0 disk icon does not disappear. The next time the Amiga is rebooted, RAD: is removed from memory completely and the icon is no longer displayed.

If the device is in use when the REMRAD command is given, the operation aborts with a device in use message. To remove it if it is in use, you must use the FORCE option.

RENAME

Changes the name of or moves a file or directory.

Format RENAME [FROM] {<name>} [TO | AS] <name>

Template FROM/A/M,TO=AS/A,QUIET/S

Location C:

RENAME renames the FROM file or directory with the specified TO name. The FROM and TO files or directories must be on the same volume. If the name refers to a directory, RENAME changes the directory name without changing the names of the files or subdirectories in that directory. When there are multiple items in the FROM argument, the TO argument must be a directory.

If you rename a directory or if you use RENAME to give a file another directory name, AmigaDOS changes the position of that directory or file in the filing system hierarchy. This effectively moves the items.

Example 1:

```
1> RENAME Work/Ex1 AS :Test/Ex2
```

renames the file Ex1 as Ex2 and moves it from the Work directory to the Test directory. The Test directory must exist in the root directory for this command to work.

Example 2:

```
1> RENAME 3.doc 5.doc a.doc TO Docs
```

moves the 3.doc, 5.doc, and a.doc files to the Docs directory. The Docs directory must already exist.

REQUESTCHOICE

Allows AmigaDOS and ARexx scripts to use custom requesters.

Format REQUESTCHOICE <title> <body> [<gadgets>]
 [PUBSCREEN <public screen name>]

Template TITLE/A,BODY/A,GADGETS/A/M, PUBSCREEN/K

Location C:

The <title> argument specifies the title of the requester.

The <body> argument specifies the text of the requester. Linefeeds can be embedded using *N.

The <gadgets> argument specifies the text for the different gadgets. The gadget labels are separated with spaces.

The number of the selected gadget is printed as a result to the console. For evaluation in a script file, you can redirect this output into an environment variable. If the requester cannot be opened, the command generates a return code of 20.

The PUBSCREEN argument allows the requester to open its window on a public screen.

Example:

```
1> RequestChoice >ENV:rcnum "New Title" "This is my  
    requester*nSelect a gadget" "OK" "Maybe"  
    "Cancel"
```

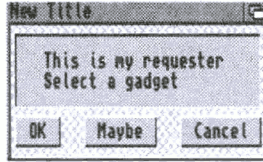


Figure 6-1. Sample RequestChoice Requester

ENV:rcnum contains 0, 1, or 2 after a gadget is selected. The script can use this value to control its later execution.

REQUESTFILE

Allows AmigaDOS and ARexx scripts to use a file requester.

Format REQUESTFILE [DRAWER <drawer name>] [*FILE*
 <file>] [*PATTERN* <pattern>] [*TITLE* <title>]
 [*POSITIVE* <text>] [*NEGATIVE* <text>]
 [*ACCEPTPATTERN* <pattern>]
 [*REJECTPATTERN* <pattern>] [SAVEMODE]
 [MULTISELECT] [DRAWERONLY] [NOICONS]
 [*PUBSCREEN* <public screen name>]

Template DRAWER,/FILE/K,PATTERN/K,TITLE/K,
 POSITIVE/K,NEGATIVE/K,
 ACCEPTPATTERN/K,REJECTPATTERN/K,
 SAVEMODE/S,MULTISELECT/S,
 DRAWERONLY/S,NOICONS/S,PUBSCREEN/K

Location C:

When entered with no arguments, a file requester with OK, Volumes, Parent, and Cancel buttons is created. Its Drawer and File gadgets are empty and it displays the contents of the current directory.

The DRAWER argument specifies the initial contents of the Drawer gadget.

The FILE option specifies the initial contents of the File gadget.

The PATTERN option allows the use of a standard AmigaDOS pattern. It includes a Pattern gadget in the requester and specifies

the initial contents of the gadget. If this option is not provided, the file requester does not have any Pattern gadget.

The **TITLE** option specifies the title of the requester.

The **POSITIVE** option specifies the text to appear in the positive (left) choice in the file requester.

The **NEGATIVE** option specifies the text to appear in the negative (right) choice in the file requester.

The **ACCEPTPATTERN** option specifies a standard AmigaDOS pattern. Only files matching this pattern are displayed in the file requester.

The **REJECTPATTERN** option specifies a standard AmigaDOS pattern. Files matching this pattern are not displayed in the file requester.

If **SAVEMODE** is specified, the requester is used for writing files to disk. If **MULTISELECT** is specified, the requester allows multiple files to be selected at once. If **DRAWERSONLY** is specified, the requester does not have a File gadget. This effectively turns the file requester into a directory requester. If **NOICONS** is specified, the requester does not display icons (.info files).

The selected files are returned on the command line, enclosed in double quotation marks and separated with spaces. The command generates a return code of 0 if you select a file or 5 if you cancel the requester.

The **PUBSCREEN** argument allows the requester to open its window on a public screen.

Example:

```
1> REQUESTFILE DRAWER Devs: TITLE "My Req" NOICONS
```

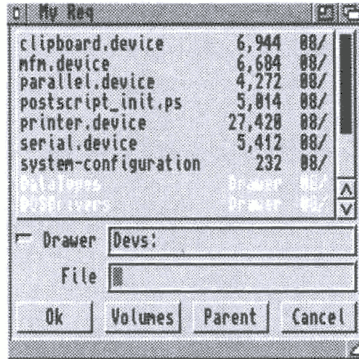


Figure 6-2. Sample RequestFile Requester

RESIDENT

Displays and modifies the list of resident commands.

Format RESIDENT [<resident name>] [<filename>]
[REMOVE] [ADD] [REPLACE] [PURE | FORCE]
[SYSTEM]

Template NAME,FILE,REMOVE/S,ADD/S,
REPLACE/S,PURE=FORCE/S,SYSTEM/S

Location Internal

RESIDENT loads a command into memory and adds it to the resident list maintained by the Shell. This allows the command to be executed without reloading it from disk each time. If RESIDENT is invoked with no options, it lists the programs on the resident list.

To be made resident, a command should be pure, meaning that it is both re-entrant and re-executable. A re-entrant command can properly support independent use by two or more programs at the same time. A re-executable command does not have to be reloaded to be executed again. Commands that have these characteristics are called pure and have the p (pure) protection bit set.

The following commands cannot be made resident: BINDDRIVERS, CONCLIP, IPREFS, LOADRESOURCE, LOADWB, and SETPATCH.

LIST the C: directory to check for the presence of the p protection bit to determine which commands are pure.

Many of the commands in the C: directory, as well as the MORE command in Utilities, are pure commands and can be made resident. If a command does not have its pure bit set, it probably cannot be made resident safely. (Setting the pure bit does not make a command or program pure).

The REPLACE option is the default option and does not need to be explicitly stated. If no <resident name> is specified (for example, only a file name is specified), RESIDENT uses the file name portion as the name on the resident list. The full path to the file must be used.

If a <resident name> is specified and RESIDENT finds a program with that name already on the list, it attempts to replace the command. That <resident name> must be used to reference the resident version of the command. The replacement succeeds only if the already-resident command is not in use.

To override REPLACEMENT and make several versions of a command resident simultaneously, use the ADD option, giving a different <resident name> for each version loaded.

If the SYSTEM option is specified, the command is added to the system portion of the resident list and becomes available as a system component. Any commands added to the resident list with the SYSTEM option cannot be removed. To list these files on the RESIDENT list, you must specify the SYSTEM option.

The PURE option forces RESIDENT to load commands that are not marked as pure and use them to test the pureness of other commands and programs. Use the PURE option with caution. Be sure the programs that you make RESIDENT meet the criteria to be resident or be careful to use the command in only one process at a time.

The availability of internal commands can also be controlled with RESIDENT. To deactivate an Internal command (for example, if an application has its own command of the same name), use RESIDENT <command> REMOVE. The command can be reactivated with the REPLACE option.

Example 1:

```
1> RESIDENT C:COPY
```

makes the COPY command resident (replaces any previous version).

Example 2:

```
1> RESIDENT Copy2 DF1:C/COPY ADD
```

adds another version of COPY to the resident list, under the name Copy2.

Example 3:

```
1> RESIDENT Xdir DF1:C/Xdir PURE
```

makes an experimental, non-pure version of the DIR command resident.

Example 4:

```
1> RESIDENT CD REMOVE
```

makes the Internal CD command unavailable.

Example 5:

```
1> RESIDENT CD REPLACE
```

restores the CD command to the system.

See also: PROTECT, LIST.

RUN

Executes commands as background processes.

Format RUN <command...> [{+ <command>}]

Template COMMAND/F

Location Internal

RUN is used to start background processes. A background process does not open its own window for input or output and does not take over the parent Shell.

RUN attempts to execute the <command> and any arguments entered on the command line. You can RUN multiple command lines by separating them with plus signs (+). If you press Return after a

plus sign, RUN interprets the next line as a continuation of the same command line.

To make it possible to close the Shell window in which the process was started, redirect the output of RUN with **RUN >NIL:**
<command>.

A new background Shell has the same search path and command stack size as the Shell from which RUN is given.

You can RUN commands stored on the resident list. Resident commands are checked before commands in the command path. A Shell started with RUN NEWSHELL uses the default startup file, S:Shell-startup.

Example 1:

```
1> RUN COPY Text TO PRT:+  
DELETE Text +  
ECHO "Printing finished"
```

prints the Text file by copying it to the printer device, deletes it, then displays the given message. Plus signs string together the command lines, causing each command to be run after the previous command finishes.

Example 2:

```
1> RUN EXECUTE Comseq
```

executes, in the background, all the commands in the script file Comseq.

For more examples using the RUN command, see Chapter 8.

SEARCH

Looks for the specified text string in the files of the specified directories.

Format SEARCH [FROM] {<name | pattern>} [SEARCH]
 <string | pattern> [ALL] [NONUM] [QUIET]
 [QUICK] [FILE] [PATTERN]

Template FROM/M,SEARCH/A,ALL/S,NONUM/S,
 QUIET/S,QUICK/S,FILE/S,PATTERN/S

Location C:

SEARCH looks through all the files in the FROM directory for the string given in the SEARCH string. (The FROM and SEARCH keywords are optional.) If the ALL switch is given, SEARCH also looks through all the subdirectories of the FROM directory. SEARCH displays the name of the file being searched and any line that contains the text sought. You must place quotation marks around any search text containing a space. The search is not case-sensitive.

The options are:

NONUM Line numbers are not printed with the strings.

QUIET Searches quietly; file names being searched are not displayed.

QUICK Uses a more compact output format.

FILE Looks for a file by the specified name, rather than for a string in the file.

PATTERN Uses pattern matching to search for the string.

SEARCH leaves a 0 in the condition flag if the object is found, and a 5 (WARN) otherwise. To abandon the search of the current file and continue to the next file, if any, press Ctrl+D. SEARCH is aborted when Ctrl+C is pressed.

Examples

```
1> SEARCH DEVS: DOSDrivers globvec
   DOSDrivers (dir)
   PIPE..
6  GlobVec      =-1
   PIPE.info
```

```
1> SEARCH Utilities #?.info FILE
Workbench:Utilities/Clock.info
Workbench:Utilities/MultiView.info
```

SET

Sets a local variable.

Format SET [<name>] [<string...>]

Template NAME,STRING/F

Location Internal

SET with no arguments lists the current local variables.

SET with <name> and <string> arguments creates a new environment variable. The first word after SET is taken as the <name>. Everything else on the command line is taken as the <string> argument. Quotation marks are not required.

An environment variable created with SET is local to the Shell in which it was created. If you create a new Shell with the NEWSHELL command, that Shell also recognizes any variables created in its parent Shell. However, if you create a new Shell with the Execute Command Workbench menu item or by opening the Shell icon, variables created with SET are not recognized in the new Shells.

You can call environment variables in a script or on a command line by placing a dollar sign (\$) in front of the variable name.

To remove a local variable definition, use the UNSET command.

Examples:

```
1> SET Origin This process launched from icon
creates the local variable Origin that stores a reminder that a Shell
was invoked from an icon rather than a NEWSHELL.
```

```
1> ECHO $Origin
This process launched from icon
```

See also: GET, UNSET

SETCLOCK

Sets or reads the battery backed-up hardware clock.

Format SETCLOCK LOAD | SAVE | RESET

Template LOAD/S,SAVE/S,RESET/S

Location C:

SETCLOCK SAVE sets the date and time of the battery backed-up hardware clock (if your system has one) from the current system time, which is set with the Time editor or with the DATE command. SETCLOCK SAVE is typically used after a DATE command.

SETCLOCK LOAD sets the current system time from the battery backed-up clock. In systems using AmigaDOS Release 2 or later, this is done automatically during the boot process.

The RESET option resets the clock completely. Use this option if the clock is accidentally turned off or LOAD and SAVE do not appear to work correctly.

Example:

```
1> DATE 22-JAN-93 07:15:25
1> SETCLOCK SAVE
```

saves the date, January 22, 1993, and the time, 7:15 a.m., to the battery backed-up hardware clock. When the system is booted, the system clock is set with the time saved in the hardware clock.

Some Amiga models do not have battery backed-up clocks unless an expansion unit has been installed.

See also: DATE

SETDATE

Change the timestamp of a file or directory.

Format SETDATE <file | pattern> [<weekday>] [<date>]
 [<time>] [ALL]

Template FILE/A,WEEKDAY,DATE,TIME,ALL/S

Location C:

SETDATE changes the timestamp, the date and time of the creation or last change, of a file or directory. SETDATE <file> changes the date/time of the file to the current system date/time. SETDATE ALL changes the date and time of all the files and subdirectories matching the pattern entered.

The system clocks are not affected by SETDATE.

You can use output from the DATE command as input to SETDATE.

Example 1:

```
1> SETDATE TestFile
```

changes the date and time associated with TestFile to the current date and time.

Example 2:

```
1> SETDATE TestFile 01-04-91 13:45:32
```

Changes the date and time associated with TestFile to April 1, 1991, 1:45 p.m.

See also: DATE

SETENV

Sets a global variable.

Format SETENV [<name>] [<string...>]

Template NAME,STRING/F

Location Internal

SETENV with no arguments lists the current global variables.

SETENV with <name> and <string> arguments creates a new global environment variable. The first word after SETENV is taken as the <name>. Everything else on the command line is taken as the <string> argument. Quotation marks are not required.

Global variables are stored in the ENV: directory and are available to all processes. However, if a local variable (defined by SET) and a global variable share the same name, the local variable is used.

Environment variables are called by scripts or other commands by including a dollar sign (\$) in front of the variable name.

To remove a global variable definition, use the UNSETENV command.

Example 1:

```
1> SETENV Editor Extras:Tools/MEmacs
```

creates the environment variable Editor that can be used with the MORE utility. This specifies the editor as MEMacs, located in the Tools drawer of EXTRAS:. The variable Editor is available in any Shell.

Example 2:

```
1> SETENV Editor C:ED
```

same as above, only the editor specified is ED.

```
1> ECHO $Editor
C:ED
```

See also: GETENV, UNSETENV

SETFONT

Changes the font of the current Shell.

Format SETFONT <size> [SCALE] [PROP] [ITALIC]
 [BOLD] [UNDERLINE]

Template NAME/A,SIZE/N/A,SCALE/S,PROP/S,
 ITALIC/S,BOLD/S,UNDERLINE/S

Location C:

SETFONT lets you change the font used in a particular Shell window, overriding the System Default Text setting specified in the Font editor. SETFONT is only effective in the window in which it is invoked.

You must specify both a font name and a size when using the SETFONT command. The other options are:

SCALE Enables bitmap font scaling.
PROP Allows proportional fonts.
ITALIC The font is italic.
BOLD The font is boldface.
UNDERLINE The font is underlined.

Invoking SETFONT clears the Shell window of its current contents and displays a new prompt, in the new font, at the top of the window. Using proportional fonts in a Shell window is not recommended because the variable character spacing prevents columns of information from lining up and makes editing the command line difficult.

Example:

```
1> SETFONT topaz 11 BOLD UNDERLINE
```

The Shell window clears and the new prompt is in 11 point Topaz, underlined and boldface.

SETKEYBOARD

Sets the keymap for the Shell.

Format SETKEYBOARD <keymap name>

Template KEYMAP/A

Location C:

SETKEYBOARD specifies the keymap used by the current Shell. The available files are listed below:

Keymap	Keyboard
cdn	Canadien Français
ch1	Suisse
ch2	Schweiz
d	Deutsch
dk	Dansk
e	Español
f	Français
gb	British
i	Italiana
n	Norsk
po	Português
s	Svenskt
usa	American
usa2	Dvorak

To specify the same permanent keymap, use the Preferences Input editor to save your choice.

Example:

To change to a French Canadian keymap, enter:

```
1> SETKEYBOARD cdn
```

The keymap file must be in the **KEYMAPS:** directory for **SETKEYBOARD** to find it.

SKIP

Skips to a label when executing script files.

Format SKIP [<label>] [BACK]

Template LABEL,BACK/S

Location Internal

SKIP is used in scripts to allow you to skip ahead in the script to a <label> defined by a LAB statement. If no <label> is specified, SKIP jumps to the next LAB statement.

SKIP always searches forward from the current line of the file. However, when the BACK option is used, SKIP starts searching for the label from the beginning of the file. This allows SKIPS to point prior to the SKIP command.

You can only SKIP as far back as the last EXECUTE statement. If there are no EXECUTE statements in a script, you SKIP back to the beginning of the file.

If SKIP does not find the label specified, the command sequence terminates and the message **Label <label> not found by skip** is displayed.

Example:

Assume you have the following script, called CheckFile:

```
.KEY name
IF exists <name>
  SKIP message
ELSE
  ECHO "<name> is not in this directory."
  QUIT
ENDIF
LAB message
ECHO "The <name> file exists."
```

You can run the script by entering:

```
1> EXECUTE CheckFile Document
```

If the Document file exists in the current directory, the execution of the script SKIPS ahead to the LAB command. The message:

```
The Document file exists.
```

is displayed in the Shell window.

If the Document file is not in the current directory, the execution of the script jumps to the line after the ELSE statement, displaying the message:

```
Document is not in this directory.
```

See also: EXECUTE, LAB. For more examples using the SKIP command, see Chapter 8.

SORT

Alphabetically sorts the lines of a file.

Format SORT [FROM] <file | pattern> [TO] <filename>
 [COLSTART <n>] [CASE] [NUMERIC]

Template FROM/A,TO/A,COLSTART/K,CASE/S,NUMERIC/S

Location C:

SORT sorts the FROM file alphabetically, line-by-line, sending the sorted results to the TO file. SORT assumes the file is a normal text file in which lines are separated by line feeds. SORT normally disregards case. If the CASE switch is given, upper-cased items are output first.

The COLSTART keyword specifies the character column at which the comparison begins. SORT starts comparing the lines from that point, wrapping around to the beginning of the line if the compared lines match to the end.

When the NUMERIC option is specified, the lines are interpreted as numbers from the first column reading to the right, stopping at the first non-numeric character. Lines not beginning with numbers are treated as 0. The lines are output in numerical order. CASE is ignored when NUMERIC is specified.

Example:

```
1> SORT DF0:Glossary TO DF0:Glossary.alpha
```

sorts the lines in the Glossary file, arranges them alphabetically, and outputs them to a new file called Glossary.alpha. The case of the words is disregarded.

For more examples using the SORT command, see Chapter 8.

STACK

Displays or sets the stack size within the current Shell.

Format STACK [[SIZE] <stack size>]

Template SIZE/N

Location Internal

A Shell uses a certain amount of stack, a special area in memory allocated for it. Each Shell has a specific stack size. If a program causes a system failure, changing the Shell's stack size may solve the problem. Commands performing operations consisting of multiple levels can require additional stack space.

Stack sizes typically range from 4096 to 40000 bytes. If the stack size is too small, a system failure can occur. If the stack size is too large, it can use too much memory.

Note A software failure message is displayed if you run out of stack space. Increase the stack size for the Shell that caused the error.

Entering the STACK command with no arguments displays the current stack size.

STATUS

Lists information about Shell processes.

Format STATUS [<process>] [FULL] [TCB] [CLI|ALL]
 [COM|COMMAND <command>]

Template PROCESS/N,FULL/S,TCB/S,CLI=ALL/S,
 COM=COMMAND/K

Location C:

STATUS without any arguments lists the numbers of the current Shell processes and the program or command running in each. The <process> argument specifies a process number, limiting STATUS to giving information about that process only.

For information on the stack size, global vector size, priority, and the current command for each process, use the FULL keyword. The TCB keyword is similar, omitting the command information. The CLI=ALL keyword gives only the command information.

STATUS searches for a command when you use the COMMAND option. STATUS scans the Shell list, looking for the specified <command>. If the command is found, the Shell's process number is output, and the condition flag is set to 0. Otherwise, the flag is set to 5 (WARN).

Example 1:

```
1> STATUS 1
Process 1: Loaded as command: status
```

Example 2:

```
1> STATUS 1 FULL
Process 1: stk 4000, gv 150, pri 0 Loaded as
command: status
```

Example 3:

```
1> STATUS >RAM:XYZ COMMAND=COPY
1> BREAK <RAM:XYZ >NIL: ?
```

sends a break to the process executing COPY.

TYPE

Displays the contents of a file.

Format TYPE {<file | pattern>} [TO <name>] [OPT H|N]
 [HEX | NUMBER]

Template FROM/A/M,TO/K,OPT/K,HEX/S,NUMBER/S

Location C:

TYPE outputs the contents of the named file to the current window if no destination is given or to a specified output file. The TO keyword types information to a specified file. If more than one file name is specified, the file names are typed in sequence.

The OPT H and OPT N options are also available by the HEX and NUMBER keywords, respectively. However, the two options are mutually exclusive. The HEX option types the file as columns of hexadecimal numbers, with an ASCII character interpretation column. The NUMBER option numbers the lines as they are output.

To pause output, press the Space bar. To resume output, press Backspace, Return, or Ctrl+X. To stop output, press Ctrl+C (****B**reak is displayed).

Example:

```
1> TYPE S:Startup-sequence
```

The contents of the Startup-sequence file in the S: directory are displayed on the screen.

For more examples using TYPE, see Chapter 8.

UNALIAS

Removes an alias.

Format UNALIAS [<name>]

Template NAME

Location Internal

UNALIAS removes the named alias from the alias list. With no arguments, UNALIAS lists the current aliases.

See also: ALIAS

UNSET

Removes a local variable.

Format UNSET [<name>]

Template NAME

Location Internal

UNSET removes the named local variable from the variable list for the current process. With no arguments, UNSET lists the current variables.

See also: SET

UNSETENV

Removes a global variable.

Format UNSETENV [<name>]

Template NAME

Location Internal

UNSETENV removes the named global variable from the current variable list. With no arguments, UNSETENV lists the current variables.

See also: SETENV

VERSION

Finds software version and revision numbers.

Format VERSION [<library | device | file>] [<version #>]
[<revision #>] [<unit #>] [FILE] [INTERNAL] [RES]
[FULL]

Template NAME,VERSION/N,REVISION/N,UNIT/N,
FILE/S,INTERNAL/S,RES/S,FULL/S

Location C:

VERSION finds the version and revision number of a library, device, command, or Workbench disk. VERSION can also test for a specific version/revision and set the condition flags if the version/revision is greater.

VERSION with no <library | device | file> argument prints the Kickstart version number and the Workbench version number and sets the two corresponding environment variables. If a name is specified, VERSION attempts to open the library, device, drive, or file

and read the version information. Specify a device name, such as DF0: or DH0:, to get the version of the file system used by a drive.

When a <version #> or a <revision #> is specified, VERSION sets the condition flag to 0 if the version and revision number of the Kickstart, library, or device driver is greater than or equal to the specified values. Otherwise, the flag is set to 5 (WARN). If a revision number is not specified, no comparison on the revision number is performed.

The <unit #> option is obsolete and is retained for compatibility with older programs.

The FILE option forces VERSION to ignore libraries or device drivers currently loaded. This allows you to get the version number of a .library or .device file on disk when a library or device of that name is already in memory or available in LIBS:. The RES option gets the version of Resident commands. Built-in Shell commands have the same version string as the Shell. INTERNAL is also obsolete and retained for compatibility. The FULL option prints out the complete version of the string, including the date.

Examples:

```
1> VERSION
Kickstart 39.92 Workbench 39.1

1> VERSION Alpha:Libs/xyz.library FILE FULL
xyz.library 1.13 (05/24/93)
```

WAIT

Waits for the specified time.

Format WAIT [<n>] [SEC | SECS | MIN | MINS] [UNTIL
<time>]

Template /N,SEC=SECS/S,MIN=MINS/S,UNTIL/K

Location C:

WAIT is used in command sequences or after RUN to wait for a certain period of time or until a specific time. The default waiting period is one second.

The <n> argument specifies the number of seconds or minutes to wait. These options are mutually exclusive; you can only enter seconds or minutes.

Use the keyword UNTIL to wait until a particular time of the day, given in the format HH:MM.

Example 1:

```
1> WAIT 10 MINS
```

waits ten minutes.

Example 2:

```
1> WAIT UNTIL 21:15
```

waits until 9:15 p.m.

WHICH

Searches the command path for a particular item.

Format WHICH <command> [NORES] [RES] [ALL]

Template FILE/A,NORES/S,RES/S,ALL/S

Location C:

WHICH lets you find a specific command, program, or directory by entering its name. If the named item is in the search path, WHICH displays the complete path to that item. WHICH lists resident commands as RESIDENT and internal commands as INTERNAL.

Normally, WHICH searches the resident list, the current directory, the command paths, and the C: directory. If the item is not found, WHICH sets the condition flag to 5 (WARN), but does not print any error message.

If the NORES option is specified, the resident list is not searched. If the RES option is specified, only the resident list is searched.

The ALL switch continues the search through the full search path, finding and listing all locations of a command or program. It can, however, lead to multiple listings of the same command if that

command is reached by more than one route (such as C: and the current directory).

Examples:

```
1> WHICH avail
C:Avail
```

```
1> WHICH C:
Workbench:C
```

```
1> WHICH alias
INTERNAL alias
```

WHY

Prints an error message explaining why the previous command failed.

Format WHY

Template (none)

Location Internal

When a command fails, the screen displays a brief message. This message typically includes the name of the file, if that was the problem, but provides no details. If the reason for a failure is not evident, enter WHY for a more complete explanation.

System Commands

System commands are required for normal system operation. They are used by the standard Startup-sequence or called automatically by the system for applications. The user does not typically invoke these commands.

ADDDATATYPES

Builds a list of data types that datatypes.library can understand.

Format ADDDATATYPES [FILES] {filenames} [QUIET]
 [REFRESH]

Template FILES/M,QUIET/S,REFRESH/S

Location C:

Data type descriptors are stored in DEVS:DataTypes. These descriptors allow programs such as MultiView to interpret different data file types. ADDDATATYPES can also be called by application installation scripts to add their own data types to the list.

The FILES argument specifies the names of the data type descriptors to add to the existing list of data type descriptors.

Specifying the QUIET option suppresses error and output messages.

Specifying the REFRESH option scans the DEVS:DataTypes directory for new or changed data type descriptors.

BINDDRIVERS

Binds device drivers to hardware.

Format BINDDRIVERS

Template (none)

Location C:

BINDDRIVERS loads and runs device drivers for add-on hardware. These devices are automatically configured by the expansion library if their device drivers are in the SYS:Expansion directory.

The BINDDRIVERS command must appear in the Startup-sequence file to configure the hardware when the system is booted.

CONCLIP

Moves data between console windows and the Clipboard.

Format CONCLIP [CLIPUNIT | UNIT <unit number>] [OFF]

Template CLIPUNIT=UNIT/N,OFF/S

Location C:

CONCLIP is called from the standard Startup-sequence. It keeps track of the information that has been cut to the Clipboard.

The CLIPUNIT option allows you to specify the clipboard.device unit number to use. Specify any unit from 0 to 255. The default number is 0. We recommend that this option be used only by advanced users or programmers who wish to use different units for different data, such as one for text and another for graphics. Run the command from the Shell, specifying the new unit number. The next time you copy and paste, that Clipboard unit is used.

Using the OFF option with Shell, MEMacs, and ED causes these commands to stop interacting with the system Clipboard during cutting and pasting operations. We recommend that you do not use this option.

IPREFS

Communicates Preferences information stored in the individual editor files to the operating system.

Format IPREFS

Template (none)

Location C:

IPREFS reads the individual system Preferences files and passes the information to the system. IPREFS is generally run in the Startup-sequence after the Preferences files are copied to ENV:. Each time a user selects Save or Use from within an editor, IPREFS is notified

and passes the information to the system. If necessary, IPREFS resets Workbench to implement those changes. If any Shell, project, or tool windows are open, IPREFS displays a requester asking you to close them.

SETPATCH

Makes ROM patches in system software.

Format SETPATCH [QUIET] [NOCACHE] [REVERSE]

Template QUIET/S,NOCACHE/S,REVERSE/S

Location C:

SETPATCH installs temporary modifications to the operating system. It must be run at the beginning of the Startup-sequence file. Updated versions of SETPATCH are made available when necessary as AmigaDOS development continues.

If QUIET is specified, no output is sent to the screen.

NOCACHE prevents data caching from being activated on some 68030 and 68040 systems.

REVERSE stores patches in reverse order. This option is useful for CDTV developers only.

Chapter 7

Workbench-Related Command Reference

The commands described in this chapter are command line equivalents of running Workbench programs. They are divided into the following functional categories:

- Preferences editors
- Commodities programs
- Other Workbench related tools and programs

These command groupings have been made for documentation purposes only.

Note A full description for using all of the Workbench editors, tools, and programs can be found in the *Workbench User's Guide*.

The following table provides a quick alphabetical reference to all of the commands in this chapter, their purpose, and the pages on which they appear:

Command	Purpose	Page
AutoPoint	Automatically selects any window the pointer is over.	7-13
Blanker	Causes the monitor screen to go blank if no input is received within a specified time.	7-14
Calculator	Provides an on-screen calculator.	7-18

Command (cont'd)	Purpose (cont'd)	Page (cont'd)
ClickToFront	Allows you to bring a window to the front of the screen by use of mouse clicks.	7-15
Clock	Provides an on-screen clock.	7-19
CMD	Redirects printer output to a file.	7-20
CrossDOS	Sets text filter and conversion options for CrossDOS devices.	7-15
DiskCopy	Copies the contents of one disk to another.	7-21
Exchange	Monitors and controls the Commodity Exchange programs.	7-16
FixFonts	Updates the .font files in the FONTS: directory.	7-22
FKey	Assigns commands to function keys.	7-16
Font	Specifies the fonts used by the system.	7-5
Format	Formats a disk for use with the Amiga.	7-23
GraphicDump	Prints the frontmost screen.	7-25
IconEdit	Edits the appearance and type of icons.	7-26
IControl	Specifies parameters used by the Workbench.	7-5
InitPrinter	Initializes a printer for print options specified in the Preferences editors.	7-26
Input	Specifies different speeds for the mouse and keyboard and selects a national keyboard.	7-6
Intellifont	Manages Intellifont outline fonts.	7-26
KeyShow	Displays the current keymap.	7-27
Locale	Allows the choice of language for interacting with the system.	7-6
MEmacs	Enables screen-oriented text editing.	7-27
More	Displays the contents of an ASCII file.	7-27
MouseBlanker	Removes the mouse pointer from the screen while entering input from the keyboard.	7-17
MultiView	Displays picture files, text files, AmigaGuide files, sound files, and animated graphics files.	7-29

Command (cont'd)	Purpose (cont'd)	Page (cont'd)
NoCapsLock	Disables the Caps Lock key.	7-17
NoFastMem	Forces the Amiga to use only Chip RAM.	7-32
OverScan	Changes the sizes of the display areas for text and graphics.	7-7
Palette	Changes the colors of the Workbench screen.	7-7
Pointer	Changes the appearance of the mouse pointer.	7-8
PrepCard	Prepares PCMCIA memory cards for use as disk or RAM.	7-32
Printer	Specifies a printer and basic print options.	7-8
PrinterGfx	Specifies graphic printing options.	7-9
PrinterPS	Specifies PostScript printing options.	7-9
ScreenMode	Selects a display mode.	7-10
Serial	Sets the specifications for communication through the serial port.	7-10
Sound	Controls the type of sound used for the display beep.	7-11
Time	Sets the system clock.	7-11
WBPattern	Creates background patterns for the Workbench and windows.	7-12

Preferences Editors

The commands listed in this section invoke the Workbench Preferences editors.

The same arguments and switches appear within many of the Preferences Editors command format statements. These have the same meaning for each command that uses them and are described as follows:

Argument	Meaning
[FROM <filename>]	Specifies a Preferences preset file to open. This file must be previously saved with the given editor's Save As menu item. These files normally have the .pre extension and are stored in the Presets drawer.
[EDIT]	Opens the editor. This is the default if you enter the editor name alone.
[USE]	Uses the settings in the FROM file without opening the editor.
[SAVE]	Saves the settings in the FROM file as the default without opening the editor.
[PUBSCREEN <public screen name>]	Allows the editor to open its window on a public screen.
[UNIT]	Causes an additional text gadget to appear in the editor for setting the default unit number.
[CLIPUNIT <clipboard unit>]	Determines which Clipboard unit to use during cut and paste operations.
[NOREMAP]	Turns off color mapping so that the system displays picture files using the colors with which they were saved.

Font

Specifies the fonts used by the system.

Format FONT [FROM <filename>][EDIT | USE | SAVE]
 [PUBSCREEN <public screen name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

Example:

```
1> FONT
```

Opens the Font editor, the same as double-clicking on the Font icon.

IControl

Specifies parameters used by the operating system.

Format ICONTROL [FROM <filename>]
 [EDIT | USE | SAVE] [PUBSCREEN <public screen
 name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

Example:

```
1> ICONTROL Prefs/Presets/IControl.pre
```

Opens the IControl editor and loads the settings saved in the IControl.pre preset file for editing.

Input

Specifies different speeds for the mouse and keyboard and selects a national keyboard.

Format INPUT [FROM <filename>][EDIT | USE | SAVE]
[PUBSCREEN <public screen name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

Example:

```
1> INPUT Prefs/Presets/Input.fast USE
```

loads and sets the settings from the Input.fast preset without opening the editor. If the system is rebooted, the previously saved default settings are used.

Locale

Allows you to choose the languages available on the system.

Format LOCALE [FROM <filename>][EDIT | USE | SAVE]
[PUBSCREEN <public screen name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

Example:

```
1> LOCALE Prefs/Presets/Locale.UK SAVE
```

Loads the settings from the Locale.UK preset and saves them as the default without opening the editor. The system retains the Locale.UK settings after rebooting.

Overscan

Changes the sizes of the display areas for text and graphics.

Format OVERSCAN [FROM <filename>]
 [EDIT | USE | SAVE] [*PUBSCREEN* <public screen
 name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

Example:

```
1> OVERSCAN PUBSCREEN MyBench
```

Opens the Overscan editor on the public screen named MyBench.

Palette

Changes the colors of the Workbench screen.

Format PALETTE [FROM <filename>] [EDIT | USE | SAVE]

Template FROM,EDIT/S,USE/S,SAVE/S

Location Extras:Prefs

Pointer

Changes the appearance of the mouse pointer.

Format	POINTER [FROM <filename>] [EDIT USE SAVE] [CLIPUNIT <clipboard unit>] [NOREMAP]
Template	FROM,EDIT/S,USE/S,SAVE/S,CLIPUNIT/K/N, NOREMAP/S
Location	Extras:Prefs

Example:

```
1> POINTER CLIPUNIT 1
```

Opens the Pointer editor, setting the Clipboard unit to 1. This is useful if the default Clipboard unit (0) is already in use.

Printer

Specifies a printer and print options.

Format	PRINTER [FROM <filename>][EDIT USE SAVE] [PUBSCREEN <public screen name>] [UNIT]
Template	FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K, UNIT/S
Location	Extras:Prefs

Example:

```
1> PRINTER Prefs/Presets/Printer.post UNIT
```

Opens the Printer editor, loading the Printer.post preset. A text gadget appears in the editor window so that the default printer unit can be specified for the preset.

PrinterGfx

Specifies graphic printing options.

Format PRINTERGFX [FROM <filename>]
 [EDIT | USE | SAVE] [*PUBSCREEN* <public screen
 name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

PrinterPS

Controls the features of PostScript printers.

Format PRINTERPS [FROM <filename>]
 [EDIT | USE | SAVE] [*PUBSCREEN* <public screen
 name>]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

This editor only applies if you have a PostScript printer and if you choose PostScript in the Printer Preferences editor.

ScreenMode

Selects a display mode for the Workbench screen.

Format SCREENMODE [FROM <filename>]
[EDIT | USE | SAVE]

Template FROM,EDIT/S,USE/S,SAVE/S

Location Extras:Prefs

Example:

```
1> SCREENMODE Prefs/Presets/DTPscreen USE
```

You are prompted to close all non-drawer windows; the system resets and uses the settings saved in the DTPscreen file. The editor window does not open. When the system is rebooted, the display mode returns to the last selection saved.

Serial

Sets the specifications for communication through the serial port.

Format SERIAL [FROM <filename>][EDIT | USE | SAVE]
[PUBSCREEN <public screen name>] [UNIT]

Template FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K,
UNIT/S

Location Extras:Prefs

Example:

```
1> SERIAL Prefs/Presets/Serial.9600 PUBSCREEN  
MidTerm UNIT
```

Opens the Serial editor, loading the Serial.9600 preset. The editor opens on the MidTerm public screen and its window contains a Unit gadget.

Sound

Controls the type of sound and sound attributes produced by the Amiga.

Format SOUND [FROM <filename>] [EDIT | USE | SAVE]
 [PUBSCREEN <public screen name>]

Template FROM,EDIT/S USE/S SAVE/S PUBSCREEN/K

Location Extras:Prefs

Time

Sets the system clock.

Format TIME [EDIT | SAVE] [PUBSCREEN <public screen
 name>]

Template EDIT/S,SAVE/S,PUBSCREEN/K

Location Extras:Prefs

Since setting the time from the Shell always involves saving, the USE option is omitted for the Time editor.

WBPattern

Creates background patterns for the Workbench and Workbench windows.

Format WBPATTERN [FROM <filename>]
 [EDIT | USE | SAVE] [CLIPUNIT <clipboard unit>]
 [NOREMAP]

Template FROM,EDIT/S,USE/S,SAVE/S,CLIPUNIT/K/N,
 NOREMAP/S

Location Extras:Prefs

Example:

```
1> WBPATTERN Prefs/Presets/Wallpaper.pattern  
NOREMAP
```

Opens the WBPATTERN editor, loading the Wallpaper.pattern preset. The NOREMAP option prevents the remapping of colors in loaded pictures and patterns.

Commodities Programs

The following commands invoke the Workbench Commodity Exchange utilities. They are located in the Tools/Commodities directory.

The following arguments in Commodities program format statements have the same meaning for each command in which they appear. These are the same as entering the corresponding Tool Type in the icon's Information window.

Argument	Meaning
[CX_PRIORITY <priority>]	Sets the priority of the commodity in relation to all the other Commodity Exchange programs. The default value is 0; values higher than 0 give priority over other commodities in regard to hot keys and other commodity related issues. This does not affect task priorities.
[CX_POPKEY <key>]	Allows you to specify the hot key that opens the program's window, if any. When specifying more than one key, enclose the keys in double quotation marks. (For example, CX_POPKEY="Shift F1").
[CX_POPUP= <yes/no>]	Selects whether the program window opens when the command is given.

AutoPoint

Automatically selects any window the pointer is over.

Format AUTOPOINT [CX_PRIORITY <priority>]

Template CX_PRIORITY/N/K

Location Extras:Tools/Commodities

Press Ctrl+C or use the BREAK command to exit AutoPoint when it is started from a Shell.

Blanker

Causes the monitor screen to go blank or display an animation if no input has been received within a specified period of time. This helps preserve your monitor.

Format BLANKER [CX_PRIORITY <priority>
 [*CX_POPKEY* <key>] [*CX_POPUP*=<yes | no>]
 [*SECONDS* <timeout>] [*CYCLECOLORS* <yes | no>]
 [*ANIMATION* <yes | no>]

Template CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K,S
 ECONDS/N/K,CYCLECOLORS/K,ANIMATION/K

Location Extras:Tools/Commodities

The arguments for Blanker are the same as the Tool Types in Blanker's Information window.

Press Ctrl+C or use the BREAK command to exit Blanker when it is started from a the Shell.

Example 1:

```
1> BLANKER SECONDS 45
```

The Blanker window opens and 45 is displayed inside its text gadget. If no mouse or keyboard input is received during a 45 second interval, the screen goes blank.

Example 2:

```
1> BLANKER CX_POPUP=no
```

The Blanker program starts. If no input is received within 60 seconds (the default), the screen goes blank. The Blanker window does not open.

A further example using Blanker appears in Chapter 8.

ClickToFront

Allows you to bring a window to the front of the screen by double-clicking on it.

Format CLICKTOFRONT [CX_PRIORITY <priority>]
 [QUALIFIER <qualifier>]

Template CX_PRIORITY/N/K,QUALIFIER/K

Location Extras:Tools/Commodities

ClickToFront does not open a window. The arguments are the same as the Tool Types in ClickToFront's Information window.

Press Ctrl+C or use the BREAK command to exit ClickToFront when it is started from a Shell.

CrossDOS

Sets text filter and conversion options for CrossDOS devices.

Format CROSSDOS [CX_PRIORITY <priority>]
 [CX_POPKEY <key>] [CX_POPUP <yes | no>]

Template CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K,

Location Extras:Tools/Commodities

CrossDOS lets you read from and write to MS-DOS formatted disks using your standard Amiga drives.

Press Ctrl+C or use the BREAK command to exit CrossDOS when it is started from a Shell.

Exchange

Monitors and controls the Commodity Exchange programs.

Format EXCHANGE [CX_PRIORITY <priority>]
 [CX_POPKEY<key>] [CX_POPUP <yes|no>]

Template CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K

Location Extras:Tools/Commodities

Press Ctrl+C or use the BREAK command to exit Exchange when it is started from a Shell.

Example:

```
1> EXCHANGE CX_POPKEY "Shift F1"
```

The Exchange program is started and its window appears on the screen. When its window is hidden, pressing Shift+F1 reveals it.

FKey

Assigns commands to special key sequences, eliminating the need for repetitive typing.

Format FKEY [CX_PRIORITY <priority>]
 [CX_POPKEY <key>] [CX_POPUP <yes|no>]

Template CX_PRIORITY/N/K,CX_POPKEY/K,CX_POPUP/K

Location Extras:Tools/Commodities

FKey assigns any of eight commands to any key sequence that can be entered.

Press Ctrl+C or use the BREAK command to exit FKey when it is started from a Shell.

MouseBlanker

Removes the mouse pointer from the screen while entering input from the keyboard.

Format MOUSEBLANKER [CX_PRIORITY <priority>]

Template CX_PRIORITY/N/K

Location Extras:Tools/Commodities

Press Ctrl+C or use the BREAK command to exit MouseBlanker when it is started from a Shell.

NoCapsLock

Disables the Caps Lock key.

Format NOCAPSLOCK [CX_PRIORITY<priority>]

Template CX_PRIORITY/N/K

Location Extras:Tools/Commodities

Press Ctrl+C or use the BREAK command to exit NoCapsLock when it is started from a Shell.

Other Workbench-Related Tools and Programs

The following group of commands invoke other Workbench tools, utilities, and programs.

Calculator

Provides an on-screen calculator.

Format `CALCULATOR [PUBSCREEN <public screen name>] [TAPE <window>]`

Template `PUBSCREEN,TAPE/K`

Location `Extras:Tools`

The output of the Calculator can be copied and pasted into any console window, such as the Shell or ED.

TAPE creates a Calculator window of a specific size in which your input and output is displayed. The specification is in the form of:

```
TAPE=RAW:x/y/width/height/title/options
```

For a description of the options and arguments used for the TAPE window, see the description of window specification for the NEWSHELL command in Chapter 6.

Clock

Provides an on-screen clock.

Format CLOCK [DIGITAL] [<LEFT>] [<TOP>] [<WIDTH>]
 [<HEIGHT>] [24HOUR] [SECONDS] [DATE]
 [<FORMAT><n>] [*PUBSCREEN* <public screen
 name>]

Template DIGITAL/S,LEFT/N, TOP/N,WIDTH/N,
 HEIGHT/N,24HOUR/S,SECONDS/S,DATE/S,
 FORMAT/N,PUBSCREEN/K

Location SYS:Utilities

The DIGITAL option opens a digital clock. A resizable analog clock is the default.

The LEFT, TOP, WIDTH, and HEIGHT options allow you to specify the size and position of the clock. The keywords are optional; if not given, the numerical arguments are interpreted by their position as follows:

- 1st number** The clock opens <n> pixels from the left edge of the screen.
- 2nd number** The clock opens <n> pixels from the top of the screen.
- 3rd number** The analog clock is <n> pixels wide.
- 4th number** The analog clock is <n> pixels high.

For example, to specify only the width and height of the Clock, use the WIDTH and HEIGHT keywords. When entering only two numbers, the clock interprets them as the LEFT and TOP positions. WIDTH and HEIGHT are not available if you use the DIGITAL option.

The 24HOUR option displays the time in 24 hour mode, which is not available for the analog clock.

The SECONDS option displays a second hand on the analog clock and has no effect if DIGITAL is specified.

The DATE option displays the date.

The FORMAT option applies only to the digital clock. It takes a value from 0 to 5, which determines which of the six digital formats is used.

Formats 4 and 5 vary, depending on your Locale Preferences Editor settings. To specify a digital format, either include the **FORMAT** keyword or use **LEFT**, **TOP**, **WIDTH**, and **HEIGHT** values; the **WIDTH** and **HEIGHT** values function as placeholders only and are ignored.

Example 1:

To open a clock that is 75 pixels from the left edge of the screen, 75 pixels from the top edge of the screen, 300 pixels wide and 100 pixels high, enter:

```
1> CLOCK 75 75 300 100
```

Example 2:

To use the **SECONDS** and **DATE** options, enter:

```
1> CLOCK SECONDS DATE
```

Example 3:

To open a 24-hour digital clock with seconds that is 320 pixels from the left edge of the screen and in the screen's title bar (0 pixels from the top), enter:

```
1> CLOCK DIGITAL 320 0 FORMAT 2
```

For more examples using Clock, see Chapter 8.

CMD

Redirects serial or parallel output to a file.

Format **CMD** <devicename> <filename> [*OPT S|M|N*]

Template DEVICENAME/A,FILENAME/A,OPT/K

Location Extras:Tools

The <devicename> can be serial or parallel. To redirect printer output, it should be the same device as specified in the Printer editor. <Filename> is the name of the file to which the redirected output should be sent.

The CMD options are as follows:

- S** Skip any short initial write (usually a reset if redirecting a screen dump).
- M** Redirect multiple files until a BREAK command or Ctrl+C is entered.
- N** Notify user of progress (messages are displayed on the screen).

Example:

```
1> CMD parallel RAM:cmd_file
```

Any output sent to the parallel port is rerouted to a file in RAM: called cmd_file.

DiskCopy

Copies the contents of one disk to another.

Format DISKCOPY [FROM] <device> [TO] <device>
[NAME <name>] [NOVERIFY] [MULTI]

Template FROM/A,TO/A,NAME/K,NOVERIFY/S,MULTI/S,

Location SYS:System

The <device> parameters specify the name of the disk devices to copy from and to copy to; for example, DF0: and DF1:.

By default, the destination disk has the same name as the source disk. If you specify the NAME option, you can give the destination disk a different name from the source disk.

Normally during a DiskCopy, the Amiga copies and verifies each cylinder of data. The NOVERIFY option allows you to skip the verification process, making the copy faster.

The MULTI option loads the data on the source disk into memory, allowing you to make multiple copies without having to read the data from the source disk each time.

Example 1:

```
1> DISKCOPY DF0: TO DF2:
```

copies the contents of the disk in drive DF0: to the disk in drive DF2: overwriting the contents of the disk in drive DF2:.

Example 2:

```
1> DISKCOPY DF0: TO DF2: NAME NewDisk NOVERIFY
```

copies the contents of the disk in drive DF0: to the disk in drive DF2: and gives the disk in drive DF2: the name NewDisk. The disk is not verified as it is copied.

FixFonts

Updates the .font files of the FONTS: directory.

Format	FIXFONTS
Template	(none)
Location	SYS:System

FixFonts does not open a window or produce any output. While the FONTS: directory is updated, the drive activity light is on. When the update is finished, the light goes off and the Shell prompt reappears. If the disk needed to update the FONTS: directory is not available or if there is a problem with it, FixFonts produces a standard requester concerning the problem.

Use FixFonts whenever you make changes in the FONTS: directory; for example, copying new font files or deleting single font sizes.

Format

Formats a disk for use with the Amiga.

Format *FORMAT DEVICE | DRIVE* <device> *NAME* <name>
 [OFS | FFS]
 [INTERNATIONAL | NOINTERNATIONAL]
 [DIRCACHE | NODIRCACHE] [NOICONS] [QUICK]

Template *DEVICE=DRIVE/K/A,NAME/K/A,OFS/S,FFS/S,*
 INTL=INTERNATIONAL/S,
 NOINTL=NOINTERNATIONAL/S,DIRCACHE/S,
 NODIRCACHE/S,NOICON/S,QUICK/S

Location SYS:System

You must specify both the *DEVICE* and the *NAME* keywords to format a disk. The name can be up to thirty-one characters in length. If there are spaces in the name, enclose it in double quotation marks.

The *OFS* option formats the disk using the Old File System. The *FFS* option formats the disk using the Fast File System. *FFS* formatted disks are faster than *OFS* formatted disks; however, the *FFS* disks are not compatible with Amiga system software releases prior to 2.04. The default setting for floppy disks is *OFS* and for PCMCIA cards and hard drive partitions is *FFS*.

The *INTERNATIONAL* option formats disks using the international versions of the file systems. International file systems handle upper and lower letter case conversions of international characters in file names. The *NOINTERNATIONAL* option forces the non-international file system on devices for which International mode is the default. The default setting for floppy disks and PCMCIA cards is *NOINTERNATIONAL* and for hard drive partitions is *INTERNATIONAL*. Disks created with *INTERNATIONAL* mode set are not compatible with Amiga system software releases prior to 2.04.

The *DIRCACHE* option enables directory caching, which speeds the opening of drawers, files, requesters, and listings. Using directory caching for floppy disks and systems with slow hard drives speeds directory listings and Workbench window opening. Directory caching is not useful on systems with fast hard drives. The *NODIRCACHE* option disables directory caching. Disks formatted with Directory

Caching set are not compatible with Amiga system software releases prior to 3.0. The default setting for floppy disks, PCMCIA cards, and hard drives is NODIRCACHE.

The NOICONS option prevents a Trashcan icon and directory from being added to the newly formatted disk.

The QUICK option specifies that Format only formats and creates the root block (and track), the boot block (and track) and creates the bitmap blocks. This can only be used with previously formatted disks. However, you cannot use this to reformat disks for AmigaDOS that were previously formatted for CrossDOS and vice versa.

Example 1:

```
1> FORMAT DRIVE DF0: NAME EmptyDisk
```

Formats the disk in drive DF0:, erasing any data, and names the disk EmptyDisk.

Example 2:

```
1> FORMAT DRIVE DF2: NAME NewDisk QUICK
```

Reformats, or erases, a disk that already contains data.

For more examples using the Format command, see Chapter 8.

GraphicDump

Prints the frontmost screen.

Format	GRAPHICDUMP [TINY SMALL MEDIUM LARGE <xdots>:<ydots>]
Template	TINY/S,SMALL/S,MEDIUM/S,LARGE/S, <xdots>:<ydots>/S
Location	Extras:Tools

GraphicDump waits ten seconds before starting to print to allow you to bring the desired screen to the front of the display.

The size options, which correspond to the program's acceptable Tool Types, determine the width of the printout:

TINY	1/4 the total width allowed by the printer
SMALL	1/2 the total width allowed by the printer
MEDIUM	3/4 the total width allowed by the printer
LARGE	The full width allowed by the printer

The printout's height maintains the proportions of the screen.

To specify exact dimensions, substitute the absolute width in printer dots for <xdots> and the absolute height for <ydots>, separated by a colon (:).

Example 1:

```
1> GRAPHICDUMP SMALL
```

produces a printout of the frontmost screen that is about one-half the total width allowed by the printer.

Example 2:

```
1> GRAPHICDUMP 600:300
```

produces a printout that is 600 dots wide by 300 dots high.

IconEdit

Edits the appearance and type of icons. IconEdit opens the IconEdit program. The command does not support any arguments. For detailed information on IconEdit, see the *Workbench User's Guide*.

InitPrinter

Initializes a printer for print options specified in the Preferences editors.

Format	INITPRINTER
Template	(none)
Location	Extras:Tools

After running InitPrinter, the Shell prompt returns when the printer resets. The printer is initialized automatically on first access, but if it is powered off, you might need to reinitialize it with InitPrinter.

Intellifont

Runs the Intellifont program to manage outline fonts.

Format	INTELLIFONT [VALIDATE]
Template	VALIDATE/S
Location	SYS:System

Intellifont installs new outline fonts on your system, specifies new sizes for existing fonts, and deletes unneeded fonts. You can also create bitmap versions of any size outline font for applications that do not support outline fonts directly.

Setting the VALIDATE option checks fonts to be sure they are installed properly and that everything needed to run them is present.

KeyShow

Displays the current Keymap. There are no arguments for the KeyShow command. For detailed information about KeyShow, see the *Workbench User's Guide*.

MEmacs

Enables screen-oriented text editing.

Format MEMACS [<filename>] [OPT W] [GOTO <n>]

Template FROM/M,OPT/K,GOTO/K

Location Extras:Tools

MEmacs is described in Chapter 4 of this manual.

More

Displays the contents of a designated ASCII file in the Shell window. More does not have an icon. It has been superseded by the MultiView program; however, it is still available.

Format MORE <filename>

Template (none)

Location SYS:Utilities

Specify the complete path if the file is not in the current directory. More displays a file requester if you do not specify a file.

Pressing the H key provides explanations for More's command keys. The following options can be used with More:

<Space>	Next Page (More).
<Return>	Next Line.
q or Ctrl+C	Quit.
h	Help.
/string	Search for string (case-sensitive). Entering n following the string searches for the next occurrence of the string.

.string	Search for string (not case-sensitive).
n	Find next occurrence of search string.
Ctrl+L	Refresh window.
<	First page.
>	Last page.
%N	Move N% into the file.
b or <Backspace>	Previous page (less).
E	Edit the current file using the editor set in ENV:EDITOR.

More also accepts input from a PIPE. The Previous Page (Backspace or b), Last Page (>), and Move N% into file (%N) commands are disabled when the More input is from a PIPE because standard input from a PIPE is of unknown length.

When you use More from the Shell, you can open an editor to use on the file you are viewing (press Shift+E) if the EDITOR variable is defined. Provide the complete path to the specified editor in the EDITOR variable; for example, C:ED.

Example:

```
1> MORE DF0:TestFile
```

displays the contents of the ASCII file called TestFile on the disk in drive DF0:.

MultiView

Displays picture files, text files, AmigaGuide files, sound files, and animated graphics files.

Format MULTIVIEW [FILE <filename>] [CLIPBOARD]
 [CLIPUNIT <clipboard unit>] [SCREEN]
 [PUBSCREEN <public screen name>]
 [REQUESTER] [BOOKMARK] [FONTNAME <font
 name>] [FONTSIZE] [BACKDROP]
 [WINDOW] [PORTNAME <ARexx port name>]

Template FILE,CLIPBOARD/S,CLIPUNIT/K/N,SCREEN/S,
 PUBSCREEN/K,REQUESTER/S,BOOKMARK/S,
 FONTNAME/K,FONTSIZE/K/N,BACKDROP/S,
 WINDOW/S,PORTNAME/K

Location SYS:Utilities

Be sure to specify the complete path to the file if it is not in the current directory. MultiView displays a file requester if no file is specified.

If CLIPBOARD is specified, the Clipboard is viewed instead of a file. CLIPUNIT specifies the Clipboard unit to use when using the CLIPBOARD keyword.

SCREEN indicates that you want the object to appear on its own screen rather than in a window on the Workbench screen, using the display mode specified by the object. For example, if an ILBM picture file is Low Res, MultiView opens a Low Res screen.

If REQUESTER is specified, MultiView displays a file requester.

BOOKMARK recalls the object and position when opening a file with a bookmark.

FONTNAME specifies the font to use when viewing text objects. FONTSIZE specifies the font size in points to use when viewing text files.

BACKDROP indicates that the window should be a backdrop window.

WINDOW allows the MultiView window to open without requesting a file to load. Using this option lets you keep a small MultiView

window open on the Workbench screen so you can drag icons into it whenever necessary.

PORTNAME allows you to specify an ARexx port name when you run MultiView. If you do not specify a name, each MultiView window is given the default port name of MultiView.x, where x is a slot number starting from 1. (For example, if you have three MultiView display windows open, their port names are MultiView.1, MultiView.2, and MultiView.3.) This port name allows you to refer to a particular MultiView display from within an ARexx script.

MultiView supports the following ARexx commands:

OPEN	Open an object from the specified file name or Clipboard unit. The options are as follows: FILENAME/K Specifies the object file name. CLIPBOARD/S Specifies that the object comes from the Clipboard. CLIPUNIT/K/N Specifies the Clipboard unit from which to obtain the object if using the CLIPBOARD option.
RELOAD	Reload the current object.
SAVEAS	Save the object to the specified file. If there is a selected block, then only that block is saved. SAVEAS has one option: NAME/K.
PRINT	Print the current contents. If there is a selected block, then only that block is printed.
ABOUT	Display the About requester.
QUIT	Close MultiView.
COPY	Copy the current contents to the Clipboard. If there is a selected block, then only that block is copied.
CLEARSELECTED	Clear the selection block.

GETTRIGGERINFO	Returns the commands for the trigger methods of the current object. The options are VAR/S and STEM/K. If using the STEM option, the following stem extensions are used: .COUNT Number of elements .n.LABEL Label .n.COMMAND Command .n.METHOD Numeric method
DOTRIGGERMETHOD	Perform the trigger method on the current object. The option is METHOD/A.
SCREEN	Specify whether to display object on a screen. The options are TRUE/S and FALSE/S.
PUBSCREEN	Specify the name of the public screen on which to display the object. The option is NAME/A.
GETCURRENTDIR	Returns the full name of the current directory associated with the current object.
GETFILEINFO	Returns the complete path and file name associated with the current object.
GETOBJECTINFO	Returns the name, basename, group, and ID for the current object. The options are VAR/S and STEM/K. If using the STEM option, the following stem extensions are used: .FILENAME File name of object .NAME Descriptive name of the DataType .BASENAME Base name of the DataType .GROUP Group containing the DataType .ID ID string for the DataType
MINIMUMSIZE	Size the window to its minimum size for the contents type.
NORMALSIZE	Size the window to its normal size for the contents type.
MAXIMUMSIZE	Size the window to its maximum size for the contents type.
WINDOWTOFRONT	Move the window to the front of the display.

WINDOWTOBACK	Move the window to the back of the display.
SCREENTOFRONT	Bring the screen to the front of the display.
SCREENTOBACK	Send the screen to the back of the display.
ACTIVATEWINDOW	Activate the window.
BEEPSCREEN	Cause a display beep in the screen that the window resides in.

NoFastMem

Forces the Amiga to use only Chip RAM. NoFastMem disables any Fast (or expansion) RAM used by the system. The expansion memory can be turned on again by sending the NoFastMem program a break, either via the BREAK command or by pressing Ctrl+C. NoFastMem has no options. For detailed information on NoFastMem, see the *Workbench User's Guide*.

PrepCard

Prepares a PCMCIA memory card for use as a disk device or system RAM on systems with card slots.

Format	PREPCARD [DISK RAM]
Template	DISK/S, RAM/S
Location	Extras:Tools

The DISK option prepares the PCMCIA card for use as disk device CC0:. Reading and writing to the DISK-prepared card is the same as reading and writing to a floppy disk. We recommend that you use a battery-backed PCMCIA card as disk if you wish to preserve your work.

The RAM option prepares the PCMCIA card for use as system RAM. Booting or rebooting with the card inserted adds the card's memory to the existing memory in the Amiga. You can use any PCMCIA cards as RAM, except Read Only (ROM) cards.

The DISK and RAM options are mutually exclusive; if you specify both options, PrepCard uses the first option specified and disregards

the other. An error message results if you try to use PrepCard on a system that does not have a card slot.

Chapter 8

Command Examples

The command examples elsewhere in this book are primarily to illustrate the proper syntax and general operation of AmigaDOS. This chapter shows you how to use the commands needed for a wide variety of common tasks.

The chapter is organized as follows:

- Basic tasks
- Occasional tasks
- Advanced tasks

Basic Tasks

This section is oriented toward the novice Shell user, showing commands and short scripts to accomplish basic tasks. Use the commands shown as models for your own commands, substituting the names of your disks, directories, and files. To use the commands, type what appears after the prompt (usually `1>`). Press Return to enter the command line you type.

Opening a Shell Window

To open a Shell window from Workbench:

1. Open the System drawer on your Workbench disk or partition.
2. Double-click on the Shell icon.

OR

1. Choose the Execute Command... item from the Workbench menu.

2. In the requester that appears, enter the command **NEWSHELL**.

To open another Shell window from a Shell, enter the NEWSHELL command at a Shell prompt:

```
1> NEWSHELL
```

Running Programs from the Shell

To run a program that is on the search path, enter the program name at the prompt:

```
1> Clock
```

To run a program that is not on the search path, enter the full path to the program:

```
1> Tempus:Fugit/Utils/SuperClock
```

To run a program that is not on the search path but is in a subdirectory of the current directory, enter the relative path to the program:

```
1> Utils/SuperClock
```

Stopping a Program

AmigaDOS commands and most Workbench programs started from the Shell can be exited, or stopped if currently running, by pressing Ctrl+C. This is important in case you need to abort a pattern matching DELETE, or to interrupt a directory listing or other lengthy process. Scripts can be stopped with Ctrl+D.

To stop a command or program that is currently running:

1. Make the Shell window from which the command or program was started the current window by clicking in it.
2. Press Ctrl+C.

In some cases you may need to press Return after Ctrl+C to bring back the Shell prompt.

To stop a script that is currently running:

1. Make the Shell window from which the script was started the current window by clicking in it.
2. Press Ctrl+D.

Changing the Current Directory

The current directory is normally part of the standard Shell prompt, as in **1.Workbench:>**. In the following examples, notice the prompt to see how the current directory changes.

To save typing, change the current directory to the one in which you are working.

If you are issuing two or more commands that refer to things in a certain directory, make it the current directory using the CD command. The following two sets of commands both accomplish the same task:

```
1.Work:> COPY Storage/Keymaps/usa2 TO DEVS:Keymaps
1.Work:> DELETE Storage/Keymaps/usa2

1.Work:> CD Storage/Keymaps
1.Storage:Keymaps> COPY usa2 TO DEVS:Keymaps
1.Storage:Keymaps> DELETE usa2
```

Entering the second set of commands instead of the first saves over a dozen keystrokes. This savings is even greater if further work in Storage/Keymaps is needed.

To change the current directory with as little typing as possible, omit the CD command, and use the slash and colon to move through the directory structure:

```
1.Workbench:Devs/Monitors> /Printers
1.Workbench:Devs/Printers> :Prefs/Presets
1.Workbench:Prefs/Presets> /
1.Workbench:Prefs>
```

To switch quickly between two current directories, use the PCD script (located in the S: directory):

```
1.Workbench:> PCD Devs/DOSDrivers
1.Workbench:Devs/DOSDrivers> Extras:Storage
1.Extras:Storage> PCD
1.Workbench:>
```

To see the current directory, if the Shell prompt does not show it, use the CD command alone:

```
1> CD
Workbench:
```

Changing the Search Path

To create a directory on the SYS: volume for additional commands and add it to the search path for the current Shell:

```
1> MAKEDIR SYS:MyCommands
1> PATH SYS:MyCommands ADD
```

To add MyCommands to the search path, effective for the whole system, use an ASSIGN command instead of PATH:

```
1> ASSIGN C: SYS:MyCommands ADD
```

To have the Amiga look for commands in a C directory on any disk inserted in drive DF0:, use ASSIGN with the PATH option:

```
1> ASSIGN C: DF0:C PATH
```

Displaying the Contents of a Directory

To display the names of files and subdirectories in a directory use DIR:

```
1> DIR DEVS:
  DataTypes (dir)
  Monitors (dir)
  DOSDrivers (dir)
  Keymaps (dir)
  Printers (dir)
clipboard.device      DataTypes.info
DOSDrivers.info       Keymaps.info
mfm.device            Monitors.info
parallel.device       postscript_init_ps
printer.device        Printers.info
serial.device         system-configuration
```

To display the names of files, subdirectories, and files in the subdirectories in a directory, add the ALL keyword (a partial listing of the output is shown here):

```
1> DIR DEVS: ALL
  DataTypes (dir)
    8SVX                8SVX.info
    AmigaGuide          AmigaGuide.info
    ANIM                ANIM.info
    CDXL                CDXL.info
    FTXT                FTXT.info
    ILBM                ILBM.info
  Monitors (dir)
    A2024               A2024.info
```

To display the names of files only, with no directories, add the FILES keyword:

```
1> DIR DEVS: FILES
  clipboard.device      DataTypes.info
  DOSDrivers.info      Keymaps.info
  mfm.device           Monitors.info
  parallel.device      postscript_init_ps
  printer.device       Printers.info
  serial.device        system-configuration
```

To display the names of files only, without .info files, use pattern matching:

```
1> DIR DEVS:~(#?.info) FILES
  clipboard.device      mfm.device
  parallel.device      postscript_init_ps
  printer.device       serial.device
  system-configuration
```

To display information about files that includes their size and protection bits, without date and time, use LIST with the FILES and NODATES keywords:

```
1> LIST DEVS:~(#?.info) FILES NODATES
  clipboard.device      6944 ----rw-d
  mfm.device           6684 ----rw-d
  parallel.device      4272 ----rw-d
  postscript_init_ps   5014 ----rw-d
  printer.device       27420 ----rw-d
  serial.device        5412 ----rw-d
  system-configuration 232 ----rw-d
```

To display information about a single file, use LIST with the path to the file:

```
1> LIST S:Startup-sequence
Directory "S:" on Tuesday 01-Dec-92
Startup-sequence 1360 -s--rw-d 30-Oct-92 12:00:21
1 file - 4 blocks used
```

To display the amount of space used by a directory and its contents, including all files in subdirectories, use the ALL keyword:

```
1> LIST ALL
```

After the contents of the current directory are listed, a summary line such as the following is displayed:

```
TOTAL: 113 files - 762 blocks used
```

Divide the number of blocks by two to get the number of kilobytes (KB).

To see information from LIST, DIR, or other commands that have scrolled off the Shell window:

Select the Shell window's zoom gadget once to switch to its alternate size, which normally fills the screen. As much of the previous output as fits fills the window. Select zoom again to restore the window to its previous size.

If the window's maximum height is not large enough to reveal the desired output, reissue the command by pressing the up arrow and then Return. Pause and resume the scrolling of the output when necessary by pressing the spacebar and backspace, respectively.

To combine the CD and DIR commands:

Create the following script and save it as S:CDD. (For an example of how to create a script, see "*Creating a User-startup File*" on page 8-8.)

```
.KEY dirpath
CD <dirpath>
DIR
```

Set the script's s protection bit by entering **PROTECT S:CDD +s**. Then whenever you enter CDD followed by the path to a directory, this script makes that directory the current directory and lists its contents.

Copying Files and Directories

When copying a single file from one place to another, you need to include only the paths for each. FROM and TO keywords are optional. For clarity, most COPY examples in this book use the TO keyword, but omit the FROM keyword.

To copy a file to an existing directory using optional keywords:

```
1> COPY FROM DF0:Pix/Fractal3 TO Work:Pictures
```

To copy the file omitting optional keywords:

```
1> COPY DF0:Pix/Fractal3 Work:Pictures
```

To copy a file and rename it at the same time, include the new file name in the TO argument:

```
1> COPY DF0:Pix/Fractal3 TO Work:Pictures/BestPic
```

To copy all the files in a directory to another directory, without copying the directory itself or the subdirectories it contains:

```
1> COPY DF0:Pix TO Work:Pictures
```

The contents of DF0:Pix are deposited in Work:Pictures, not grouped in their own directory. If Pix contained directories, they are not copied, but any .info files for drawers Pix contained are copied, making it appear at first that the drawers were copied.

To copy all the files in a directory to another directory, copying the directory itself but not the subdirectories it contains:

```
1> COPY DF0:Pix TO Work:Pictures/Pix
```

The directory Pix is created in Work:Pictures if it does not already exist. The destination directory does not have to be the same name as the source; you can copy to Work:Pictures/Fractals, for example.

To copy a complete directory and its contents to another directory, use the ALL keyword:

```
1> COPY DF0:Pix TO Work:Pictures/Pix ALL
```

Work:Pictures/Pix is a duplicate of DF0:Pix.

To copy only certain files to another directory, use pattern matching if their names are similar:

```
1> COPY DF0:Pix/Fractal[3-7] TO Work:Pictures/Pix
```

Files in DF0:Pix with names beginning in Fractal and ending in the digits 3 through 7 are copied.

To copy specific files to another directory, include all the file names. Change to the source directory first to avoid having to enter the full path for each:

```
1> CD DF0:Pix
1> COPY Fractal3 Julia Dragon TO Work:Pictures/Pix
```

When copying more than one file at once without using the TO keyword, COPY expects the last name to be the destination directory. With whole-directory, multiple-file, and pattern matching operations, COPY outputs the names of the files copied and directories created as it executes.

Creating a User-startup File

A User-startup file is the Shell equivalent of the Workbench WBStartup drawer. It is a text file that is executed as a script by the default Startup-sequence. Place here any configuration commands, such as ASSIGNS, and the names of programs you wish to run automatically whenever you boot.

To create a User-startup file:

```
1> RUN ED S:User-startup
```

After the ED window opens, enter the commands you want on subsequent lines. For example, you can add some cache buffers to speed floppy access, add a directory of custom commands to the path, and start the screen blanker, by entering these lines:

```
ADDBUFFERS >NIL: DF0: 25
PATH >NIL: SYS:MyCommands ADD
RUN Blanker CX_POPUP=NO SECONDS=600 ANIMATION=YES
```

Then save the file and exit by pressing Esc,X,Return. The next time you boot or reboot, these commands are executed. When you have more commands to add, edit the file by entering **RUN ED S:User-startup** again.

Creating an Assignment

On a floppy-only system, a requester similar to that illustrated in Figure 8-1 usually means that you need to insert the named floppy disk.

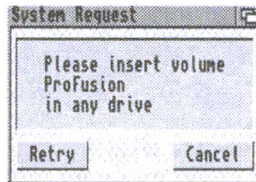


Figure 8-1. Sample System Requester

The most likely reason for this requester on a hard disk system, aside from entering a command with a misplaced colon (:), is that an application you are running requires an assignment to be made with the ASSIGN command. This is often done for you by an installation program, but not all applications have an installation program or one that works correctly for all systems.

The assignment tells the application to look on your hard drive instead of a floppy drive for things it needs. If the application is one you use regularly, you should place the ASSIGN statement in your User-startup. When the requester first comes up, however, you can enter the assignment in the Shell and then select the requester's Retry gadget.

To allow you to continue your work with a program installed from a disk called ProFusion to the Work: volume, enter the statement:

```
1> ASSIGN ProFusion: Work:ProFusion
```

where the first argument is the volume requested, followed by a colon, and the second argument is the device and directory where ProFusion is installed. Click the Retry gadget after entering the command. If the statement is correct, your application works properly and you should add the statement you entered to your User-startup. If it does not work, the second argument needs to be modified.

Accessing the Expanded ED Menus

The default S:Ed-startup file sets up a series of menus for the ED text editor. There is also an expanded set of menus that is built into ED, containing more options. You can make these available by renaming Ed-startup, which prevents it from being executed.

To allow access to the expanded ED menus, enter the following command:

```
1> RENAME S:Ed-startup TO S:Ed-startup.not
```

Working with a Single Shell

Although you can open several independent Shell windows at once, you may wish to avoid cluttering the Workbench screen with several windows. Starting a program from a Shell normally takes over that Shell window while the program is running, forcing you to open another Shell to enter additional commands. Use the following techniques to avoid this and allow any number of programs to run from a single Shell.

To run a program in the background so that the Shell window prompt returns, use the RUN command:

```
1> RUN DMC OPT def RHYME on  
[CLI 2]  
1>
```

The message in square brackets indicates the number of the new process started to run the program. The prompt returns immediately.

Even when a program is run this way, the Shell window cannot be closed until all programs started from it are exited. To avoid this, you can detach the program using redirection.

To detach a program, RUN it, adding output redirection to the NIL: device:

```
1> RUN >NIL: MultiView 8SVX/Sample  
[CLI 2]  
1>
```

You can now close the Shell window if necessary.

Note Redirection to NIL: also prevents any console output the program produces from appearing in the Shell window.

Attaching Icons

To attach an icon to a file or directory, you can create an icon with the IconEdit tool. However, it is often easier just to copy an existing icon with a Shell command.

You must copy a .info file of the right icon type, giving it a name that matches the file or directory to which you are attaching it plus the .info extension. If necessary, use the Information menu item to adjust the Default Tool of a copied Project icon, or the Tool Types of a copied Tool icon, as appropriate for the file.

Note The capitalization of the name under the icon matches that given in the COPY command, regardless of the capitalization of the associated file or directory.

To attach an icon to a file called PCX in the DataTypes directory, copy the .info file of an existing datatype in the directory:

```
1> COPY DataTypes/ILBM.info TO DataTypes/PCX.info
```

An icon titled PCX appears in the DataTypes window when you open it or choose Update from the Window menu.

Note If the icon you copy is snapshotted, the new icon retains the original icon's position and appears directly on top of it. Drag the new icon to a different position and Snapshot it to keep the two icons separate.

To attach a custom icon to a disk called VidTools, copy the desired disk-type .info file to the root directory of the disk, giving it the name disk.info:

```
1> COPY SYS:disk.info TO VidTools:disk.info
```

You must eject and reinsert the disk or reboot for a new disk icon to appear on the Workbench.

Creating Scripts Conveniently

To make creating and editing scripts easier:

Create a script containing the following lines, and save it as S:Edscr. (For an example of how to create a script, see "*Creating a User-startup File*" on page 8-8.)

```
.KEY script/A
ED S:<script>
FAILAT 11
IF EXISTS S:<script>
    PROTECT S:<script> srwd
ENDIF
```

Set the script's s protection bit by entering **PROTECT S:Edscr srwd**. Now using Edscr you can create and edit scripts without having to decide where to put them or remember to set their s bit. Just enter **Edscr** followed by the name of a script.

When you save and exit from ED, the script you worked on is saved in the S: directory under the name you gave. Its s protection bit is set automatically so that you can run the script from a Shell without needing the EXECUTE command.

Occasional Tasks

Tasks in this section are used less often, but almost every user needs to do these things at some time. These examples assume a certain amount of familiarity with AmigaDOS and the Shell.

Creating Aliases To Reduce Keystrokes

The aliases listed below can help speed your Shell work by reducing the number of keystrokes required for common commands.

To enable these as global aliases, edit S:Shell-startup as described previously for User-startup, adding these lines:

```
ALIAS c0 CD DF0:
ALIAS cs CD SYS:
ALIAS css CD S:
ALIAS d0 DIR DF0:
ALIAS dr DIR RAM:
ALIAS qdir DIR ~(#?.info)
ALIAS ls LIST
ALIAS cp COPY
ALIAS cc COPY [] CLONE
ALIAS del DELETE
ALIAS ren RENAME
ALIAS ns NEWSHELL
ALIAS es ENDSHELL
ALIAS pf printfiles
ALIAS fmt0 FORMAT DRIVE DF0: NAME [] FFS NOICONS
DIRCACHE
ALIAS edus RUN ED S:User-startup
ALIAS edsh RUN ED S:Shell-startup
ALIAS ednew RUN ED RAM:newfile
ALIAS chip ECHO "There are `avail chip` bytes of
Chip memory free."
```

Modify the alias names as desired. Use these as models for your own aliases.

Customizing NEWSHELL

You can control the Shell window with the WINDOW argument of the NEWSHELL command. It allows you to specify custom sizes, positions, and features for the Shell window. Following are two examples of different window specifications.

To open a convenient Shell window on your Workbench screen, enter the following command in a Shell or as a line in User-startup:

```
NEWSHELL CON://400/100/AShell/CLOSE/
ALT0/12/640/388
```

This creates a small Shell window titled AShell that leaves the left side of the Workbench window clear so that disk icons are not obscured. It has a close gadget and, on a High-Res Interlace screen, it expands to fill the entire screen except for the screen title bar when you select its zoom gadget.

To open a Shell window on a public screen, such as telecommunications program's terminal screen, use a window specification with the SCREEN option:

```
CON:0/20///???/CLOSE/SCREENTerm
```

This creates a Shell window called ??? that is near the top of the screen and is as wide and short as possible. The window opens on a public screen named Term, if that screen is available. If it is not, the Shell opens on the Workbench screen.

Modifying the Prompt

The Shell prompt is easily modified using the PROMPT command. You can add any fixed text to the prompt string and include, reorder, or leave out the substitution operators that display the process number, current directory, and return code. Placing escape sequences in the prompt string lets you make the prompt stand out visually from the command line and command output. You can also embed a command in the prompt with the back apostrophe feature.

Figure 8-2 illustrates two ways of modifying the prompt. The first uses escape sequences to produce a boldface, color 2 (white) prompt; see Appendix D for a listing of escape sequences. The second shifts the usual position of the substitution operators, embeds a DATE command using the back apostrophe, and changes the final character to a dollar sign (\$).

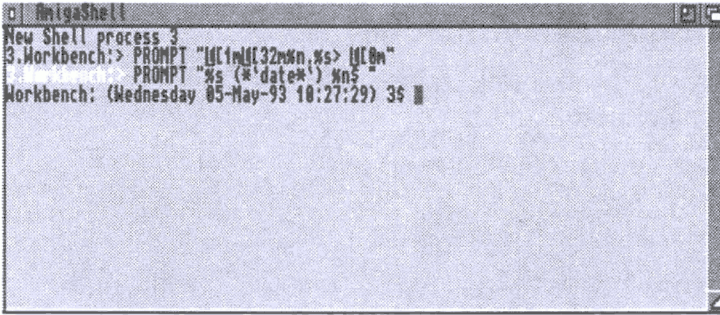


Figure 8-2. Sample Uses of PROMPT Command

Creating a Custom Ram Disk Icon

To have a custom icon for the Ram Disk, you need to place a COPY statement in User-startup. First, create the icon in IconEdit. Make sure it is a Disk type icon, then save it on your boot disk as DEVS:Ramdisk.info.

Note This example uses the DEVS: directory, but it does not matter where you store this icon. It is not visible, even in Show All Files mode, because Disk icons are visible only in the Workbench window.

To make your custom Ram Disk icon appear in the Workbench window, enter this line in the S:User-startup file, save the changed file, and reboot:

```
COPY DEVS:Ramdisk.info TO RAM:disk.info
```

If you also want to rename Ram Disk, include a RELABEL statement in User-startup after the COPY statement, such as:

```
RELABEL RAM:ramname
```

Deleting Files with Icons

To delete a file that has an icon and the file's .info file with a single command:

1. Create and save the following script as S:Delinf:

```
.KEY file/A  
DELETE <file> <file>.info QUIET
```
2. Enter **EXECUTE Delinf** with the name of the file as its argument.

Testing Commands

It is sometimes necessary to test the results of certain commands before using them on actual files. This is particularly true when using complex pattern matching with potentially destructive commands, such as **DELETE**, or escape sequences, with which it can be difficult to predict the outcome. There are various ways of testing commands that are quick and safe.

To test a potentially destructive pattern matching command:

1. Enter a non-destructive command such as **LIST** containing the pattern. For example,

```
LIST ~ (?.info|?.c|[0-9]#?)
```
2. Check the output to see whether the intended files were matched.
3. Modify the pattern if necessary and repeat the command until the command lists only the desired files.
4. Enter the intended command, using the same pattern.

To test the effects of escape sequences:

1. Enter an **ECHO** command containing the escape sequences and an example word. For example,

```
ECHO "*E[1mBOLD*E[0m"
```
2. Check to see whether the output is as intended.
3. Modify the escape sequences if necessary and repeat the command until the desired result appears.

4. Use the escape sequences in the intended command, such as PROMPT.

To clear the window and reset all escape sequence modes to the defaults, enter:

```
Esc, c, Return
```

You must use a lower case c. After the window is cleared and reset, the Shell displays a harmless `:Unknown command` message on the window's top line.

To create a test file in the Ram Disk, enter:

```
1> ECHO "This is only a test" TO RAM:foo
```

This provides a small, expendable file on which to test other commands. It is best to create such files in RAM:, to avoid cluttering your disks with small, useless files. Traditionally, files like this are named "foo" or "bar."

To test commands safely on actual files, copy the files to a directory in the Ram Disk and test with those files:

```
1> COPY Work:MyFiles/#? TO RAM:Testdir
```

Creating a Script to Move Files

AmigaDOS does not have a Move command. Normally, there are two ways of moving a file with AmigaDOS: a RENAME command or a combination of COPY and DELETE commands.

Using RENAME to move a file is possible only when moving the file to a destination on the same volume. Attempting to rename across devices causes an error. The copy-and-delete method works in any situation, but is cumbersome.

To create a Move command that lets you move a file within or across devices with a single command, create a script with the following commands:

```
1> RUN ED S:Move+  
PROTECT S:Move swrd
```

Enter these commands in the ED window:

```
.KEY source/A,to/A
.BRA {
.KET }
FAILAT 21
RENAME >NIL: {source} TO {to}
IF WARN
    COPY {source} TO {to}
    IF WARN
        ECHO "The file {source} could not be moved."
        QUIT 20
    ENDIF
    DELETE {source} QUIET
ENDIF
ECHO {source} "has been moved to" {to}
```

Save the script and exit ED. The s protection bit is automatically set after you exit.

Use this script the same as you would use a command. Enter MOVE followed by two arguments: the current path of the file to move and the path to the desired location.

Deleting with Interactive DIR

The DIR command has an interactive mode that pauses after each file or directory it lists and allows you to enter one of several simple commands. One option is to delete the item, which can be useful in certain situations.

For example, if you accidentally name a file #? (the wildcard combination that matches anything), attempting to delete it through normal methods is inadvisable. This is because **DELETE #?** deletes everything in the directory, including any other valuable files that are there.

The interactive DIR can solve this problem and be used as a general-purpose query/delete tool. This is helpful when you have a large directory containing many items to delete, but the file names do not have enough in common to make a pattern matching DELETE practical.

To delete various files on a disk called Debris safely with an interactive DIR, enter:

```
1> DIR Debris: INTER
```

DIR lists the names of the files in Debris alphabetically one by one, each followed by a question mark prompt. Press Return to go to the next file or E to enter a directory. When the name of an unwanted file appears, enter **DEL** to delete the file. Enter **Q** to leave the interactive DIR.

Generating Scripts with LIST LFORMAT

One of the prime uses of the LIST command's LFORMAT option is for automatically creating scripts used to process a series of files. You can produce a raw script using a LIST statement with LFORMAT, a TO argument to redirect the LIST output to a file, and pattern matching. You can then view the script and edit it manually if necessary before executing it.

To create a script that renames all the files in the current directory, adds the extension .IFF to their present file names, and places them in a directory called Paintfiles in the Work partition:

```
1> LIST #? TO T:renamer LFORMAT="RENAME %P%N TO  
Work:Paintfiles/%N.IFF"
```

Enter **ED T:renamer** to check the script, which should resemble the following. If the match pattern lists files that you did not want processed or your LFORMAT string does not work as expected, you can edit the script or modify and reenter the command.

```
RENAME Paint:Vince TO Work:Paintfiles/Vince.IFF  
RENAME Paint:Henri TO Work:Paintfiles/Henri.IFF  
RENAME Paint:Paul TO Work:Paintfiles/Paul.IFF  
RENAME Paint:Pablo TO Work:Paintfiles/Pablo.IFF  
RENAME Paint:Andy TO Work:Paintfiles/Andy.IFF
```

When the script is correct, leave ED and enter **EXECUTE T:renamer**.

See the "*Recursive AmigaDOS Command Script*" example on page 8-26 for a more advanced use of LIST LFORMAT.

Customizing LIST Output

You can also use LFORMAT to customize the output of LIST for special purposes. Save the line as an alias to make it easier to use in the future.

To create an alias for LIST that displays information only on files created since the date you enter, with the date first and protection bits and time omitted:

```
1> ALIAS lsince LIST FILES SINCE [ ] LFORMAT="%D
%-25N %L"
```

If the current directory is S:, the output of **lsince 01-sep-92** would be similar to this:

19-Oct-92	PCD	715
30-Nov-92	Startup-sequence	1360
Friday	Shell-startup	671
Yesterday	User-startup	609

Using ICONX to Run Scripts

If you prefer to work with the mouse whenever possible or you are setting up an Amiga for a Workbench-only user, ICONX is useful. Using C:ICONX as the Default Tool of a project icon lets you run a script (or a command that lacks a Workbench interface) from the icon. It lets you start the script by opening the icon, as if the script were a standard Workbench tool.

There are also advantages to starting programs this way for the advanced user, including scripting preparatory steps (such as loading special Preferences presets) before launching a program and allowing the program's task priority to be changed easily. The following example demonstrates these techniques.

To start an application called OldApp from an ICONX icon, first loading a special Preferences font preset, and adjusting the application's task priority, create this script for the icon:

```
CD SYS:
Prefs/Font FROM SYS:Prefs/Presets/defscrn.pre USE
CHANGETASKPRI -1
OldApp
```

Preventing Displayable Output From Scripts

In scripts, you often want to execute a command without the command displaying its usual Shell output. This can prevent an unnecessary series of messages or keep an output window from opening at an inopportune time.

To prevent all console output, redirect command output to a dummy destination with the >NIL: argument:

```
1> DELETE >NIL: T:Tempfile
```

The >NIL: argument prevents the printing of the message **T:Tempfile Deleted**, or any error messages.

To prevent console output except for error messages, use the QUIET option with those commands that support it:

```
1> DELETE T:Tempfile QUIET
```

The **T:Tempfile Deleted** message is not printed; however, if T:Tempfile does not exist, the **No file to delete** message appears.

Entering and Testing ARexx Macros

To make entering and testing ARexx macros simpler:

Enter the following AmigaDOS script and save it as S:Edrx:

```
.KEY mac
ED REXX:<mac>.rexx
PROTECT S:<mac>.rexx +S-E
RX REXX:<mac>
```

Use this script when experimenting with ARexx macros. Enter **Edrx** followed by the name of the macro. Edrx invokes the editor without you having to type in the .rexx extension and directly calls the ARexx interpreter RX with your macro as an argument when you exit.

Sorting and Joining Files

If you regularly capture a list of new files each time you log on to a telecommunications service, you might want to add them to a list of existing files and sort all the entries by date. Each file name has the

same prefix and a suffix that reflects the date, such as NewFiles.921009 for the list of files on October 9, 1992.

To create a single list of the files, sorted by date, use a pattern matching JOIN and SORT:

```
JOIN NewFiles.#? TO RAM:Temp.joined
SORT RAM:Temp.joined TO NewFiles.sorted
DELETE RAM:Temp.joined
```

Advanced Tasks

The following examples illustrate advanced tasks for users who are quite familiar with AmigaDOS.

Testing Software Versions

You might need to control what a script does depending on which software version a user of the script has. It is easy to test for specific version numbers from a script.

To test whether a script is running on an Amiga with Release 2 level system software, preface the version-dependent portion of the script with a sequence similar to this:

```
VERSION >NIL: 37
IF WARN
    ECHO "It's really time to update your system."
    QUIT
ELSE
    ECHO "You have Release 2 or better. Good!"
    ECHO "Let's continue."
ENDIF
```

Versions below 37 are pre-Release 2, version 38 is Release 2.1, version 39 is Release 3, and version 40 is Release 3.1.

Flushing Unused Fonts and Libraries

When fonts and libraries are loaded into memory, they remain in memory even if they are not currently in use. They are removed from memory automatically only when the memory they occupy is needed

for some other purpose. In some cases it is useful to remove unneeded resources yourself.

For example, if you want to edit and then test a font or find the version number of a library on a new disk, the Amiga may appear not to see the new font or library. This is because AmigaDOS uses the font or library that is in memory whenever possible. You can flush unused resources from memory and decrease memory fragmentation by using the AVAIL command's FLUSH option.

To flush an unused font or library from memory without having to reboot the Amiga, enter:

```
1> AVAIL FLUSH
```

For this command to eliminate a font or library, the library or font must not be in use by the Workbench or some other application.

AmigaDOS Loops Using EVAL

To create a loop in AmigaDOS that can prompt for the number of times to loop:

Enter the following script and save it as ALoop:

```
.KEY loop
; change bracket characters used for substitution
; since script uses < and > for redirection:
.BRA {
.KET }
; test whether user provided an argument
; for the number of loops, prompt if not:
IF NOT {loop}
    ECHO "Please type in the number of loops"
    ECHO "and press Return: " NOLINE
    SETENV >NIL: loop{$$} ?
ELSE
    ; there was an argument, so store its value
    ECHO >ENV:Loop{$$} {loop}
ENDIF
;
LAB start                                ; top of loop
ECHO "Loop #" NOLINE                      ; here, substitute the
TYPE ENV:Loop{$$}                         ; commands to repeat
EVAL <ENV:Loop{$$} >NIL: TO=T:Qwe{$$} VALUE2=1
OP=- ?
TYPE >ENV:Loop{$$} T:Qwe{$$}
```

```

IF VAL $loop{$$} GT 0
    SKIP start BACK          ;loop not finished yet
ENDIF
;
DELETE ENV:loop{$$} T:Qwe{$$} QUIET          ; clean up
ECHO "Done"

```

If you invoke this script without providing a number as an argument, you are asked for input and this value is used as the initial loop number. If you do provide a number, as in:

```
1> EXECUTE ALoop 5
```

the following results are displayed:

```

Loop #5
Loop #4
Loop #3
Loop #2
Loop #1
Done

```

The only action inside the loop is to display the current loop count. However, you can insert more meaningful actions using the Loop{\$\$} environment variable.

The first IF block checks whether an argument was given when the script was invoked. If not, the script prompts for a value. In either case, the script stores the value in the ENV:Loop{\$\$} variable file. The {\$\$} operator appends the process number to the name Loop to create a unique file name to avoid potential conflicts while multitasking. For example, the file name in ENV: might be Loop4.

Within the loop, an ECHO command coupled with a TYPE command displays Loop # followed by the number given as the loop argument. The first time through the loop, it displays **Loop #5**.

The EVAL command takes the number in the ENV:Loop{\$\$} file as <value1>, making the question mark at the end of the line necessary. <Value2> is 1 and the operation is subtraction. The output of the EVAL command is sent to the T:Qwe{\$\$} file. The next TYPE command sends the value in T:Qwe{\$\$} to the ENV:Loop{\$\$} file. The effect of these two lines is to subtract one from the value in ENV:Loop{\$\$}.

The IF statement instructs the script to start over as long as the value for Loop(\$\$) is greater than 0. This results in the Loop # line being printed again showing the new value.

The script continues until Loop(\$\$) is equal to 0. At the end of the script, the two temporary files are deleted.

Using PIPE:

To get a listing of one device's contents to another process:

From process 1:

```
1> LIST Work: TO PIPE: ALL
```

From process 2:

```
2> TYPE pipe:
```

To gather the results of several C compilations:

```
1> sc >pipe:ll milk.c
1> sc >pipe:ll snap.c
1> sc >pipe:ll crackle.c
1> sc >pipe:ll pop.c
1> TYPE pipe:ll
```

To use channel names:

```
1> LIST >pipe:crazy
```

This lists to a pipe called "crazy".

```
1> COPY #?.c TO >pipe:all_c/32000
```

This specifies a channel called "all_c" and a buffer size of 32000 bytes.

To set a limit on the number of buffers to 5:

```
1> DIR >pipe://5
```

This creates a channel without a channel name and allows only 5 buffers.

Recursive AmigaDOS Command Scripts

To create a script that allows you to apply any AmigaDOS command to the contents of a directory, including all its subdirectories and their contents, enter this script and save it as **S:RPAT**:

```
.KEY COM/A,PATH,OPT,RD
; enter the command for COM, enter the path to the
; directory tree to process for PATH, if it is not
; the current directory, and enter an option for
; the command as OPT. You do not need to enter
; anything for RD.
;
.BRA {
.KET }
FAILAT 21      ; do not stop when List finds nothing
;
; first scan for files, then generate new script:
LIST >T:trf{$$} "{PATH}" FILES LFORMAT="{COM}
*"%p%n*" {OPT}"
IF EXISTS T:trf{$$}
    ; files were found, execute the new script and
    ; clean up
    EXECUTE T:trf{$$}
    DELETE T:trf{$$}
ENDIF
;
; list any subdirectories, call RPAT for each one:
LIST >T:trd{$$}{RD} "{PATH}" DIRS LFORMAT="RPAT
*"%COM)*" *"%p%n*" *"%OPT)*" RD={RD}"
; the RD argument appends periods to the names of
; the temporary script files to distinguish each
; level
;
IF EXISTS T:trd{$$}{RD}
    ; subdirectories exist, execute new script
    ; file and clean up
    EXECUTE T:trd{$$}{RD}
    DELETE T:trd{$$}{RD}
ENDIF
```

Make sure the RPAT script has its s protection bit set and that RPAT is in the search path when you run it. Examine the S:SPAT and S:DPAT scripts that come with your system for examples of similar techniques.

Appendix A

Error Messages

This appendix lists AmigaDOS errors with their probable causes and suggestions for recovery. These error messages are the output from the system when your program fails or if a command is not executed as the result of a user error. Error messages differ from requesters, which are messages from the system that allow you to enter specific corrections, changes, or input so that the program, script, or command can continue execution. A requester that is not satisfied produces an error.

Error	Message	Probable Cause	Recovery Suggestion
103	Not enough memory available	Not enough memory in your Amiga to execute the operation. Memory may be fragmented.	Close unnecessary windows and applications and re-issue the command. Reboot if this does not work. You may need to add more RAM to your system.
115	Bad number	The command requires a numerical argument.	Use the correct command format.
116	Required argument missing	The command requires an argument that you did not supply.	Use the correct command format.
117	Value after keyword missing	Keyword was specified with no argument.	Use the correct command format.

Error	Message	Probable Cause	Recovery Suggestion
118	Wrong number of arguments	Too few or too many arguments.	Use the correct command format.
119	Unmatched quotes	You have an odd number of quotation marks.	Place double quotation marks at the beginning and end of the path or string.
120	Argument line invalid or too long	Your command line is incorrect or contains too many arguments.	Use the correct command format.
121	File is not executable	You misspelled the command name or the file is not a loadable (program or script) file.	Retype the file name, ensuring that the file is a program file. To execute a script, the s bit must be set or the EXECUTE command must be used.
202	Object is in use	The specified file or directory is being edited by another application or is assigned.	Stop the application using the file or directory or remove the assignment.
203	Object already exists	The name specified is assigned to another file or directory.	Use another name or delete the existing file or directory first.
204	Directory not found	AmigaDOS cannot find the specified directory.	Check the directory name and location (use DIR if necessary).
205	Object not found	AmigaDOS cannot find the specified file or device.	Check the file name (use DIR) or the device name (use INFO).

Error	Message	Probable Cause	Recovery Suggestion
206	Invalid window description	Occurs when specifying a window size for a Shell, ED, or ICONX window. The window may be too big or too small or you omitted an argument. Also occurs with the NEWSHELL command, if a device name is supplied that is not a window.	Use the correct window specification.
209	Packet request type unknown	The device handler cannot do the requested operation. For example, the console handler cannot rename things.	Check the request code passed to device handlers for the appropriate request.
210	Object name invalid	There is an invalid character in the file name or the file name is too long.	Retype the name; do not use any invalid characters or exceed the maximum length.
212	Object is not of required type	You may have specified a file name for an operation that requires a directory name, or vice versa.	Use the correct name and command format.

Error	Message	Probable Cause	Recovery Suggestion
213	Disk not validated	If you have just inserted a disk, the disk validation process may be in progress. It is also possible that the disk is corrupt.	Wait for the validation process to finish. Watch for drive light to turn off. Allow a minute for floppy disks and several minutes for hard disks. Corrupt disks cannot be validated. If corrupted, try retrieving and copying the files to another disk.
214	Disk is write-protected	The plastic tab is in the write-protect position or the disk has been locked.	Remove the disk, move the tab, and reinsert the disk, use a different disk, or use LOCK OFF command.
215	Rename across devices attempted	RENAME can move a file from one directory to another, but not from one volume to another.	Use COPY to copy the file to the destination volume. Delete it from the source volume, if desired. Then use RENAME, if desired.
216	Directory not empty	You tried to delete a directory that contains files or subdirectories.	Use the ALL option of DELETE if you wish to delete the directory and its contents.
217	Too many levels	Directory nesting is too deep.	Reorganize directories so that there are fewer levels or change directories in stages to reach the desired level.

Error	Message	Probable Cause	Recovery Suggestion
218	Device (or volume) is not mounted	If the device is a floppy disk, it has not been inserted in a drive. If it is another type of device, it has not been mounted, or the name is misspelled.	Insert the correct floppy disk, mount the device, check the spelling of the device name, revise your MountList/mount file, or assign the device name appropriately.
219	Seek error	An error occurred while processing a file.	Be sure that you only SEEK within the file. You cannot SEEK outside the bounds of the file.
220	Comment is too long	Your filenote has exceeded the maximum number of characters (79).	Use a shorter filenote.
221	Disk is full	There is not enough room on the disk to perform the requested operation.	Delete unnecessary files or directories or use a different disk.
222	Object is protected from deletion	The d (deletable) protection bit of the file or directory is clear.	If you are certain that you want to delete the file or directory, use PROTECT to set the d bit or use the FORCE option of DELETE.
223	File is write protected	The w (writable) protection bit of the file is clear.	If you are certain that you want to overwrite the file, use PROTECT to set the w bit.
224	File is read protected	The r (readable) protection bit of the file is clear.	Use PROTECT to set the r bit of the file.

Error	Message	Probable Cause	Recovery Suggestion
225	Not a valid DOS disk	The disk in the drive is not an AmigaDOS disk, it has not been formatted, or it is corrupt.	Be sure you are using the correct disk. If the disk worked previously, use a disk recovery program to salvage its files. Format unformatted disks.
226	No disk in drive	The disk is not inserted in the specified drive.	Insert the appropriate disk in the specified drive.
232	No more entries in directory	The AmigaDOS call EXNEXT has no further entries in the directory you are examining.	Stop calling EXNEXT.
233	Object is soft link	Attempt was made to access a soft-link for a device that does not support it.	No recovery.
235	Bad load file hunk	The program loaded is corrupted.	Load a new or original copy of the program.
241	Record lock collision	Another application is accessing the database.	Try accessing the database again.
242	Record lock timeout	Another application has the database entry locked.	Try again or quit the other application and retry.
303	Buffer overflow	Occurs if pattern matching string is too long.	Make pattern matching string shorter.
304	***Break	Occurs if program stopped via Ctrl+C.	No recovery
305	File not executable	The e (executable) bit of the file is clear.	Same as Error 121.

Appendix B

Additional Amiga Directories

In addition to the AmigaDOS commands, there are other files and directories on your Workbench disk. This chapter includes the following:

- DEVS:
- S:
- L:
- FONTS:
- LIBS:
- REXX:
- LOCALE:
- ENVARC:
- ENV:
- CLIPS:
- T:
- Classes
- C:

The drawers contained in DEVS: are described in the *Workbench User's Guide*.

You do not need a detailed understanding of the contents of the directories listed here. Unless specifically directed otherwise, you can safely ignore them. However, you should know their purposes and locations in case you inadvertently delete or rename a file in a directory or need to copy something to the appropriate directory.

Figure B-1 illustrates the standard directory structure of a hard disk Amiga system. Directories with icons visible from Workbench (drawers) are shown on the left; other directories are on the right. The standard contents and structure of these directories may change as Commodore adds, changes, or removes resources.

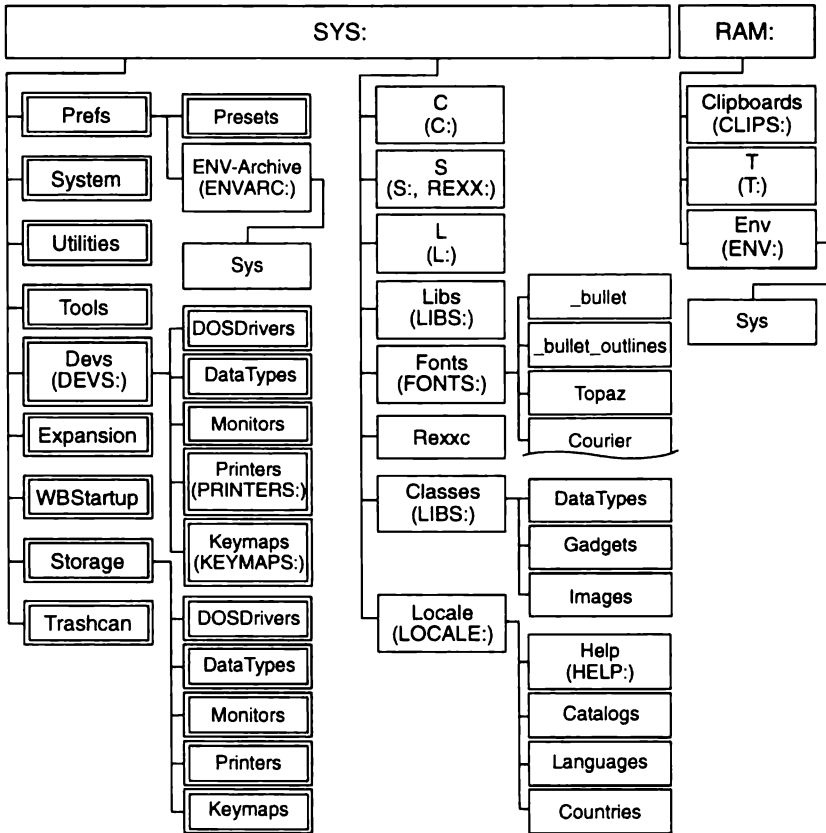


Figure B-1. Hard Disk System Directory Tree

Note On most floppy based systems, the LOCALE:, FONTS:, and Storage directories are separate volumes.

Most of these directories are automatically assigned to the SYS: volume or to the Ram Disk. These directories, as well as SYS:, can be ASSIGNED to different volumes when necessary.

For example, you can assign FONTS: to a particular disk, such as FontDisk:. Most applications automatically look for the fonts that they need in the FONTS: directory, regardless of where that is. By changing the FONTS: assignment, you can allow applications to use the fonts on FontDisk:

DEVS:

In addition to the DOSDrivers, Keymaps, Printers, Monitors, and DataTypes drawers described in the *Workbench User's Guide*, the DEVS: drawer contains files and subdirectories that pertain to the devices that can be used with the Amiga.

Note that you can refer to the DEVS:Keymaps and DEVS:Printers drawers by their assigned names KEYMAPS: and PRINTERS:, respectively.

Device Files

The following lists the .device files in DEVS: and their functions:

clipboard.device	Controls access to CLIPS:.
parallel.device	Controls access to the parallel port.
printer.device	Controls access to the printer device.
serial.device	Controls access to the serial port.
mfm.device	Controls access to MS-DOS disks with CrossDOS.

For more information on the .device files, see the *Amiga ROM Kernel Reference Manuals* published by Addison-Wesley.

Other Files

The following additional files are found in DEVS:

system-configuration	Holds certain Preferences configuration data needed when booting.
postscript_init.ps	Holds information needed to initialize a PostScript printer when using PrinterPS.

Using Mount Files or a MountList

To access new devices, the Amiga must be informed when any are added. These may be physical devices, such as a tape drive, or software (logical) devices, such as a recoverable RAM disk. There are several ways to do this:

- Placing a driver in the Expansion drawer
- Placing a mount file in the DOSDrivers drawer
- Making an entry for the device in a MountList file and using the MOUNT command

A mount file represents a device, handler, or file system. Standard devices have their own mount files with icons in the DOSDrivers drawer in DEVS:. These are mounted automatically during the standard Startup-sequence. Alternatively, devices can use the MOUNT command to read a MountList entry that determines the characteristics of the device.

The need to use a MountList and the MOUNT command has been eliminated by the mount file method used in Amiga system software Release 2.1 and beyond. Rather than requiring a MountList file with entries for each device you want to mount, DEVS: now contains the DOSDrivers drawer, which holds a separate mount file or DOS driver for each device. The contents of a mount file are essentially the same as an individual MountList entry.

You can, however, continue to use a MountList to mount devices. Copy the MountList to DEVS: and remove any DOS drivers in the DOSDrivers drawers that have the same name as one of your MountList entries. (A mount file overrides a MountList entry of the same name.)

Creating a Mount File or MountList Entry

The following information on mount files also applies to MountLists, except as noted.

Mount files contain keywords describing the device, handler, or file system, as well as values for those keywords. Some keywords apply only to a filesystem or a handler. If a keyword is omitted, a default value is used. Be sure that the default value is appropriate for the device.

The following are rules for creating a mount file or MountList:

- The file must be a plain ASCII text file.
- The mount file name must be the name of the device; the name of a MountList should be MountList.
- Each entry in a MountList must start with the name of the device. Omit this for a mount file.
- Keywords are followed by an equals sign (=).
- Keywords must be separated by a semicolon or be placed on separate lines.
- Comments are allowed in standard C style (that is, comments start with /* and end with */).
- Each MountList entry must end with the # symbol. Omit this for a mount file.

When creating a new mount file or MountList entry, refer to the documentation that came with the device or start with an example of one for a similar device. Change or add only the necessary keywords.

The following table lists keywords and their default values supported in a mount file or MountList and their functions. Default values are shown in angle brackets.

Keyword	Function
Handler=<none>	A handler entry (for example, Handler = L:queue-handler).
EHandler=<none>	An environment handler entry.
FileSystem=<none>	A file system entry (for example, FileSystem = L:CrossDOSFileSystem).
Device=<none>	A device entry (for example, Device = DEVS:mfm.device). This argument is required to mount a file system. There is no default value for Device; you must supply a value.
Priority=<10>	The priority of the process; 5 is good for handlers, 10 for file systems.
Unit=<0>	The unit number of the device (for example, 0 for PC0:).
Flags=<0>	Flags for OpenDevice (usually 0).
Surfaces=<none>	The number of surfaces (2 for floppy devices, varies for hard drives). This argument is required to mount a file system. There is no default value for Surfaces; you must supply a value.
SectorsPerBlock=<none>	Defines the number of physical disk sectors in each logical block used by the file system.
SectorsPerTrack=<none>	The number of blocks per track. This argument is required to mount a file system. There is no default value for SectorsPerTrack; you must supply a value.
SectorSize=<512>	Specifies the number of bytes in a block on the device. Most devices use a 512 byte block; however, some devices use other sizes (for example, CD-ROMs use 2048 bytes).
Reserved=<2>	The number of blocks reserved for the boot block; should be 2.
Interleave=<0>	Interleave value; varies with the device.
LowCyl=<none>	Starting cylinder to use. This argument is required to mount a file system. There is no default value for LowCyl; you must supply a value.

Keyword (cont'd)	Function (cont'd)
HighCyl=<none>	Ending cylinder to use. This argument is required to mount a file system. There is no default value for HighCyl; you must supply a value.
Stacksize=<600>	Amount of stack allocated to the process.
Buffers=<5>	Number of initial 512-byte cache buffers. Increase this for higher disk performance if you have RAM to spare.
BufMemType=<3>	Memory type used for buffers; (0 and 1 = Any, 2 and 3 = Chip, 4 and 5 = Fast).
Mount=<0>	See the description of ACTIVATE, which is a synonym for MOUNT.
MaxTransfer=<0x7fffff>	The maximum number of bytes transferred at one time with any file system. Use MaxTransfer for compatibility with older hard drive systems.
Mask=<0xfffffff>	Address Mask to specify memory range that DMA transfers can use at one time with any file system. Use Mask for compatibility with older hard drive systems.
GlobVec=<2>	A global vector for the process; -1 is no Global Vector (for C and assembler programs), 0 sets up a private GV; if the keyword is absent, the shared Global Vector is used. Omit this keyword for Amiga file system devices.
Startup=<none>	A string passed to the device, handler, or file system on startup as a BPTR to a BSTR.
Activate=<0>	If a positive value, ACTIVATE loads the device or handler immediately rather than waiting for first access. Synonymous with MOUNT.
BootPri=<0>	A value that sets the boot priority of a bootable and mountable device. This value can range from -129 to 127. By convention, -129 indicates that the device is not bootable and is not automatically mounted.

Keyword (cont'd)	Function (cont'd)																								
DosType= <0x444F5300>	Indicates the type of file system, giving hexadecimal ASCII codes for three letters and a concluding number as follows: <table border="1"> <thead> <tr> <th>Value</th> <th>ASCII</th> <th>File System</th> </tr> </thead> <tbody> <tr> <td>0x444F5300</td> <td>DOS0</td> <td>Original (OFS)</td> </tr> <tr> <td>0x444F5301</td> <td>DOS1</td> <td>FastFileSystem (FFS)</td> </tr> <tr> <td>0x444F5302</td> <td>DOS2</td> <td>International Mode OFS</td> </tr> <tr> <td>0x444F5303</td> <td>DOS3</td> <td>International Mode FFS</td> </tr> <tr> <td>0x444F5304</td> <td>DOS4</td> <td>Directory caching International Mode OFS</td> </tr> <tr> <td>0x444F5305</td> <td>DOS5</td> <td>Directory caching International Mode FFS</td> </tr> <tr> <td>0x4D534400</td> <td>MSD0</td> <td>MS-DOS</td> </tr> </tbody> </table>	Value	ASCII	File System	0x444F5300	DOS0	Original (OFS)	0x444F5301	DOS1	FastFileSystem (FFS)	0x444F5302	DOS2	International Mode OFS	0x444F5303	DOS3	International Mode FFS	0x444F5304	DOS4	Directory caching International Mode OFS	0x444F5305	DOS5	Directory caching International Mode FFS	0x4D534400	MSD0	MS-DOS
Value	ASCII	File System																							
0x444F5300	DOS0	Original (OFS)																							
0x444F5301	DOS1	FastFileSystem (FFS)																							
0x444F5302	DOS2	International Mode OFS																							
0x444F5303	DOS3	International Mode FFS																							
0x444F5304	DOS4	Directory caching International Mode OFS																							
0x444F5305	DOS5	Directory caching International Mode FFS																							
0x4D534400	MSD0	MS-DOS																							
Baud=<1200>	Serial device baud rate.																								
Control=<0>	Serial device word length, parity, and stop bits.																								
Forceload=<0>	Forces a file system to be loaded from disk even though a suitable entry is in the resource list. If 0 (the default), check the resource list before the disk. If 1, always load from disk.																								

S: Directory

The S: directory is generally reserved for AmigaDOS and ARexx scripts. However, you may place non-script files in S: or place script files in other directories. In addition to the Startup-sequence file, the User-startup file that you create, and Shell-startup files, the S: directory also contains the scripts described in this section.

ED-Startup

This file contains ED commands used to configure the ED text editor, assigning the default function key options. Key assignments can be customized by editing this file. Removing or renaming this file

activates expanded ED command menus; see Chapter 4 for instructions on how to do this.

Other files containing ED commands can be stored in S: for use by the WITH keyword of ED, which allows ED command files to perform custom editing operations.

SPat, DPat

These scripts allow pattern matching with commands that do not normally support it. When run with a command, SPat and DPat use the LIST command to create temporary script files in the T: directory and then execute the scripts. SPat and DPat can be used within command aliases.

SPat adds pattern matching to single-argument commands. For example, to use ED to edit all the files in the S: directory beginning with the letter s, enter:

```
1> SPat ED S:s#?
```

A script similar to the following is generated:

```
ED "s:Shell-startup"  
ED "s:SPat"  
ED "s:Startup-sequence"
```

SPat executes the script, invoking ED three times to display the files.

DPat adds pattern matching to double-argument commands. After DPat and the command name, enter the two arguments separated by a space, using the wildcards required to produce the desired matches.

PCD

Like the CD command, the PCD script changes the current directory. However, PCD also remembers the directory from which you are changing, so you can return to it without entering the entire path. Each Shell has an independent PCD memory.

The first time you use PCD in a given Shell, you must give it the path to a directory. When entered with this path argument, PCD has the same effect as CD, changing to the named directory, but it also stores the path to the directory from which you changed. You can then

change the current directory by the usual methods. To return to the initial directory, enter PCD alone. For example:

```
1.System:> PCD Work:Paint/24bit
1.Work:Paint/24bit> PCD
1.System:>
```

Subsequent invocations of PCD switch to the directory in which you last used PCD:

```
1.System:> PCD
1.Work:Paint/24bit> /
1.Work:Paint> PCD
1.System:>
```

PCD uses the assign list to store the remembered directory. When you use PCD, the list contains the assignment **from<n>**, where <n> is the Shell number. Using PCD with no argument before establishing the first from directory produces an error requester.

For further examples of PCD, see Chapter 8.

L: Directory

This directory contains device handlers, which are software modules that go between AmigaDOS and the devices used by the Amiga. However, most handlers are treated as if they are actual physical devices and they are referred to by their device name.

Handlers must be named in the mount file or MountList for their respective devices. Handlers are called and manipulated by programs, not users. New handlers can be supplied with some devices or programs and should be copied to the L: directory.

Aux-Handler

The Aux-Handler provides unbuffered serial input and output. It is essentially a console handler that uses the serial port rather than the Amiga screen and keyboard.

The DOSDrivers mount file for AUX is:

```
Handler = L:Aux-handler
Stacksize = 1000
```

Priority = 5

You can use Aux-Handler to use a serial terminal with your Amiga. For example:

```
1> NEWSHELL AUX:
```

Queue-Handler (PIPE:)

The Queue-Handler is an input/output mechanism used to provide I/O communication between programs. It creates an interprocess communication channel named PIPE. For more information about PIPE:, see Appendix D.

Port-Handler

The Port-Handler is the AmigaDOS interface for the SER:, PAR:, and PRT: devices.

When accessing SER:, you can supply settings for the baud rate and control information. The form for this is SER:<baud/control>, where baud is a number representing the baud rate and where control is a three character sequence indicating the following:

Number of read/write bits	First character; either 7 or 8
Parity	Second character; N (no parity), O (odd parity), E (even parity), M (mark parity), S (space parity)
Number of stop bits	Third character; either 1 or 2

For example:

```
SER:9600/8N1
```

connects to the serial port, sets the baud rate to 9600 with 8 bit data, no parity, and one stop bit.

If you specify no baud rate or control values when accessing SER:, the values set in the Serial Preferences editor are used.

CrossDOSFileSystem

The CrossDOSFileSystem is required to use CrossDOS.

FileSystem_Trans

The FileSystem_Trans directory contains the following files required for CrossDOS text translation:

DANSK.crossdos	Filters Danish text files
INTL.crossdos	Preserves international characters
MAC.crossdos	Converts Apple Macintosh ASCII files

CDFileSystem

The CDFileSystem is required to use a CD-ROM drive.

FONTS:

FONTS: is the disk or directory that contains the information for all of the different styles of fonts available to the Amiga. Font information is stored differently for bitmap and outline fonts.

Bitmap Fonts

For each bitmap font, there is a subdirectory and a .font file. The font subdirectory contains files for the different point sizes that are available. Each of these files contains the bitmap representation of every character in the font at that point size.

For example, for the Emerald font, there is an Emerald directory and an Emerald.font file. The Emerald directory contains two files: 17 and 20. The files contain the data needed for the 17 point Emerald font and the 20 point Emerald font, respectively. The Emerald.font file contains the list of point sizes and any available styles, such as bold and italics for the font.

Many word processor or desktop publishing programs contain additional fonts that you should copy to your FONTS: directory. After you add new fonts to FONTS:, run the FixFonts program to create the .font file for the new additions.

The Topaz font is the default font used by the Amiga. In addition to existing in FONTS:, Topaz 8 and Topaz 9 are built into ROM so that text can always be displayed; however only Topaz11 is in the Topaz directory.

Outline Fonts

The Compugraphic Intellifont outline font system used on the Amiga stores font data differently. As with the bitmap fonts, there is a .font file for each typeface. There is also a .otag file for each. The two subdirectories `_bullet` and `_bullet_outlines` contain the actual outline information for the typefaces installed on your system. You should not try to manipulate these files directly. Always use the Intellifont program or your applications' font installation tools to manage outline fonts.

Note Not all Amiga systems have outline fonts installed.

LIBS: Directory

LIBS: contains libraries of software routines and math functions commonly used by the operating system and applications. The files found in the LIBS: directory are:

.library File	Function
amigaguide.library	Functions used by the AmigaGuide hypertext system.
asl.library	File, font, and screen mode requester functions.
bullet.library	Library functions for finding and loading outline fonts.

.library File (cont'd)	Function (cont'd)
commodities.library	Functions used by Commodities Exchange programs.
datatypes.library	Functions for enabling manipulation of the file types in DEVS:DataTypes.
diskfont.library	Library functions for finding and loading font files.
iffparse.library	Functions to read IFF files.
locale.library	Functions for using localization features in the Amiga operating system.
lowlevel.library	Library to aid in game programming.
mathieeedoubbas.library	Double-precision IEEE math routine functions for basic functions (addition, subtraction, and so forth).
mathieeedoubtrans.library	Double-precision IEEE math routine functions for transcendental functions (sine, cosine, and so forth).
mathieeesingtrans.library	Fast single-precision IEEE math routine functions.
mathtrans.library	FFP transcendental function math routine functions.
realtime.library	Function to synchronize multimedia events, such as music and animation.
rexksupport.library	Functions used by ARexx.
rexksyslib.library	Main ARexx functions.
version.library	Contains current software version and revision information.
68040.library	Functions needed on Amigas using the 68040 microprocessor chip.

REXX:

REXX: is another name normally assigned to the SYS:S directory in addition to S:. Storing any ARexx programs you write in REXX: allows you to execute them without entering the full path. Create a new directory and reASSIGN REXX: to it if you wish to keep ARexx programs separate from AmigaDOS scripts.

LOCALE:

LOCALE: is where the Amiga looks for language and country files when it needs to display non-English text. On floppy systems, LOCALE: is the Locale floppy. On hard disk systems, it is the SYS:Locale directory.

LOCALE: contains four directories:

- | | |
|---------------------|---|
| Countries | Contains a .country file for all available countries. These files hold country-specific information, such as date/time format and monetary symbols. |
| Languages | Contains .language files for the available languages. These files hold general language-specific information, such as the days of the week. |
| Catalogs | Contains subdirectories for the available languages, each of which contains a Sys directory. This directory holds translated text for that language. Catalogs/<language>/Sys holds menu, gadget, and message text for that language. |
| Help (HELP:) | Contains subdirectories for the available languages, each of which contains a Sys directory. This directory holds translated text for that language. Help/<language>/Sys is reserved for AmigaGuide text files for any applications that have AmigaGuide help in that language. |

ENVARC:

ENVARC: is the name assigned to the directory SYS:Prefs/Env-Archive. ENVARC: always contains a Sys directory, in which each of the Preferences editors stores the .prefs file created when you save a setting with its Save gadget. Custom default icons created in IconEdit are also saved here. (Named settings saved with the Save As menu item in an Editor are stored by default in the Prefs/Presets drawer.)

Other Workbench programs that allow you to save configuration settings, such as MultiView, also place their files in ENVARC:.

ENV:

ENV: is the directory RAM:Env, into which the contents of ENVARC: are copied when booting. Preferences settings activated with Save or Use gadgets are stored in ENV:. Global environment variables are also stored here, as small files.

CLIPS:

The directory RAM:Clipboards has the assigned name CLIPS:. It stores information clipped with the Cut or Copy items on a program's Edit menu.

T:

T: is the RAM:T directory, which may be used by scripts and some commands for storing miscellaneous temporary files.

Classes

The Classes directory stores information related to the object-oriented features of the Workbench, such as the DataTypes system upon which MultiView is based. Classes contains the directories DataTypes and Gadgets.

C:

C: is the SYS:C directory, which stores non-internal AmigaDOS commands. It is always in the search path.

Appendix C

Using Floppy-Only Systems

The limitations of working with floppy disk systems and how to swap disks are described in the *Workbench User's Guide*. This appendix explains how to use AmigaDOS to minimize disk swapping and maximize your available work space with the following:

- Making commands resident
- Preloading resources into memory
- Assigning a path
- Removing files from your Workbench disk
- Using the Ram Disk
- Using RAD:

Making Commands Resident

If you have a floppy-based system, we recommend that you make frequently-used commands resident for quick access. Resident commands, which reside in the Amiga's memory, do not require the insertion of the Workbench disk each time the command is used. Use the **RESIDENT** command to copy a command into the Amiga's memory.

Since resident commands use RAM space, the number of commands you can make resident depends on the amount of RAM in your system. Before making a command resident, use the **LIST** command to get an approximation of the memory it uses. For example, entering **LIST C:COPY** produces output similar to this:

```
Directory "Sys:C" on Monday 15-Jun-92
copy          5496 --p-rwed 03-Jun-92 17:22:02
```

The size of the file is shown to the right of the file name. The COPY command uses approximately 5.5 KB of RAM if made resident.

To conserve as much memory as possible for your applications, make resident only the AmigaDOS commands and programs that are frequently used and are not built into Workbench. These include ASSIGN, ED, STATUS, Format, and DiskCopy. If you have sufficient memory and you regularly use AmigaDOS, you can also make resident commands that have Workbench equivalents, such as COPY, DELETE, DIR, LIST, MAKEDIR, and RENAME.

Do not make resident commands that are not used often, such as the startup command ADDBUFFERS. The following commands cannot be made resident: BINDDRIVERS, CONCLIP, IPREFS, LOADRESOURCE, LOADWB, and SETPATCH.

For more information on making commands resident, see the RESIDENT command in Chapter 6.

Preloading Resources

Preloading resources into free memory makes them available without inserting the floppy disk each time they are needed. Using the LOADRESOURCE command, you can preload libraries, devices, fonts, and catalogs into RAM. These resources remain in RAM as long as they are in use or until the system flushes them due to low memory. Resources other than devices can be locked into RAM, forcing them to stay there until you unlock them.

For more information on the LOADRESOURCE command, see Chapter 6.

Using ASSIGN's PATH Option

Using the PATH option of the ASSIGN command reduces disk swaps by directing AmigaDOS to search any disk in a given drive for the command it requires. Normally AmigaDOS displays a requester asking for the original boot disk even when the currently inserted disk contains the necessary file.

Add the following commands to your User-startup script to use the PATH option:

```
ASSIGN DEVS: DF0:Devs PATH
ASSIGN C: DF0:C PATH
ASSIGN L: DF0:L PATH
ASSIGN FONTS: DF0:Fonts PATH
```

Then copy these directories from the Workbench disk onto application disks that require them.

For more information on the PATH option of the ASSIGN command, see the ASSIGN command in Chapter 6.

Removing Files From Your Workbench Disk

Copying printer drivers, fonts, or Preferences editors to your Workbench disk makes them readily available. However, a **Volume is full** requester can result if you try to copy too much to your Workbench disk. It is possible to delete files from the Workbench disk to make room for other files.

Note When working directly with your Workbench disk, use a backup copy of the original disk that came with your system. Your original, unchanged Workbench disk should never be altered in case you need to restore a file or reinstall the Workbench.

Deleting system software results in some limitation of your Amiga's capabilities and may cause an error if an application attempts to use a file that was deleted. If you experience an unexpected requester or error, repeat the same operation using the original Workbench disk. If no error occurs, the application uses something that was deleted and it should be restored.

Document all changes that you make to your system disks. Adding a comment in the disk's User-startup file can remind you that you are working with a non-standard Workbench disk.

Files You Can Delete

- Delete files from your Workbench disk starting with the least critical files, such as the Clock program, and programs that do not apply to your system, such as C:MAGTAPE if you do not use a tape drive and System/NoFastMem if you do not have any Fast ("other") memory.
- Delete the AmigaDOS text editor C:EDIT to free approximately 18 KB of disk space.
- Delete L:CrossDOSFileSystem and DEVS:mfm.device if you do not need to use CrossDOS, freeing approximately 32 KB.
- Delete with caution the contents of the Classes directory to save about 115 KB; however, this contains subdirectories that other programs use. The Classes/DataTypes directory, for example, is used by MultiView and should not be deleted if you wish to use that program to view files. The Classes/Gadgets directory is used by the Palette Preferences editor and should not be deleted if you wish to change your color settings.

Files To Avoid Deleting

The following should not be deleted under any circumstances:

- Any directory normally found on the Workbench disk; delete only the files within directories
- DEVS:parallel.device
- DEVS:printer.device
- DEVS:serial.device
- LIBS:asl.library
- LIBS:commodities.library
- LIBS:diskfont.library
- LIBS:iffparse.library
- LIBS:locale.library (if your language and country are not English and united_states)
- LIBS:68040.library (if you have a Commodore 68040 board)

- S:Startup-sequence
- L:Port-handler
- Any non-Internal command that appears in the default Startup-sequence

Do not delete any file whose purpose you do not know. Do not delete more files than necessary to fit the new material you intend to add to the disk.

Although deleting everything in REXXC, in addition to System/RexxMast, System/RexxMast.info, LIBS:rexsyslib.library, and LIBS:rexsupport.library frees almost 50 KB of disk space, we do not recommend this since many Amiga applications use ARexx and must call upon these files.

Using the Ram Disk

RAM: or Ram Disk, which is represented on the Workbench screen by the Ram Disk icon, is an area of the Amiga's internal memory that is set up as a file storage device. Files, directories, and—available memory permitting—entire floppy disks can be copied to RAM: for temporary storage.

The size of RAM: is dynamic. It is never any larger than necessary to hold its contents. Therefore, it is always 100% full. Its maximum size is limited by the amount of free memory.

The primary advantage of RAM: is speed. Since it is electronic, rather than mechanical, storage and retrieval are almost instantaneous. The disadvantage of RAM: is that data stored in RAM: does not survive when the computer is turned off or rebooted.

Applications commonly use RAM: for the storage of temporary files created during the use of the program or backup files created when the program is exited. RAM: can also be used for the storage of experimental script files, as a destination for testing command output, and whenever the creation of a file on an actual disk is too slow, risky, or inconvenient.

Copying From One Disk to Another

The most efficient way to copy information from one disk to another on a single-floppy system is to use the Ram Disk:

1. Copy the information from the source floppy disk to the Ram Disk.
2. Remove the source floppy disk from the drive.
3. Insert the destination disk.
4. Copy the information to it from the Ram Disk.

Be careful when using RAM: for storing important files. If the Amiga loses power, has a software failure, or you reboot, everything stored in RAM: is lost. Be sure when working with RAM: to regularly back up any important files on a floppy disk.

Note You cannot copy a disk to RAM: by dragging the source disk icon over the Ram Disk icon. To copy a disk to RAM:, open the Ram Disk icon and drag the floppy disk icon into the Ram Disk window to create a drawer with the name and contents of the floppy disk.

Recoverable Ram Disk

AmigaDOS also provides a recoverable Ram Disk, which has the device name RAD:. The contents of RAD: survive reboots and most software failures, making it a safer place for work files. (Data in RAD: is still lost if the computer is turned off.)

RAD: is not automatically created. To activate a recoverable RAM Disk, double-click on the RAD icon in the DOSDrivers drawer of the Storage directory. To start RAD: whenever you boot, copy the RAD icon to the Devs/DOSDrivers drawer on the Workbench disk. When RAD: has been activated, a disk icon labelled RAM_0 appears on the Workbench screen.

Unlike RAM:, the size of RAD: is fixed. The size is set in the RAD: mount file's HighCyl parameter. Change its size by entering a

different value for HighCyl. A HighCyl entry of 79 results in a RAD: with the same capacity as a normal 880 KB floppy disk.

Bootable RAD:

On an Amiga with more than 2 MB of RAM, you can create a floppy-size RAD:; the default as configured by the MountList is floppy-sized. By copying your Workbench files into this RAD: and reassigning to it all the directories normally assigned to the Workbench disk, it can be used as a recoverable Workbench-in-RAM. This allows you to reboot from RAD: instead of from the Workbench disk.

You can also set up multiple RAD: devices of different sizes by copying the RAD: mount file and changing the name and unit number.

Appendix D

Advanced AmigaDOS Features

The information in this appendix is intended for experienced AmigaDOS users. It includes the following:

- Customizing the window
- Customizing your Shell environment
- Using Escape sequences
- Customizing startup files

Customizing the Window

The Shell supports a WINDOW Tool Type in the Shell icon that allows you to specify the size, position, and features of the Shell window. The format of the Tool Type is as follows:

```
WINDOW=CON:x/y/width/height/title/option/options
```

For a description of the options and arguments for the Shell window specification, see the description of the NEWSHELL command in Chapter 6 and the examples in Chapter 8.

Public Screens - PUBSCREEN Option

Applications creating screens can mark them as public, enabling other applications and utilities to open windows on the same screen. AmigaDOS commands that have PUBSCREEN/K in their templates allow the commands to open windows on public screens.

For example, the Input Preferences editor template contains PUBSCREEN/K; open this editor on a public screen by entering the following:

```
1> INPUT PUBSCREEN "public screen name"
```

You must supply the name of the public screen. The internal names for public screens are given by applications and do not necessarily match the screen title printed in the screen's title bar. A screen's name can be the same as the application's name; however, consult the application's documentation to determine if it opens public screens and how it names them.

Customizing the Shell

You can customize your Shell environment by changing the S:Shell-startup file, which is a script that is executed each time a new Shell is opened. You can edit Shell-startup to set up command aliases and to change the Shell prompt.

Using Aliases

An alias is an abbreviation for a long and/or frequently used command. Aliases can be local or global. Local aliases are entered in a Shell window and are only recognized in that Shell. Global aliases are entered into the Shell-startup file and are recognized by all Shells.

The Alias format is as follows:

```
ALIAS <name> <string>
```

where <name> is the alias name to be entered at the Shell prompt to execute a command. The <string> is the command line to be executed.

See Chapter 6 for a full description of the ALIAS command and Chapter 8 for a list of useful aliases.

Changing the Prompt

The PROMPT command lets you customize the Shell prompt. By default, it shows the process number, a period, the current directory, a right angle bracket (>), and a space:

```
1.Workbench:>
```

The prompt can display almost anything, with or without the process number and directory information. The return code of the last command executed can be included. The prompt can contain escape sequences, allowing you to change text color and style in the prompt string or clear the screen.

Benefits of customizing your Shell prompt include making the prompt:

- Easier to distinguish from commands and their output
- Match a prompt style with which you are familiar
- More informative
- Shorter

See Chapter 8 for examples of how to use escape sequences to make the prompt more readable.

Using Escape Sequences

Escape sequences can control how the text appears in a console window, such as the text color, style (bold, italics, underline), and margins. AmigaDOS recognizes standard ANSI X3.64 sequences entered on the command line or embedded in strings. Escape sequences consist of one or more characters, sometimes with a numerical argument, prefaced by the escape character. Spaces are not normally used in the sequence of characters.

The escape sequence is shown using the following format:

Esc [#X

where:

- Esc** Represents the Escape key. Press Esc or substitute *E if you are in an application such as ED that uses the Esc key for its own purposes. When you press Esc, a reversed-color open bracket ([) appears in the console window.
- [** Represents the open bracket key, displayed in Figure D-1. If your country's keyboard does not have an open bracket key, press Alt plus the key shown, regardless of what is shown on the keycap.

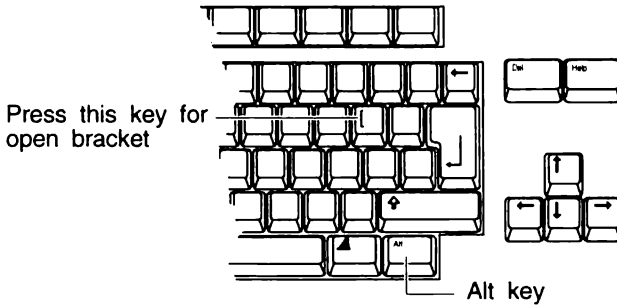


Figure D-1. Open Bracket Key Location

- #** Represents a numerical argument.
- X** Represents an alphabetic key. Escape codes are case-sensitive. If an upper case letter is shown, press Shift and the key. If a lower case letter is shown, press the unshifted key.

The standard escape sequences for console windows are:

Sequence	Action
----------	--------

Esc	Clears the window and resets all modes to defaults.
Esc[0m	Resets graphics modes to defaults.
Esc[1m	Turns on text boldface.
Esc[3m	Turns on text italic.

Sequence (cont'd)	Action (cont'd)
Esc[4m	Turns on text underlining.
Esc[7m	Turns on reverse video text.
Esc[8m	Makes text match background color.
Esc[22m	Turns off boldface.
Esc[23m	Turns off italics.
Esc[24m	Turns off underlining.
Esc[27m	Turns off reverse video.
Esc[28m	Returns the text color to normal.
Esc[30m	Turns on text color0 (background, default grey).
Esc[31m	Turns on text color1 (shadow, default black).
Esc[32m	Turns on text color2 (shine, default white).
Esc[33m	Turns on text color3 (accent, default blue).
Esc[3#m	Turns on text color# (4-7).
Esc[39m	Turns on text default color (color1).
Esc[40m	Turns on text background color0 (default grey).
Esc[41m	Turns on text background color1 (default black).
Esc[42m	Turns on text background color2 (default white).
Esc[43m	Turns on text background color3 (default blue).
Esc[4#m	Turns on text background color# (4-7).
Esc[49m	Turns on default text background color (color0).
Esc[#u	Sets maximum length of lines in window to #.
Esc[#t	Sets maximum number of lines in window to #.
Esc[#x	Starts text # pixels from left edge of window.
Esc[#y	Starts text # pixels from top edge of window.

The escape sequence is executed when you press Return or when the string containing the sequence is printed.

Certain characters cannot normally be used in string arguments for AmigaDOS commands. Preceding these characters with an asterisk allows them to be used in string arguments for most commands, as follows:

- *E Represents the Escape key in a string.
- *N Forces a new line in the output.
- ** Allows a quotation mark character inside quotation marks.
- ** Gives a single asterisk.

Customizing Startup Files

Each time your Amiga is booted, it executes the Startup-sequence script file Startup-sequence script file located in the S: directory. The Startup-sequence file allocates disk buffers, makes device assignments, reads saved Preferences settings, and performs other functions that configure the Amiga for use.

Because any errors introduced into the Startup-sequence file can cause a fatal disruption of the normal system startup, we strongly recommend that you do not alter your Startup-sequence file. Instead, we recommend that you create a file called User-startup User-startup file in the S: directory. Creating a User-startup file allows you to customize your system at startup while preventing any disruption of the normal booting process. This file is automatically executed by the Startup-sequence before opening Workbench.

Note Do not modify the original Startup-sequence file. Altering your Startup-sequence file can cause fatal system startup errors.

The User-startup and other startup files in the S: directory can be modified to run programs at startup, print special introductory messages, or automatically open a Shell window on the Workbench screen. Any AmigaDOS command can appear in a startup script, including commands to execute other scripts.

Refer to Chapters 6 and 7 for complete specifications about each command before making changes to any existing script.

Editing Startup Files

If you have a floppy-only system, make changes to your startup files only if you are working on a copy of your Workbench disk, not the original. If you make a mistake in your startup files and the execution of the Startup-sequence is aborted, only a Shell prompt remains. Normally, the FAILAT 21 command ensures that the Startup-sequence completes execution even after an error.

As long as you do not alter your standard Startup-sequence file, the possibility of a serious startup error is unlikely. If you attempt to boot from a disk that has no file named S:Startup-sequence or there is serious error in the startup process, you may see a screen similar to that shown in Figure D-2.

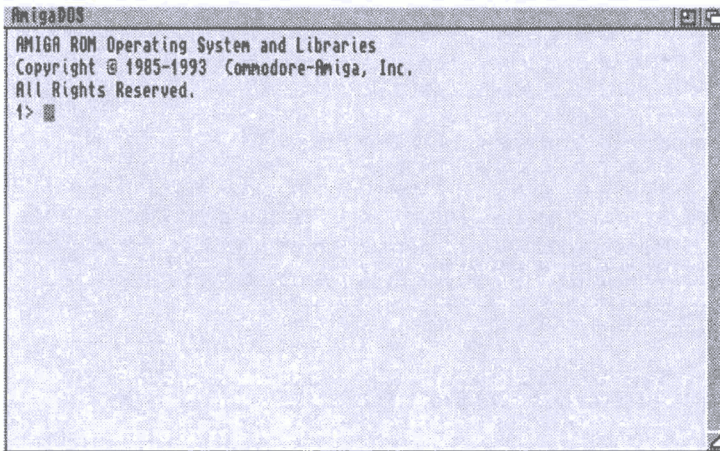


Figure D-2. AmigaDOS Boot Failure Shell Window

This is a Shell window at which you can enter commands, but from which your usual aliases and search path may not be available. This Shell allows you to try to edit the startup file that caused the problem. Because the system has not been fully configured, this can be difficult. It is usually easier to reboot using a different disk.

When editing your User-startup file:

- Be sure to use the correct command syntax. Test any commands you plan to insert into the User-startup in a Shell window first. If a command works properly in the Shell, it should work as expected in the User-startup.
- Pay attention to the order of commands:order of commands in script in the script. Some commands, such as ECHO and RESIDENT, can be put anywhere. However, when inserting commands that refer to directories and files, an error can result if you reference anything that has not yet been created, assigned, or given a valid path.
- Specify the complete path to directories that you wish to access.
- Add comments to your scripts. If you insert a semicolon at the end of a command line, anything to the right of the semicolon is ignored by AmigaDOS, but appears in the script as a comment. For example:

```
ASSIGN T: RAM:T ; set up directory for scripts
```

explains why you inserted the command.
- Test changes to your User-startup file. When experimenting with your User-startup file, reboot your Amiga after each change to test the result. However, be sure to wait until you are sure the file is completely written to disk before the reboot to avoid potential data loss.

Common Additions to the Startup Files

The following is a list of some of the most common additions that can be made to your startup files.

- Opening a Shell window
- Adding directories to the search path
- Adding logical device names to the assign list
- Making additional commands resident
- Starting input commodities and other programs
- Adding drive cache buffers

Using PIPE:

The PIPE: device transfers data from one program to another using temporary storage in RAM. Use a pipe: when you wish to apply output from one process as input to another process. Using a pipe to transfer data reduces the possibility of running out of memory when working with very large files and eliminates the need to create, save, and later delete a file needed only once.

The source and destination processes cannot be the same when using PIPE:. As you write information to the pipe, another process can read the data in First In First Out (FIFO) order. Providing a name after PIPE: gives the pipe a name.

Data sent to PIPE: is buffered in memory. When another application reads the pipe it gets the data in the order it came in. The pipe stays in RAM until its contents are emptied by the reading processes or until the next reboot.

PIPE: uses at least a 4 KB buffer per pipe-name. When the maximum buffer capacity is reached, PIPE: stops accepting data until another process reads data out of the buffer. If a process is reading data, it waits until more data arrives in the buffer. Buffer size can only be specified on the first reference to a particular channel.

The following are the options for PIPE:PIPE:options

channel_name	A unique channel name. Must begin with a non-numeric character. The channel_name is optional; using PIPE: alone is possible if not using multiple pipes.
buf_size	Size in bytes of the buffer to allocate. (The default size is 4 KB.) The buf_size is optional.
max_buffers	Maximum number of buffers allowed. Suspends the output channel if exceeded. Max_buffers = 0 indicates there should be no fixed limit on the number of buffers allocated.

PIPE: can be used from other programs, such as word processors or terminal programs. You can use any pipe-name. If the application reads the file sequentially, you can specify PIPE:<name> and it appears the same as an ordinary file to the application.

Information from one pipe can be copied to another. For example:

Shell window 1: COPY Hugefile PIPE:a
Shell window 2: COPY PIPE:a PIPE:b
Shell window 3: COPY PIPE:b PIPE:c
Shell window 4: COPY PIPE:c PIPE:d
Shell window 5: COPY PIPE:d PIPE:e
Shell window 6: TYPE PIPE:e ;Hugefile is TYPEd

The DOSDrivers mount file for PIPE: mount files:for PIPE:is as follows:

```
Handler      = L:Queue Handler
Priority      = 5
StackSize    = 3000
GlobVec      = -1
```

PIPE: is mounted by default during the standard Startup-sequence.

See Chapter 8 for additional PIPE: examples.

Glossary

absolute path

A path that gives the full information for locating a file, including the volume or device name, any subdirectories, and the file name.

aliases

Alternative names for AmigaDOS commands. An alias can be used to abbreviate frequently used commands or replace standard command names with different names.

archive

A backup copy of one or more files or a whole volume.

argument

A filename, option, or other kind of information passed along with the command name on a command line. Also known as a parameter.

assign

To identify a path to a directory or file under an additional name. This defines logical devices used by the operating system, such as C: where AmigaDOS stores command files, or S: where scripts are stored.

background process

A process that does not open its own window for input or output and does not take over the parent Shell.

block

1. A contiguous series of bytes on a storage device (a disk block normally equals 512 bytes)
2. A contiguous part of a script/program file; for example, an IF-ENDIF block.

Glossary-2

boot block

An area on a disk or PCMCIA card that contains boot code for the system to read when booting. The presence of a valid boot block on a disk or card makes it bootable.

Bridgeboard

A special hardware expansion board made by Commodore that makes your Amiga PC-compatible.

buffer

An area in RAM used for temporary data storage for disk or serial input/output or by some other operations.

cache memory

A storage area consisting of very fast RAM chips. It serves as a buffer between a fast CPU chip and a slower system memory. Built into systems with bigger processors.

Chip RAM

The part of the system RAM that the Amiga custom chips can access. All video and sound data is stored here.

circular link

A link that refers to a link that refers back to it.

Clipboard

An area in memory used to hold data in cut and paste operations.

command history

A list containing the most recently issued command lines. These command lines can be recalled, edited, and reissued.

command line interface (CLI)

See Shell.

comment

1. A line or part of a line that is not executed, but is included to document the operation of a program or script for future reference.
2. A short descriptive note that can be attached to any file using the FILENOTE command.

condition flag

A variable that indicates the condition on which a command ends.

console windows

Windows used by the Shell for text-oriented data input and output.

current directory

A Shell's current location in the directory structure, used as the default directory in which commands operate.

current window

The highlighted window that accepts input from the keyboard. A Shell window is current when it is opened.

cursor

The small rectangular highlighted box that indicates the position in the current file at which the next character will appear.

data cache

Hardware feature present on 68030 and 68040 microprocessors that substantially speeds memory access.

delimiter characters

Characters that define the beginning and ending of an argument string. For example, in the ED text editor, ", /, \, !, :, +, and % are valid delimiters.

detach

To separate a program from the process in which it was invoked so that the process' Shell window can close before the program is finished.

device drivers

Files that provide functions needed for a device to function properly on the system.

directory caching

A file system option used to increase directory listing speed.

directory link

See Link.

disk operating system

A part of the operating system that is devoted to managing disks and files.

Glossary-4

endless loop

See endless loop.

escape sequence

A string of control characters, normally introduced by the Escape character. You can control the window format and font style with escape sequences.

extended commands

In the ED text editor, commands consisting of one or two characters that can be grouped together, introduced by the Esc key.

extension

A sequence of characters beginning with a period, such as .info, added to the end of a filename to identify the type of file.

fail limit

The limit at which a return code value terminates a sequence of non-interactive commands.

Fast RAM

Part of the system RAM to which custom chips do not have access. Since only the CPU and some peripheral devices have access to Fast RAM, it is considerably faster to use.

fence characters

Used in MEMacs to find the beginning and end of a program nest structure. Fence characters can be parentheses, brackets, braces, or angle brackets.

field

1. An on screen area in which a variable value is displayed or entered.
2. The screen area behind the text of a Workbench icon label. The color of the field can be changed with the Font editor.

file system

A part of the operating system that defines how information is stored on storage devices. This includes file headers, data sectors, subdirectory headers, and bitmaps that indicate which sectors on a disk are already occupied and which are free.

global vector (GlobeVec)

A mount parameter needed by some devices.

hexadecimal

The base 16 numbering system.

hierarchical file system

A file system that allows directories to contain other directories, as well as files.

immediate commands

Commands in the ED text editor that are executed as soon as you press the associated key combination.

instruction cache

A type of memory in the 68020, 68030, and 68040 microprocessors that allows instructions to execute more quickly.

interactive listing mode

A mode of the DIR command that stops after each name in a directory listing to display a question mark at which commands controlling the listing can be entered.

internal commands

Commands that are built into the Shell. These do not need to be loaded from disk to use.

interprocess communication (IPC)

The mechanism by which two programs pass data to and from each other.

line windows

Subsections of the line on which the EDIT line editor executes all subsequent commands.

link

A file or directory that is a pointer to another file or directory on a disk. When an application or command calls the initial file or directory the linked file or directory is used. (Also called a hard link.)

logical device

An assigned directory or software device that is referred to by a device, but points to a directory or device handler.

MountList

A text file in the DEVS: directory that contains information about connected or logically defined devices. The MOUNT command uses this information to activate devices.

multitasking

The ability to run more than one program simultaneously. For example, on the Amiga you can start several programs from a single Shell window using the RUN command.

nesting

Multiple levels of IF statements within scripts or programs or multiple levels of subdirectories within directories.

non-detached programs

Commands that occupy the Shell while processing, preventing the Shell's use or closure until completion.

octal

The base 8 numbering system.

operating system

Underlying software that controls the computer's functions.

path

Information that provides the system with the location of a particular file. This can include a volume or drive name and subdirectory names. See also search path.

pattern matching

A way of searching files and directories that match a specified pattern; this can be done with wildcard characters.

PCMCIA

Personal Computer Memory Card International Association, an organization that establishes standards for memory card slot devices.

pipe

An Interprocess Communication (IPC) mechanism for sending the output of one command as input to another.

process

An Amiga operating system task that can communicate with AmigaDOS to access files. Each process is identified with a number that can be displayed using the STATUS command.

prompt

A special customizable text string that always appears at the start of a Shell window line to indicate that the system is ready to receive another command line.

protection bits

Attributes that indicate a file's type and the operations permitted on it. Use the LIST command to display the protection bits associated with a file.

pure

Describes a command or program that can be made resident. If a file is pure, the p protection bit is set.

redirection

The process of sending command input or output to a destination other than the Shell; for example, to a file.

re-entrant commands

Commands that support independent use by two or more programs simultaneously.

re-executable commands

Commands that do not have to be reloaded to be executed more than once.

resident

Commands stored in memory so that they do not have to be reloaded each time they are used. These commands should be pure.

return code

A number returned to the Shell from a command at completion to indicate its success or failure.

root

The main directory of a disk, containing all its files and/or subdirectories.

SCSI (Small Computer System Interface)

An interface standard for connecting peripheral devices to a computer system. (This is pronounced "scuzzy.")

search path

A series of directories in which AmigaDOS looks for commands that are entered without paths. The default search path includes the C: directory, the current directory, and several other directories specified in the standard Startup-sequence.

sector

The smallest storage unit on a disk, usually 512 bytes.

Shell

A text-oriented user interface in which you type in command lines and receive output in a special Shell or console window. Also called CLI (Command Line Interface).

soft link

A file or directory link that can span multiple devices or volumes. This is not currently supported on the Amiga.

spawn window

An AmigaDOS Shell window opened from MEMacs using the MEMacs New-Cli command. AmigaDOS commands entered into the spawn window do not interfere with MEMacs.

stack

An area of the computer's memory that is set aside by a program for intermediate storage. Allocating insufficient stack space can cause program failure.

Startup-sequence

A special script file that is automatically executed when the Amiga is booted.

string

A series of text characters treated as a unit, such as a message printed by ECHO. A string usually requires delimiter characters, such as spaces or quotation marks, to mark its beginning and end.

template

A string that defines a Shell command's arguments and argument types.

timestamp

The date and time associated with a file. This reflects when the file was created or the date and time that changes to the file were last saved.

User-startup

A special script file that you create to customize your system's startup. If a User-startup file exists, it is automatically executed by the Startup-sequence.

wildcard characters

Characters with special meaning when used in file name specifications. These characters are used in pattern matching operations.

Index

-
- .BRA, 5-7, 5-8
- .DEF, 5-7, 5-9
- .DOLLAR, 5-7, 5-10
- .DOLLAR keyword, 5-5
- .DOT, 5-7
- .info files, 3-5
- .KET, 5-7, 5-8
- .KEY, 5-7
 - arguments, 5-8
 - description, 5-7
 - position in file, 5-7

6

68040.library, B-14

A

ADDBUFFERS, 6-10
ADDDATATYPES, 6-92
adding cache buffers, 6-10
advanced features, D-1
ALIAS, 6-11
aliases, 6-11, 8-13, D-2

creating global alias, 6-11
displaying, 6-11
entering, 6-11
global, D-2
local, D-2
removing, 6-11, 6-87
setting, 6-11
AmigaDOS commands, 6-1
ADDBUFFERS, 6-10
ADDDATATYPES, 6-92
ALIAS, 6-11
alphabetic listing, 6-1
alternative names, 6-11
arguments, 3-12
ASK, 6-12
ASSIGN, 6-13
AVAIL, 6-17
BINDDRIVERS, 6-92
BREAK, 6-18
CD, 3-11, 6-19
CHANGETASKPRI, 6-21
command line, 3-13
command line length, 3-13
command name, 3-12
CONCLIP, 6-93
COPY, 3-11, 6-22
CPU, 6-24
DATE, 3-11, 6-26
DELETE, 3-11, 6-28
DIR, 3-11, 6-29
disk-based, 3-11
DISKCHANGE, 6-32
DISKCOPY, 3-11

Index-2

- ECHO, 6-32
- ED, 6-33
- EDIT, 6-34
- ELSE, 6-34
- ENDCLI, 6-35
- ENDIF, 6-35
- ENDSHELL, 3-11, 6-36
- ENDSKIP, 6-36
- entering, 2-3
 - similar commands, 2-6
- EVAL, 6-37
- EXECUTE, 6-38
- executing from within a string, 5-4
- FAILAT, 6-39
- FAULT, 6-40
- FILENOTE, 6-41
- FORMAT, 3-11
- general information, 3-12
- GET, 6-42
- GETENV, 6-43
- how to read, 6-5
 - angle brackets, 6-6
 - braces, 6-6
 - ellipsis, 6-7
 - indentation, 6-7
 - italics, 6-6
 - numeric value, 6-6
 - square brackets, 6-6
 - template, 6-8
 - vertical bar, 6-6
- ICONX, 6-43
- IF, 6-45
- INFO, 3-11, 6-46
- INSTALL, 6-47
- internal, 3-11
- IPREFS, 6-93
- JOIN, 6-48
- keywords, 3-7
- LAB, 6-49
- LIST, 3-12, 6-49
- list of basic commands, 3-11
- LOADRESOURCE, 6-52
- LOADWB, 6-53
- LOCK, 6-54
- MAGTAPE, 6-55
- MAKEDIR, 3-12, 6-56
- MAKELINK, 6-57
- making resident, C-1
- MOUNT, 6-57
- NEWCLI, 6-59
- NEWSHELL, 3-12, 6-59
- PATH, 6-62
- PROMPT, 6-64
- PROTECT, 6-65
- QUIT, 6-66
- RELABEL, 3-12, 6-67
- REMRAD, 6-68
- RENAME, 3-12, 6-68
- repeating, 2-6
- REQUESTCHOICE, 6-69
- REQUESTFILE, 6-70
- RESIDENT, 6-72
- RUN, 3-21, 6-74
- SEARCH, 6-76
- SET, 6-77
- SETCLOCK, 3-12, 6-78
- SETDATE, 6-79
- SETENV, 6-79
- SETFONT, 6-80
- SETKEYBOARD, 6-81
- SETPATCH, 6-94
- SKIP, 6-82
- SORT, 6-84
- STACK, 6-85
- STATUS, 6-85
- storage, 3-8
- template
 - displaying, 3-16
 - entering arguments, 3-16
- testing, 8-16
- TYPE, 3-12, 6-86
- types of, 3-11
- UNALIAS, 6-87
- UNSET, 6-87
- UNSETENV, 6-88
- VERSION, 6-88
- WAIT, 6-89

WHICH, 6-90
WHY, 6-91
Workbench-related, 7-1
 alphabetic listing, 7-1
AmigaDOS examples, 8-1
AmigaDOS file system, 3-1
 elements, 3-2
amigaguide.library, B-13
angle brackets, 3-18
 using in script files, 5-8
ARexx
 Stem Variables, 4-24
 support within ED, 4-24
arguments, 3-12
 argument passing, 3-21
 description, 3-13
 enclosing in quotation marks, 3-15
 keywords, 3-13
ASK, 5-6, 6-12
asl.library, B-13
ASSIGN, 6-13, C-2
 dismounting volumes/devices, 6-15
 late binding, 6-14
 non-binding, 6-15
 removing target names from list, 6-14
 using PATH option, C-2
assignments, 6-13, 8-9
 creating, 8-9
 listing, 6-14
 multiple-directory assignments, 6-16
asterisk, 3-19
AUTOPOINT, 7-13
Aux-Handler, B-10
AVAIL, 6-17

B

background processes, 6-74

 running, 6-74
 baud rates, setting, B-11
BINDDRIVERS, 6-92
bitmap fonts, B-12
BLANKER, 7-14
Boolean expressions
 evaluating, 6-37
boot block, 6-47
 checking for valid boot code, 6-47
 removing, 6-47
 writing or inspecting, 6-47
BREAK, 6-18
buffer allocation, 6-10
 recommended number, 6-10
 subtracting buffers, 6-10
bullet.library, B-13
burst mode, 6-25

C

C:, B-17
CALCULATOR, 7-18
CD, 3-11, 6-19
CDFileSystem, B-12
CHANGETASKPRI, 6-21
changing search path, 8-4
changing the mouse pointer, 7-8
changing Workbench screen colors, 7-7
Chip memory, 6-17
choosing languages, 7-6
Classes, B-17
 directories
 DataTypes, B-17
 Gadgets, B-17
CLICKTOFRONT, 7-15
Clipboard, 6-93
CLIPS:, B-16
CLOCK, 7-19
 options, 7-19
closing the Shell, 2-3

Index-4

- methods, 2-3
- CMD, 7-20
 - options, 7-21
- command conventions
 - Examples, 6-6
 - Format, 6-5
 - Path, 6-5
 - template, 6-5
- command history, 2-6
 - command line buffer, 2-7
 - searching for commands, 2-7
- command line, 2-5
 - adding comments, 5-4
 - argument passing, 3-21
 - concepts, 3-7
 - editing, 2-5
 - sample, 3-13
 - separating arguments, 3-19
- command line characters, 3-14
 - asterisk (*), 3-19
 - colon (:), 3-14
 - double quotation mark ("), 3-15
 - plus sign (+), 3-16
 - question mark (?), 3-16
 - slash (/), 3-15
- command name, 3-12
- commands, 3-8, 6-1, 7-4
 - ADDBUFFERS, 6-10
 - ADDDATATYPES, 6-92
 - ALIAS, 6-11
 - ASK, 6-12
 - ASSIGN, 6-13
 - AUTOPOINT, 7-13
 - AVAIL, 6-17
 - BINDDRIVERS, 6-92
 - BLANKER, 7-14
 - BREAK, 6-18
 - CALCULATOR, 7-18
 - CD, 6-19
 - CHANGTASKPRI, 6-21
 - CLICKTOFRONT, 7-15
 - CLOCK, 7-19
 - CMD, 7-20
 - CONCLIP, 6-93
 - COPY, 6-22
 - CPU, 6-24
 - CROSSDOS, 7-15
 - DATE, 6-26
 - DELETE, 6-28
 - description, 3-8
 - DIR, 6-29
 - DISKCHANGE, 6-32
 - DISKCOPY, 7-21
 - ECHO, 6-32, D-8
 - ED, 6-33
 - EDIT, 6-34
 - ELSE, 6-34
 - ENDCLI, 6-35
 - ENDIF, 6-35
 - ENDSHELL, 2-3, 6-36
 - ENDSKIP, 6-36
 - EVAL, 6-37
 - EXCHANGE, 7-16
 - EXECUTE, 6-38
 - FAILAT, 5-13, 6-39
 - FAULT, 6-40
 - FILENOTE, 6-41
 - FIXFONTS, 7-22
 - FKEY, 7-16
 - FONT, 7-5
 - FORMAT, 7-23
 - GET, 6-42
 - GETENV, 6-43
 - GRAPHICDUMP, 7-25
 - ICONEDIT, 7-26
 - ICONTROL, 7-5
 - ICONX, 6-43
 - IF, 6-45
 - INFO, 6-46
 - INITPRINTER, 7-26
 - INPUT, 7-6
 - INSTALL, 6-47
 - INTELLIFONT, 7-26
 - IPREFS, 6-93
 - JOIN, 6-48
 - KEYSHOW, 7-27
 - LAB, 6-49
 - LIST, 6-49

- LOADRESOURCE, 6-52
- LOADWB, 6-53
- LOCALE, 7-6
- LOCK, 6-54
- MAGTAPE, 6-55
- MAKEDIR, 6-56
- MAKELINK, 6-57
- MEmacs, 7-27
- MORE, 7-27
- MOUNT, 6-57
- MOUSEBLANKER, 7-17
- MULTIVIEW, 7-29
 - nesting, 5-10
- NEWCLI, 6-59
- NEWSHELL, 6-59
- NOCAPSLOCK, 7-17
- NOFASTMEM, 7-32
 - order of commands in script, D-8
- OVERSCAN, 7-7
- PALETTE, 7-7
- PATH, 6-62
- POINTER, 7-8
- PREPCARD, 7-32
- PRINTER, 7-8
- PRINTERGFX, 7-9
- PRINTERPS, 7-9
- PROMPT, 6-64
- PROTECT, 6-65
- QUIT, 6-66
- RELABEL, 6-67
- REMRAD, 6-68
- RENAME, 6-68
 - repeating, 5-12
- REQUESTCHOICE, 6-69
- REQUESTFILE, 6-70
- RESIDENT, 6-72
- RUN, 6-74
- SCREENMODE, 7-10
- SEARCH, 6-76
- SERIAL, 7-10
- SET, 6-77
- SETCLOCK, 6-78, D-8
- SETDATE, 6-79
- SETENV, 6-79
- SETFONT, 6-80
- SETKEYBOARD, 6-81
- SETPATCH, 6-94
- SKIP, 6-82
- SORT, 6-84
- SOUND, 7-11
- STACK, 6-85
- STATUS, 6-85
- TIME, 7-11
- TYPE, 6-86
- UNALIAS, 6-87
- UNSET, 6-87
- UNSETENV, 6-88
- VERSION, 6-88
- WAIT, 6-89
- WBPATTERN, 7-12
- WHICH, 6-90
- WHY, 6-91
 - comments, 5-4, 6-41
 - adding comments to command line, 5-4
 - adding with FILENOTE, 6-41
- Commodities programs, 7-12
 - arguments, 7-12
 - AUTOPOINT, 7-13
 - BLANKER, 7-14
 - CLICKTOFRONT, 7-15
 - CrossDOS, 7-15
 - EXCHANGE, 7-16
 - FKEY, 7-16
 - MOUSEBLANKER, 7-17
 - NOCAPSLOCK, 7-17
- commodities.library, B-14
- CON:, 3-4
- CONCLIP, 6-93
- condition flags, 5-13
- conditional statements, 6-34, 6-45
 - ELSE, 6-34
- console handler, B-10
- console window, 2-1, 3-4
 - current, 3-4
- CONSOLE:, 3-4, 3-19
- COPY, 3-11, 6-22

- defaults, 6-23
- copyback cache., 6-25
- copying within a Shell, 2-7
- CPU, 6-24
 - processor options list, 6-25
- CrossDOS, 7-15
 - CrossDOSFileSystem, B-11
 - DANSK.crossdos, B-12
 - FileSystem_Trans, B-12
 - INTL.crossdos, B-12
 - MAC.crossdos, B-12
- CrossDOSFileSystem, B-11
- Ctrl+C, 8-2
- Ctrl+D, 8-3
- current directory, 3-10
 - changing, 3-10, 8-3
 - changing with PCD, B-9
 - displaying, 6-19
 - properties, 3-10
 - referencing with quotation marks, 3-15
- cursor, 2-3
- customizing ED, 4-21
- customizing startup files, D-6
- customizing system startup, D-6

D

- data cache, 6-25
- data types, 6-92
- datatypes.library, B-14
- DATE, 3-11, 6-7, 6-26, 6-78
- debugging script files, 5-14
- default search path, 3-9
- DELETE, 3-11, 6-28
- devices, 3-3
 - clipboard.device, B-3
 - creating mount files, B-5
 - creating MountLists, B-5
 - description, 3-2
 - device handlers, B-10
 - device name, 3-3
 - floppy disks, 3-3
 - getting information, 6-46
 - hard disks, 3-3
 - loading device drivers, 6-92
 - logical, 3-4
 - mfm.device, B-3
 - mounting, 6-57, B-4
 - with mount files, B-4
 - with MountList, B-4
 - parallel.device, B-3
 - printer.device, B-3
 - Ram Disk, 3-3
 - RAM:, C-5
 - removing RAD:, 6-68
 - running device drivers, 6-92
 - serial device, B-3
 - setting write protection, 6-54
 - standard names, 3-4
 - CON:, 3-4
 - CONSOLE:, 3-4
 - DF0:, 3-4
 - NIL:, 3-4
 - PAR:, 3-4
 - PRT:, 3-4
 - RAD:, 3-4
 - RAM:, 3-4
 - SER:, 3-4
 - SYS:, 3-4
 - version number, 6-88
- DEVS:, B-3
 - device files, B-3
 - other files, B-4
 - postscript_init.ps, B-4
 - system configuration, B-4
- DF0:, 3-4
- DIR, 3-11, 6-12, 6-29, 8-18
 - interactive mode, 8-18
 - options, 6-30
- directories, 3-5, B-1
 - C:, B-17
 - CD, 6-19
 - Classes, B-17
 - CLIPS:, B-16
 - COPY, 6-22

- copying, 6-22, 8-7
- creating, 6-56
- current directory, 3-10
- deleting, 6-28
- description, 3-2
- DEVS:, B-3
- displaying contents, 8-4
- displaying sorted list, 6-29
- ENV:, B-16
- ENVARC:, B-16
- FONTs:, B-12
- L:, B-10
- LIBS:, B-13
- listing information, 6-49
- LOCALE:, B-15
- moving, 6-68
- naming conventions, 3-6
- nesting, 3-5
- renaming, 6-68
- REXX:, B-15
- root directory, 3-5
- S:, B-8
- search path, 3-9
- subdirectories, 3-5
- T:, B-16
- directory caching, 7-23
- DISKCHANGE, 6-32
- DISKCOPY, 3-11, 7-21
- diskfont.library, B-14
- display system date, 6-26
- displaying cache buffers, 6-10
- dollar sign, 5-4
- dot commands, 5-6
 - .<space>, 5-7
 - .BRA, 5-7
 - .DEF, 5-7
 - .DOLLAR, 5-7
 - .DOT, 5-7
 - .KET, 5-7
 - .KEY, 5-7
 - list, 5-10
 - substituting parameters, 5-8
- double dollar sign, 5-5

E

- ECHO, 5-6, 5-14, 6-32, D-8
 - debugging with SET ECHO ON, 5-14
- ED text editor, 4-1, 6-33
 - accessing expanded menus, 8-10
 - ARexx support, 4-24
 - changing case, 4-7
 - Ctrl-F, 4-7
 - Command menu, 4-18
 - cursor control, 4-13
 - customizing, 4-21
 - special key mappings, 4-16
 - deleting text, 4-6
 - Backspace, 4-6
 - Ctrl-O, 4-6
 - Ctrl-Y, 4-6
 - Del, 4-6
 - Edit menu, 4-12
 - extended commands, 4-7
 - cursor control, 4-13
 - editing commands, 4-12
 - environment setting commands, 4-15
 - file manipulation commands, 4-18
 - program control commands, 4-11
 - searching and exchanging, 4-13
 - extended mode
 - entering, 4-7
 - format, 4-2
 - grouping commands together, 4-7
 - immediate commands, 4-4
 - changing case, 4-7
 - deleting text, 4-6
 - inserting text, 4-6

- moving the cursor, 4-4
 - specifying, 4-4
 - inserting lines, 4-6
 - inserting text, 4-6
 - invoking a requester, 4-8
 - maximum characters on a line, 4-6
 - menus
 - enabling expanded menu, 4-9
 - miscellaneous commands, 4-18
 - Movement menu, 4-13
 - moving the cursor, 4-4
 - printing from, 4-23
 - Project menu, 4-11
 - quitting, 4-24
 - repeating extended commands, 4-20
 - scrolling through file, 4-5
 - Search menu, 4-13
 - Settings menu, 4-15
 - starting, 4-3
 - status line, 4-2
 - using delimiters, 4-8
 - using ED, 4-4
 - immediate commands, 4-4
 - ED text editors
 - menus, 4-8
 - ED-Startup File, B-8
 - EDIT, 6-34
 - EDIT text editor, 4-43
 - commands, 4-45
 - changing command, 4-53
 - changing input files, 4-53
 - changing output files, 4-53
 - editing current line, 4-47
 - editing line windows, 4-48
 - ending EDIT, 4-56
 - entering, 4-45
 - inserting and deleting lines, 4-47
 - inspecting the source file, 4-51
 - joining lines, 4-50
 - making global changes, 4-52
 - moving through file, 4-46
 - renumbering lines, 4-50
 - selecting the Current Line, 4-46
 - splitting lines, 4-50
 - verifying lines, 4-51
 - Starting, 4-44
 - ELSE, 5-6, 6-34
 - ENDCLI, 6-35
 - ENDIF, 5-6, 6-35
 - ENDSHELL, 2-3, 3-11, 6-36
 - ENDSKIP, 5-6, 6-36
 - ENV:, B-16
 - ENVARC:, B-16
 - environment variables, 5-14
 - calling from scripts, 6-80
 - creating, 6-77
 - creating global, 6-80
 - creating with SET, 5-16
 - creating with SETENV, 5-16
 - ECHO, 5-15
 - getting the value of global variables, 6-43
 - getting the value of local variables, 6-42
 - global, 5-16
 - in calculations, 5-15
 - local, 5-16
 - PROCESS, 5-15
 - RC, 5-15
 - RESULT2, 5-15
 - using, 5-14
- error messages, 6-40, A-1
 - WHY command, 6-91
- escape sequences, D-3
 - standard (table), D-4
 - with asterisk, D-6
- EVAL, 5-15, 6-37
 - creating loops, 8-23
 - LFORMAT option, 6-37
 - operations table, 6-37
- EXCHANGE, 7-16
- EXECUTE, 5-6, 6-38

- starting ED, 4-3
- executing commands from within a string, 5-4
- external cache, 6-26

F

- fail limit, 6-39
- FAILAT, 5-6, 5-13, 6-39
- Fast File System, 7-23
- Fast memory, 6-17
- FASTROM, 6-25
- FAULT, 6-40
- file management, 3-2
- FILENOTE, 6-41
 - creating a comment, 6-41
- files, 3-5, 3-8
 - .info files, 3-5
 - COPY, 6-22
 - copying, 8-7
 - creating links, 6-57
 - deleting, 6-28
 - deleting files with icons, 8-16
 - deleting from Workbench disk, C-3
 - description, 3-2
 - displaying text files, 6-86
 - joining, 6-48
 - listing information, 6-49
 - MEmacs file size, 4-27
 - moving, 6-68, 8-17
 - naming conventions, 3-6
 - protection bits, 6-65
 - renaming, 6-68
 - sorting lines alphabetically, 6-84
- FileSystem_Trans, B-12
- FixFonts, 7-22, B-12
- FKEY, 7-16
- floppy disk systems, C-1
 - advantages, C-6
 - ASSIGNing PATH, 6-15

- deleting files from Workbench disk, C-3
 - files not to delete, C-4
 - files you can delete, C-4
- making commands resident, C-1
- preloading resources, 6-52, C-2
- using Ram Disk, C-5
- using RESIDENT, 6-72

- floppy disks, 3-3

- FONT, 7-5

- fonts, B-12

- adding fonts, B-12
- bitmap fonts, B-12
- changing in current Shell, 6-80
- flushing unused, 8-22
- INTELLIFONT, 7-26
- outline fonts, B-13
- SETFONT options, 6-81
- specifying, 7-5
- updating .fonts files, 7-22
- using FixFonts, B-12

- FORMAT, 3-11, 7-23

- options, 7-23

- formatting disks, 7-23

- freeing memory, 6-18, 8-22

G

- GET, 5-16, 6-42

- GETENV, 5-16, 6-43

- global variables, 5-16

- getting value, 6-43

- removing, 6-88

- setting, 6-79

- GRAPHICDUMP, 7-25

H

hard disks, 3-3
hardware clock, 6-78
 reading, 6-78
 setting, 6-78

I

ICONEDIT, 7-26
icons, 8-11
 .info files, 3-6
 copying, 3-6
 attaching to file or directory, 8-11
ICONTROL, 7-5
ICONX, 6-43, 8-20
 using, 8-20
IF, 5-6, 6-45
 keywords, 6-45
IF block, 6-34
 ELSE, 6-34
 ENDIF, 6-35
iffparse.library, B-14
implied CD, 6-20
INFO, 3-11, 6-46
information storage, 3-1
INITPRINTER, 7-26
INPUT, 7-6
INSTALL, 6-47
instruction cache, 6-25
integer expressions
 evaluating, 6-37
INTELLIFONT, 7-26
international file system, 7-23
IPREFS, 6-93

J

JOIN, 6-48, 8-22

K

key combinations, 2-5
keymaps, 6-81
 available, 6-81
 setting, 6-81
KEYSHOW, 7-27
keywords, 3-7, 3-13
 .KEY, 5-7
 dot commands, 5-6
 IF command, 6-45
 optional, 3-13

L

L:, B-10
LAB, 5-6, 6-49
late binding assign, 6-14
LFORMAT, 5-3, 6-37, 6-50
 generating scripts, 8-19
LIBS:, B-13
links, 6-57
 directory links, 6-57
 hard links, 6-57
LIST, 3-12, 5-3, 6-49, 8-19
 generating scripts with
 LFORMAT, 5-3
 LFORMAT
 customizing output, 8-20
 description, 6-50
 generating scripts, 8-19

- options, 6-49
- LOADRESOURCE, 6-52, C-2
- LOADWB, 6-53
- local variables, 5-16
 - getting value, 6-42
 - removing, 6-87
 - setting, 6-77
- LOCALE command, 7-6
- locale.library, B-14
- LOCALE:, B-15
 - directories, B-15
 - Catalogs, B-15
 - Countries, B-15
 - Help, B-15
 - Languages, B-15
- LOCK, 6-54
- logical devices, 3-4
 - controlling assignment, 6-13
- lowlevel.library, B-14

M

- MAGTAPE, 6-55
- MAKEDIR, 3-12, 6-56
- MAKELINK, 6-57
- making commands resident, C-1
- making room on Workbench disk, C-3
- math functions, B-13
- mathieeedoubbas.library, B-14
- mathieeedoubtrans.library, B-14
- mathieeesingtrans.library, B-14
- mathtrans.library, B-14
- MEmacs text editor, 4-27, 7-27
 - command mode, 4-28
 - customizing, 4-42
 - file size, 4-27
 - menu commands, 4-30
 - Edit, 4-33
 - Extras, 4-39
 - Line, 4-36
 - Move, 4-35
 - Project, 4-31
 - Search, 4-38
 - Set-Mark, 4-29
 - Window, 4-35
 - Word, 4-37
 - non-menu commands, 4-41
 - normal mode, 4-28
 - quitting, 4-43
 - special terms, 4-29
 - Buffer, 4-29
 - Dot, 4-29
 - Kill, 4-29
 - Mark, 4-29
 - Modified Buffers, 4-30
 - Window, 4-29
 - starting, 4-27
 - using, 4-28
- memory availability, 6-17
 - freeing memory, 6-18
- MMU, 6-25
- MORE, 5-16, 7-27
- MOUNT, 6-57, B-4
- mount files, 6-57, B-4
 - creating, B-5
 - default values, B-5
 - device handlers, B-10
 - for PIPE:, D-10
- MountList, 6-57, B-4
 - creating, B-5
 - default values, B-5
 - device handlers, B-10
- MOUSEBLANKER, 7-17
- moving files, 8-17
- MS-DOS formatted disks, 7-15
 - reading and writing, 7-15
- MultiView, 7-29
 - options, 7-29
 - supported ARexx commands, 7-30

N

- naming files and directories, 3-6
 - allowable characters, 3-6
 - case-sensitivity, 3-6
 - name length, 3-6
 - reserved characters, 3-6
 - using spaces, 3-6
- nested directories, 3-5
- nesting commands, 5-10
- NEWCLI, 6-59
- NEWSHELL, 3-12, 6-59, 8-2
 - window options, 6-60
- NIL:, 3-4
- NOCAPSLOCK, 7-17
- NOFASTMEM, 7-32
- non-binding assign, 6-15

O

- Old File System, 7-23
- opening a Shell window from Workbench, 8-1
- outline fonts, B-13
 - INTELLIFONT, 7-26
- OVERSCAN, 7-7

P

- PALETTE, 7-7
- PAR:, 3-4
- parallel port, 3-4
- parameter substitution, 5-6, 6-39
- partitions
 - description, 3-2
 - getting information about, 6-46

- setting write protection, 6-54
- pasting within a Shell, 2-7
- PATH, 6-62
- paths
 - changing search path, 8-4
 - description, 3-2
 - search path, 3-9
 - relative path, 8-2
 - searching for specific items, 6-90
 - specifying, 3-14
- pattern matching, 3-16, 8-8
 - DPat, B-9
 - examples, 3-17
 - SPat, B-9
 - used with CD, 6-20
 - using scripts for multiple, 5-3
- PCD, B-9
- PCMCIA memory cards, 7-32
 - preparing, 7-32
- Pipe-Handler, 7-28
- PIPE:, 8-25, B-11
 - buffer size, D-9
 - description, D-9
 - mount file, D-10
 - naming, D-9
 - options, D-9
 - source and destination processes, D-9
 - using, 8-25, D-9
- POINTER, 7-8
- Port-Handler, B-11
 - setting baud rate, B-11
- PostScript printers, 7-9
 - controlling features, 7-9
- postscript_init.ps, B-4
- Preferences editors, 7-4
 - arguments, 7-4
 - FONT, 7-5
 - ICONTROL, 7-5
 - INPUT, 7-6
 - IPREFS, 6-93
 - LOCALE, 7-6
 - OVERSCAN, 7-7

- PALETTE, 7-7
- POINTER, 7-8
- PRINTER, 7-8
- PRINTERGFX, 7-9
- PRINTERPS, 7-9
- SCREENMODE, 7-10
- SERIAL, 7-10
- SOUND, 7-11
- TIME, 7-11
- WBPATTERN, 7-12
- preloading resources, C-2
 - using LOADRESOURCE, C-2
- PREPCARD, 7-32
 - options, 7-32
- PRINTER, 7-8
 - printer, 3-4
- PRINTERGFX, 7-9
- PRINTERPS, 7-9
- process numbers, 3-21
 - displaying, 6-18
 - referencing current Shell, 5-5
 - setting attention flags, 6-18
 - substituting current, 5-5
- processes
 - changing priority, 6-21
 - end Shell, 6-35, 6-36
 - executing in background, 6-74
 - listing information about, 6-85
 - stopping, 6-19
- processor options, 6-24
 - data cache, 6-25
 - instruction cache, 6-25
 - list, 6-25
- programs
 - commands, 3-8
 - description, 3-8
 - names, 3-12
 - running from within the Shell, 3-20
 - scripts, 3-8
 - stopping, 8-2
 - Ctrl+C, 8-2
 - storage, 3-8
- PROMPT, 6-64

- PROTECT, 6-65
 - protection bits, 6-65
 - displaying, 6-65
 - list, 6-65
- PRT:, 3-4
- public screens, 8-14
 - internal names, D-2
 - opening a Shell window on, 8-14
- PUBSCREEN, D-1
- pure commands, 6-72

Q

- Queue-Handler, B-11
- QUIT, 5-6, 6-66

R

- RAD:, 3-4, C-6
- Ram Disk, 3-3, 3-4, C-5
 - custom icon, 8-15
 - recoverable, C-6
 - size, C-5
- RAM:, 3-4
- RAM:Clipboards, B-16
- re-entrant commands, 6-72
- re-executable commands, 6-72
- realtime.library, B-14
- recoverable RAM disk, 6-68
 - removing, 6-68
- redirection, 3-18
 - angle brackets, 3-18
 - asterisk, 3-20
 - double right angle brackets, 3-19
 - redirecting input, 3-18
 - left angle bracket, 3-19
 - redirecting output, 3-18

- right angle bracket, 3-18
- redirecting printer output, 7-20
- using question mark character, 5-5
- RELABEL, 3-12, 6-67
- relative path, 8-2
- REMRAD, 6-68
- RENAME, 3-12, 6-68
- repeating commands, 2-6, 5-12
- REQUESTCHOICE, 5-6, 6-69
- REQUESTFILE, 5-6, 6-70
- RESIDENT, 6-72
- resident commands, C-1
 - list of commands that cannot be made resident, 6-72
- return codes, 5-13, 6-39
- revision numbers, 6-88
- REXX:, B-15
- rexsupport.library, B-14
- rexsyslib.library, B-14
- ROM patches, 6-94
- root directory, 3-2
- RUN, 3-21, 6-74
- running programs, 3-20

S

- S:, B-8
- SCREENMODE, 7-10
- scripting characters, 5-3
 - back apostrophe (`), 5-4
 - dollar sign, 5-4
 - double dollar sign, 5-5
 - question mark, 5-5
 - semicolon (;), 5-4
- scripts, 3-8, 5-1
 - .BRA, 5-8
 - .KET, 5-8
 - Adding Comments, D-8
 - ARexx programs, 3-9
 - ASK command, 6-12
 - calling environment variables, 6-80
 - commands, 5-5
 - ASK, 5-6
 - ECHO, 5-6
 - ELSE, 5-6
 - ENDIF, 5-6
 - ENDSKIP, 5-6
 - EXECUTE, 5-6
 - FAILAT, 5-6
 - IF, 5-6
 - LAB, 5-6
 - QUIT, 5-6
 - REQUESTCHOICE, 5-6
 - REQUESTFILE, 5-6
 - SKIP, 5-6
 - WAIT, 5-6
 - conditional statements, 6-45
 - creating, 5-1
 - creating Move command, 8-17
 - creating requesters for, 5-11
 - debugging, 5-14
 - using SET ECHO ON, 5-14
 - description, 3-8
 - displaying requesters, 6-69
 - dot commands, 5-6
 - list, 5-10
 - DPat, B-9
 - ending, 5-12
 - entering comments, 5-10
 - environment variables
 - creating with SET, 5-16
 - creating with SETENV, 5-16
 - global, 5-16
 - local, 5-16
 - using GET, 5-16
 - using GETENV, 5-16
 - using UNSET, 5-16
 - using UNSETENV, 5-16
 - error messages, 5-14
 - executing through icon, 6-43
 - executing with argument substitution, 6-38
 - exiting, 6-66

- generating automatically, 5-3
 - LFORMAT, 5-3
 - halting, 5-11
 - interactive, 5-11
 - order of commands within, D-8
 - parameter substitution, 5-6, 6-39
 - .DEF, 5-9
 - .DOLLAR, 5-10
 - specifying default strings, 5-9
 - passing variables to, 6-39
 - PCD, B-9
 - preventing output, 8-21
 - repeating command, 5-12
 - skip to label, 6-82
 - SPat, B-9
 - specifying labels, 6-49
 - stopping, 5-13, 8-2
 - Ctrl+D, 8-3
 - types, 5-2
 - AmigaDOS, 5-2
 - ARexx, 5-2
 - simple, 5-3
 - understanding, 5-1
 - User-startup file, 8-8
 - using environment variables, 5-14
 - using EXECUTE command, 5-2
 - using file requester, 6-70
 - using MORE, 5-16
 - writing interactive scripts, 6-12
- SCSI tapes, 6-55
- SEARCH, 6-76
 - options, 6-76
- search path, 3-9
 - adding directory names, 6-63
 - C:, B-17
 - changing, 6-62, 8-4
 - default, 3-9
 - displaying directory names in current, 6-62
 - extending, 3-23
 - PATH command, 6-62
 - PCD, B-9
 - removing a directory name, 6-63
 - replacing, 6-63
 - running programs, 8-2
 - running programs not on, 8-2
 - WHICH, 6-90
- selecting a display mode, 7-10
- SER:, 3-4
 - setting baud rate, B-11
- SERIAL, 7-10
- serial port, 3-4
- SET, 5-16, 6-77
- set system date, 6-26
- SETCLOCK, 3-12, 6-78, D-8
- SETDATE, 6-79
- SETENV, 5-16, 6-79
- SETFONT, 6-80
 - options, 6-81
- SETKEYBOARD, 6-81
 - available keymaps list, 6-81
- SETPATCH, 6-94
- setting the system clock, 7-11
- Shell, 2-1
 - copying within, 2-7
 - customizing environment, D-2
 - description, 2-1
 - editing within the Shell, 2-5
 - helpful hints, 2-9
 - listing output, 2-6
 - maximum command line length, 3-13
 - pasting within, 2-7
 - running programs, 3-20, 8-2
 - starting ED, 4-3
 - using, 2-3
- Shell prompt, 2-3
 - changing, D-3
 - changing the prompt string, 6-64
 - customizing, 3-22
 - default prompt string, 6-64
 - modifying, 8-14
 - substitution strings, 6-64

- Shell window, 2-1
 - asterisk, 3-20
 - closing, 2-3
 - CONSOLE:, 3-20
 - controlling with WINDOW, 8-13
 - current window, 2-3
 - customizing, 6-59, D-1
 - description, 2-1
 - ENDSHELL, 6-36
 - enlarging, 2-4
 - fonts, 2-1
 - multiple windows, 2-2
 - opening, 2-2, 6-59
 - opening additional Shells, 8-2
 - opening from Workbench, 8-1
 - opening with NEWSHELL, 2-2, 6-59
 - options, 6-60
 - using icons with, 2-1
 - using mouse with, 2-1, 2-8
 - working with a single Shell, 8-10
- Shell-startup file, 6-61, D-2
 - changing the prompt, D-3
 - creating global alias, 6-11
 - escape sequences, using, D-3
 - using aliases, D-2
- SKIP, 5-6, 6-82
- SKIP block, 6-36
 - terminating, 6-36
- SORT, 6-84, 8-22
- SOUND, 7-11
- SPat, B-9
- special AmigaDOS characters, 3-14
- specifying fonts, 7-5
- specifying paths, 3-14
- specifying printer/print options, 7-8
- specifying Workbench parameters, 7-5
- STACK, 6-85
- stack size
 - displaying, 6-85
 - range, 6-85
 - setting, 6-85
- starting Workbench, 6-53

- Startup-sequence script file, D-6
- STATUS, 6-85
- strings, 6-32
 - displaying, 6-32
 - echoing a substring, 6-32
 - searching, 6-76
- subdirectories, 3-5
 - description, 3-2
- substitution operators, 6-51
- SYS:, 3-4
- system commands, 6-92
 - ADDDATATYPES, 6-92
 - BINDDRIVERS, 6-92
 - CONCLIP, 6-93
 - IPREFS, 6-93
 - SETPATCH, 6-94
- system-configuration, B-4

T

- T:, B-16
- template, 6-5
 - description, 6-8
 - displaying, 6-8
 - notation, 6-8
- testing commands, 8-16
- text editing keys, 2-5
- text editors, 4-1
 - ED, 4-1
 - EDIT, 4-43
 - MEmacs, 4-27
- TIME, 7-11
- timestamp
 - changing, 6-79
- TRAP, 6-25
- TYPE, 3-12, 6-86
- types of commands, 3-11

U

- UNALIAS, 6-87
- UNSET, 5-16, 6-87
- UNSETENV, 5-16, 6-88
- User-startup file, 8-8, D-6
 - common additions, D-8
 - creating, 8-8
 - customizing system startup, D-6
 - editing, D-8
 - using PATH option, C-3

V

- VERSION, 6-88
- version.library, B-14
- versions
 - testing, 8-22
- volumes
 - description, 3-2
 - SYS:, 3-4
 - volume name, 3-3

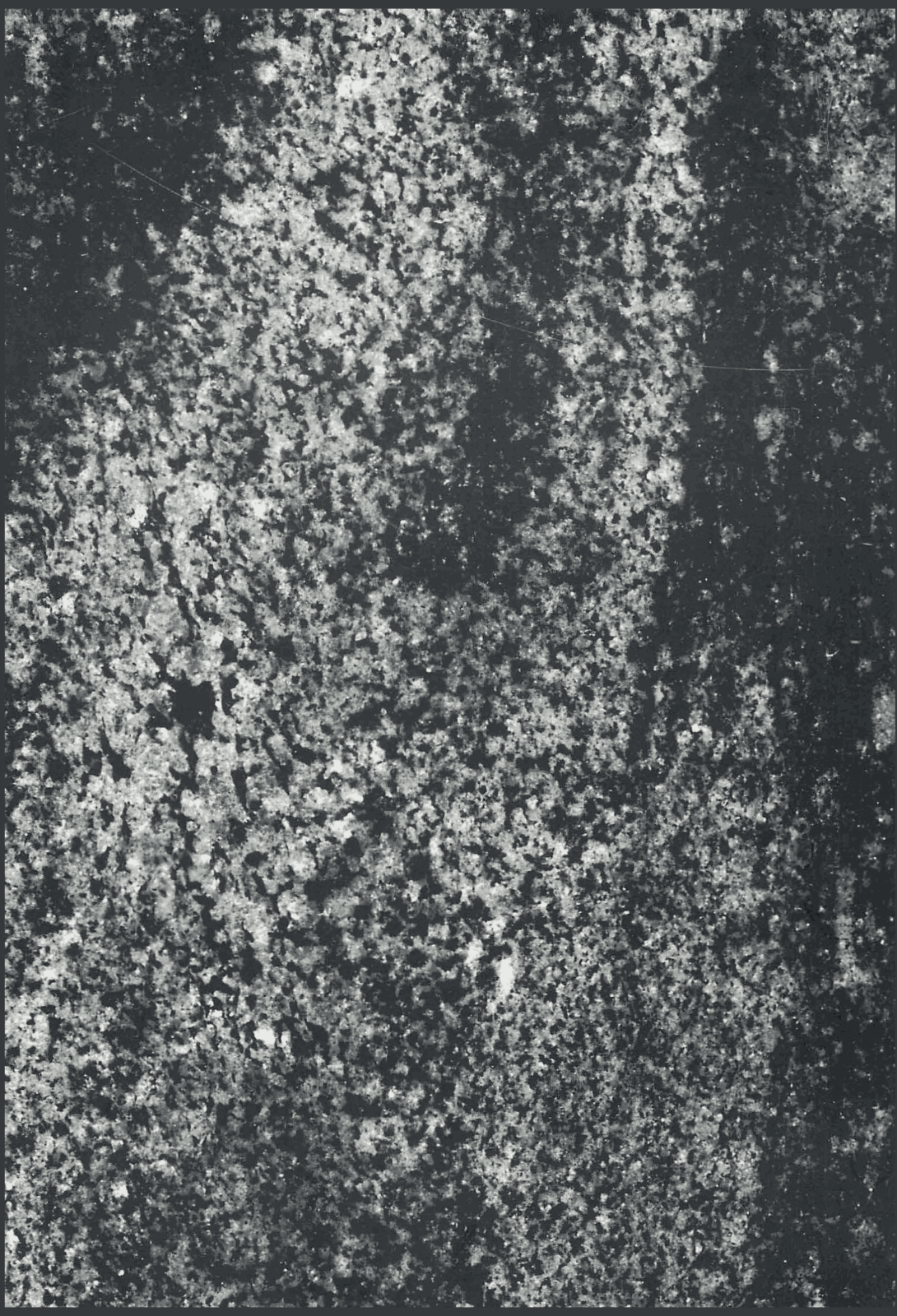
W

- WAIT, 5-6, 6-89
- WBPATTERN, 7-12
- WHICH, 6-90
- WHY, 6-91
- wildcard characters, 3-16
 - list, 3-16
 - using for multiple deletes, 6-28
- Workbench-related commands, 7-1
 - AUTOPOINT, 7-13
 - BLANKER, 7-14
 - CALCULATOR, 7-18

- CLICKTOFRONT, 7-15
- CLOCK, 7-19
- CMD, 7-20
- Commodities programs, 7-12
- CROSSDOS, 7-15
- DISKCOPY, 7-21
- EXCHANGE, 7-16
- FIXFONTS, 7-22
- FKEY, 7-16
- FONT, 7-5
- FORMAT, 7-23
- GRAPHICDUMP, 7-25
- ICONEDIT, 7-26
- ICONTROL, 7-5
- INITPRINTER, 7-26
- INPUT, 7-6
- INTELLIFONT, 7-26
- KEYSHOW, 7-27
- LOCALE, 7-6
- MEmacs, 7-27
- MORE, 7-27
- MOUSEBLANKER, 7-17
- MULTIVIEW, 7-29
- NOCAPSLOCK, 7-17
- NOFASTMEM, 7-32
- OVERSCAN, 7-7
- PALETTE, 7-7
- POINTER, 7-8
- Preferences editors, 7-4
- PREPCARD, 7-32
- PRINTER, 7-8
- PRINTERGFX, 7-9
- PRINTERPS, 7-9
- SCREENMODE, 7-10
- SERIAL, 7-10
- SOUND, 7-11
- TIME, 7-11
- WBPATTERN, 7-12
- write protecting devices and partitions, 6-54

Notice

Notice





**This was brought to you
from the archives of**

<http://retro-commodore.eu>