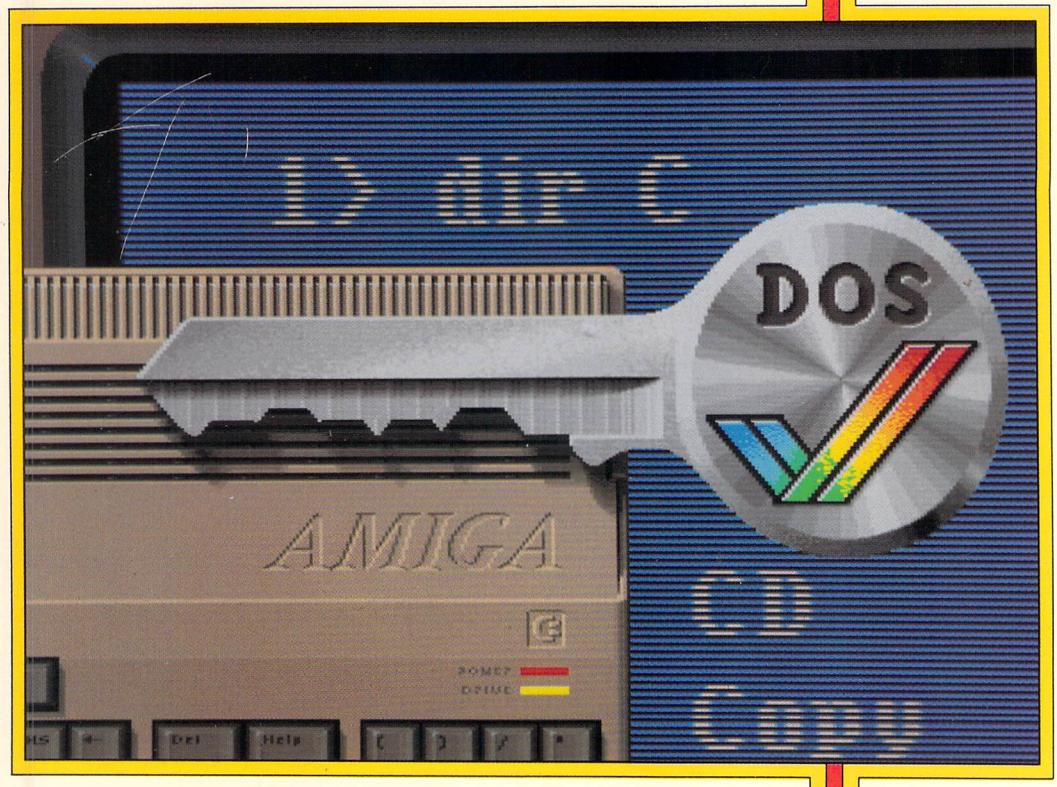
The Amiga™ Companion

A complete, up-to-date guide to using AmigaDOS and the CLI.

By Rob Peck



An IDGC/Peterborough Publication

The Amiga™ Companion

The Amiga™ Companion

Robert A. Peck

IDGC/Peterborough
Peterborough, NH

ISBN 0-928579-00-X

Cover Art by Jim Sachs

First Printing: September, 1988
Printed in the United States of America

TO MY DARLING ANDREA

No matter how long we are together, the days will always seem as bright to me as that marvelous day when first we met.

Acknowledgements

I received many valuable comments on the early drafts of the manuscript from Harv Laser, Glenn Agans, Dan and Marilyn Bonachea at the Amiga Zone of the American People Link network. They helped me to take a step out of "developer shoes" and back into "user shoes." Their critique, I believe, has made this a better book.

I also wish to thank the people on both PeopleLink and CompuServe who contributed startup sequences and answers to the question: "What should I tell new users?" These include Larry Phillips, Steve Ahlstrom, Nick Sullivan and others too numerous to mention.

To the staff at *AmigaWorld* magazine, including Bob Ryan, Dan Sullivan, Linda Barrett, Shawn Laflamme, Barbara Gefvert, Eileen Terrill, Guy Wright, Ken Sutcliffe, Howard Happ, Debbie Davies and Anne Dillon. Thanks for the effort that you have put into this book. Sometimes we writers are the lucky ones—we just have to figure out what the reader wants to KNOW. Then the editors work to make sure the material really is something that people will want to READ. Nice job, folks.

And last, but not least, thanks to all of the folks who have contributed their time and effort to produce all of the wondrous public-domain (and commercial) software for the Amiga. And to Commodore, for continuing the Amiga development effort.

Amiga lives! And we like it!

Contents

Preface

In July of 1985 Commodore-Amiga introduced its eagerly awaited new machine—the Amiga 1000—amid tremendous excitement and high expectations. Here was a machine with a new kind of hardware, offering high-performance graphics and sound at a reasonable price. Today, with the introduction of the Amiga 500, this technology has come down to a price that many more people can afford. And the Amiga 2000, with its expansion slots and IBM-PC-compatibility options, provides a base upon which the business and scientific communities can build.

If you have any doubts at all about the benefits of the Amiga's custom hardware, just perform the following test:

1. Open a drawer on the Amiga's Workbench screen and make its window about two-thirds the size of the screen.
2. Open another drawer and make its window just about the same size as the first one, positioned so that the Front/Back gadgets are lined up over those of the window that is now in the back.
3. Click on the Back Gadget repeatedly and rapidly. Watch how quickly the back window pops to the front. It seems to flash into existence, rather than being painted one line at a time.

That shows the Amiga's special hardware in action. Try window-popping like this on some other machines and you will notice a difference in speed. If you do find a faster machine, it's probably a lot more expensive than the Amiga.

Even today, three years after it was introduced, the Amiga remains a very advanced piece of hardware. And much of that initial excitement about the Amiga is still alive, even for those of us who were part of the Amiga's original development team. Although most of us no longer work directly for Commodore, almost all of the original team still retain close ties with the Amiga community—through local user groups, the Amiga developers' community, bulletin boards, and so forth. Some of the crew is developing software for the machine, and I continue to look for and develop tools for the Amiga. We just can't get away from it, and don't want to.

About This Book

When I first read the manuals Commodore includes with the Amiga, I was dismayed by the lack of information about AmigaDOS and the CLI. I thought the new user would need more information than these introductory manuals provide. On examining *The AmigaDOS Manual* (Bantam Books, 1987), the "official" source of AmigaDOS information, I thought that it had too few examples to show new users what the CLI is all about. The same held true for other books I read on AmigaDOS.

I've tried a different approach here. Instead of one command per page, this book is organized by tasks. Just ask yourself, "What do I want to do?" and you will probably find a section that covers that topic. The commands that perform a particular kind of job are all in one place and are explained with a series of tutorial examples.

The book is intended to help all Amiga owners—whether you have a 500, 1000, or 2000—and it includes tips on using the CLI even if you have only a single-drive system. I have tried to ensure that readers at all levels of expertise will learn something new about AmigaDOS. If you are a beginner, you may feel that certain sections contain "excess" information. As you learn more about your computer and its operating system, you will begin to appreciate some of the more esoteric com-

mands and options available. That is just part of the familiarization process.

This book contains three major sections:

The Command Line Interface (Chapters 1–10)

Your *Introduction to the Amiga* manual shows you how to use the Workbench. I assume that you already have a general working knowledge of the Workbench interface. If you are not familiar with the Workbench, you may find it necessary to review the introductory manuals.

This section tells you how to use the "other" user interface of the Amiga—the CLI.

Low-Cost Accessory Software (Chapter 11)

This section discusses several public-domain (also called "PD") programs, as well as some low-cost and shareware programs that I have found to be either useful or indeed essential to the effective use of the Amiga. It also suggests certain tools and demonstration programs that you may want to have to show off the power of your machine.

My intention in this section is to show you a way to save money on building a personal library of programs. As of this writing, there are at least 128 disks in the largest freely redistributable (PD) library for the Amiga. These disks, each containing several programs, demos, and utilities, were created by a fellow named Fred Fish and are available from him or from your local computer store. Add to this the wide variety of user-group disks and other low-cost and shareware programs in existence, and you have quite a large selection to choose from. You need to know where and how to find the "neat stuff," and that is what I have tried to provide you with here. Sure, my taste in programs may be different from yours, but I explain why I (or my friends) found a particular program useful.

Questions and Answers (Chapter 12)

I have scoured computer bulletin boards to come up with a series of questions that new users often ask, and have provided an answer for each. In future editions of this book, I would like to expand this section

(as well as the PD Software section). I can do this, however, only if I get help from you. If you're a new user with questions that I have not already answered here, please send me your questions: I'll collect questions for a few weeks, then see that the answers to the most frequently asked questions become part of the next update to this book. If you would like to obtain a copy of the most recent update to the questions chapter as soon as it becomes available, please send a self-addressed, stamped, business-sized envelope, along with a check or money order (no cash please) for $2.00 (to cover the cost of duplication and handling) to the address below. Also, if you are a more experienced user who would like to share the solution to a problem with the rest of us, I would enjoy hearing from you as well. The address is:

Rob Peck
DATAPATH
PO Box 1828
Los Gatos, CA 95031-1828

At the end of the book, I have included a few appendices that cover additional topics that may be mentioned only briefly somewhere in the text. This allowed me to stick to the subject within each chapter while still providing a place to include other essential information.

The material in this book is based on the 1.3 Gamma version of the Workbench disk. In Commodore terminology, the Gamma version is the last pre-release version of a product, happening just prior to the product's actual release to customers. By the time this book is published, Workbench 1.3 will be finished and available to the public. I mention this here because there may be minor discrepancies between what I show to be the contents of a directory on the Workbench disk and what the contents might actually be on the final version. I have tried, however, to ensure that the descriptions of the commands and their options accurately reflect what will appear in the final version.

I hope you will find this book helpful in learning to use the Amiga.

Rob Peck
Los Gatos, CA
April 1988

Introduction: The Multitasking Environment

Your Amiga differs from other personal-computer systems in that multitasking was built into its operating system when it was first designed. Multitasking is the ability to do more than one thing at a time, that is, to run two or more programs simultaneously.

Some other computers provide a Workbench-like interface that lets you move and open icons on its screen, but when a program finally loads, it takes over the entire system, and the "Workbench" (or whatever it is called) disappears.

By contrast, the Amiga lets you run several programs at the same time. For example, if you open the Demos drawer, you can double-click on any one of the demos and it will open a window that runs that particular graphics program. If you open a second demo, the first one that you opened continues to run. In fact, you can open several copies of the same demo if you wish.

What does multitasking mean to you? Well, it means having the freedom to mix things together, so to speak. Most of the software written for the Amiga takes advantage of the Amiga's multitasking capabilities, but I have to note that there are some exceptions. A few developers of Amiga software have chosen to make their programs (games in particular) take over the machine rather than allow multitasking. The majority of Amiga software, however, is written with multitasking in mind.

You will probably find plenty of your own uses for the multitasking

capabilities of the Amiga. You may want to run a music program in the background of a slide show you put together. Or perhaps you will want to write a letter in one window while you are downloading a file with your terminal program. Or maybe play a game while taking a brief break from working with your spreadsheet program. You will even find that there are some programs that run "in the background," just waiting for you to hit some special key combination that will bring them to life to help you with your tasks.

You will see many references to multitasking in this book. It was important to tell you early about just what this means on the Amiga.

1 Introduction to the CLI

This chapter introduces the CLI and AmigaDOS. You will learn how to open the CLI and also how to pause and abort the output of CLI commands. Finally, you will use the DIR and CD commands to learn how AmigaDOS organizes files on disk.

AmigaDOS and the CLI

AmigaDOS, the Amiga Disk Operating System, is one of the most important parts of your Amiga. Besides letting you store data and programs on floppy and hard disks, it controls access to other parts of your system, such as printers and modems. You can communicate with AmigaDOS through the Workbench icon interface or the more powerful Command Line Interface (CLI). The CLI is similar to the disk operating system interfaces of other computers. If you have had experience with CP/M, Unix, and MS-DOS, the CLI should seem like second nature to you.

Opening the CLI

You run the CLI like any other program file on your Workbench disk; double-click on its icon, or single click and choose Open from the Workbench menu. The trick to opening the CLI, however, is finding the icon. Commodore put it in different places on the disk for versions 1.2 and 1.3 of Workbench. If you are using the latest version of Workbench, 1.3, you will always find the CLI icon in the System drawer.

If the CLI icon does not appear when you double-click on the System drawer, your Workbench is 1.2 (assuming that you have not changed the disk's configuration). In this case, you will have to use Preferences to make the CLI icon visible in the System drawer. Select the On

gadget for CLI in the main Preferences screen. Click on either SAVE or USE (preferably SAVE, as this will write the change to the disk) to exit Preferences. (Be sure your disk is not write-protected!) Now, when you open the System drawer, the CLI icon, a tiny window with 1> in it, appears in the System window. To shorten the clicking trip, you may want to move the CLI icon into the main Workbench window.

How to Tell If the CLI Window Is Active

The CLI window opens in the center of the screen and is titled New CLI Window. You can tell if the window is active (ready to accept input) by looking at the drag bar at the top of the window. If the window is not active, the drag bar will be "ghosted," meaning that a cross-hatch pattern will make the title harder to read. Although several windows can produce output at the same time under the Amiga's multitasking system, only one window can be active for input at any time. As a test, click the left mouse button once outside the boundaries of the window and notice that the window becomes inactive. Now reactivate the CLI window by clicking inside it.

The Prompt String

In the upper left corner of the window, you will see the CLI prompt, 1>. If you opened second and third CLI window's, their prompts would be 2> and 3>. The prompt means that the CLI is waiting for you to enter a command. If this window is active, then what you type on the keyboard is sent to the CLI. When you press the Return key, the CLI interprets what you entered. If it was a proper command, the CLI performs the command for you.

How the CLI Interprets What You Type

The CLI breaks up the characters you input on the command line into words. Each time the CLI finds a space in your input line, it assumes it has found a new word. The CLI considers the first word on a command line to be the command itself. The remaining words are parameters, information that the command needs to perform properly or words the command acts upon. Parameters are not interpreted by the CLI, but are passed to the command.

Because the CLI separates words with spaces, you must take care when a command or parameter (such as a disk name) contains a space. You must instruct the CLI not to treat the blank as a gap between words by enclosing the word in quotes. For example, the CLI recognizes:

ThisIsAWord as one word: ThisIsAWord

A Simple Sentence as word one: A, word two: Simple, word three: Sentence

"Workbench 1.3" as a single word that includes an embedded space.

A sample command line would be:

```
1> LIST test1 test2 "Name With Blanks"
```

In this case, the CLI considers LIST the command because it is the first word on the line, and the CLI sends it three parameters: test1, test2, and "Name With Blanks." If the quotes were not around the last parameter, LIST would receive five parameters: test1, test2, Name, With, and Blanks.

For simplicity, you should avoid using spaces within names, but when necessary the CLI can handle embedded spaces.

What Are CLI Commands?

CLI commands are instructions to the operating system. You can tell AmigaDOS to run a program by typing its name and any parameters needed into the CLI window. You can run application programs such as word processors or utilities on your Workbench disk. Most of the instructions you will issue from the CLI will be located in the C (short for "Commands") directory of your Workbench disk.

When you run a program from the CLI, it prohibits further keyboard input to the CLI. Only when the program exits (finishes) does the prompt return, allowing you to send additional commands to the CLI. This can be inconvenient if a program needs to run a long time and you want to do something else, such as copy a disk, while you wait. Thanks to multitasking, you can simply open another CLI and enter your commands. For example, if you have two CLIs open, the original

CLI still works on your first program, while the second CLI copies a disk for you.

Multitasking and New CLIs

If your first program fills the whole screen so you cannot see the CLI icon to double-click it, press the Left-Amiga key and the N key at the same time. (On some Amiga 500s and 2000s, this combination is Commodore-N.) The Workbench screen will pop to the front so that you can get to your CLI icon. Press the Left-Amiga and M keys to return Workbench to the back. This is a vital sequence because some programs open custom screens that cannot be dragged down with the mouse to expose the Workbench. Be aware that some programs take over the system and shut off the pop-Workbench key combination.

About File and Command Names

AmigaDOS stores information on disks in files. Files can hold programs, pictures, letters created with a word processor—anything that the Amiga can store. A file name in AmigaDOS can be as many as 30 characters and you may use both upper- and lowercase letters. Be careful—AmigaDOS does not differentiate between upper- and lowercase; MyFile, MYFILE, and myfile are all the same. (Just to keep things clear in the book, I will always use capital letters for CLI command names. I will use a mixture of upper- and lowercase characters in parameters.) Some valid AmigaDOS file names are:

```
fred
Mailing_List
My Own Recipes
letter.txt
```

Remember, if a file name contains spaces, you must enclose the name in quotes when you refer to it in the CLI.

Some operating systems tack an extension (a period and up to three letters) onto the end of a file name to show what type of file it is. AmigaDOS does not require file extensions but some Amiga programs use them, particularly language compilers. While not essential, file extensions can be helpful, especially if you are constantly transferring

files between AmigaDOS and other operating systems. If you are intent on compatibility, you should also avoid using embedded spaces and limit your names to eight characters. Here are some commonly used file extensions:

.pic a picture file
.txt an ASCII text file
.bas a BASIC program
.exe an executable file
.c C language source file
.h C language include file
.asm assembly-language source-code file
.arc an archive file used in modem file transfers

The Directory Function

Now that you can access the CLI and understand AmigaDOS naming conventions, you are ready to start using the commands themselves. The most basic AmigaDOS command is DIR. The DIR command first lists the names of other directories stored in the directory you are currently looking at, then lists the names of files kept there. Names followed by the notation (dir) as in

```
fonts (dir)
```

are directories, which can contain the names of other directories and files. Names not followed by (dir) represent individual program files or data files. Make sure that the CLI window is active and type:

```
DIR
```

then press Return. (From now on, I won't mention pressing Return. Just remember you must press Return at the end of every command line in order to pass the input on to the CLI.)

Assuming that you have your Workbench disk in the internal disk drive, the output from the DIR command, a directory listing, should look like this (only part of it is shown):

```
Trashcan (dir)
c (dir)
```

```
demos (dir)
. . . more names . . .
Clock      Clock.info
Preferences      Preference.info
. . . more names . . .
```

Did the information scroll past too quickly? If you want to take your time reading you can stop the output.

Stopping and Starting the CLI Output

You can pause CLI output by pressing any key, but the space bar is the best choice. While there is input pending (such as a space), the CLI will not try to output anything.

To resume scrolling, press the Backspace key, that is, erase the space you entered. The CLI was designed this way to avoid any possibility of mixing program output with user input.

Instead of deleting the space to continue, you could enter another command and press Return. The type-ahead feature lets you stack up a series of commands for the CLI to perform. Quickly type:

```
DIR

DIR

DIR
```

The CLI performs the DIR command three times, each time printing a prompt on a line, then beginning with the output of the next command. When it finishes, you get a prompt on a line by itself. The output from the CLI resumes each time you pressed Return, because there is no longer input pending.

Permanently Halting Output Display

If you change your mind once you issue a command, you can abort most commands by pressing the Ctrl key and the C key simultaneously.

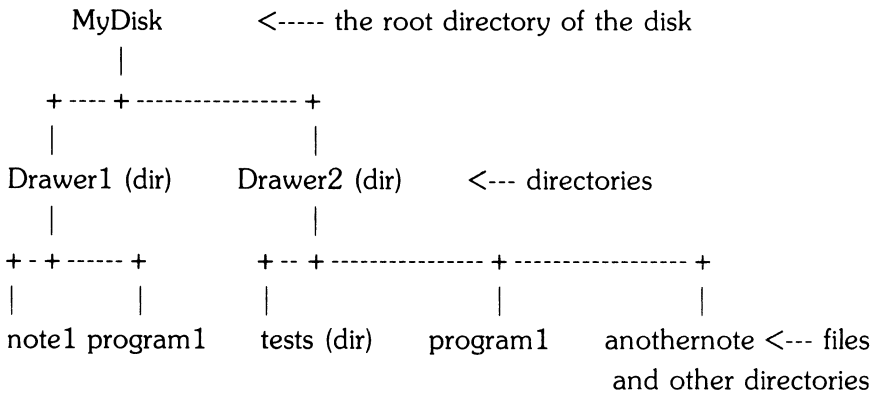
Following a CTRL-C, hit Return; AmigaDOS will report BREAK for most cases.

How Amiga Disks Are Organized

An Amiga disk is like a filing cabinet. It has a name, which is printed below the disk icon in the Workbench. (Note that AmigaDOS refers to a disk as a volume and to the disk name as the volume name.) Inside the filing cabinet (disk) are file folders (directories) as well as files. Just as file folders can hold other folders and files, so directories can hold other directories in addition to files. Files can be either programs or the data that your program creates. The Amiga lets you organize things on your disk by putting them into directories. Because you can organize your disk in a way that makes sense (such as keeping your word-processor documents in a directory called wp.data), you are able to find information quickly. As in Workbench, you can have two different programs with the same name on one disk, as long as you keep them in separate windows. The CLI knows the difference and does the right thing when each is selected.

Upon opening the CLI, you are at the "root" level of the Workbench disk. The following diagram (Figure 1.1) illustrates the relationship between directories and their contents.

Figure 1.1. The hierarchical storage arrangement of files and directories.



The name of the root directory of any disk corresponds to the name of the disk. If you were located in the root directory of MyDisk, you could find out what files or directories it contained by typing DIR. The CLI would respond:

```
Drawer1 (dir)     Drawer2 (dir)
```

Parents and Children

As with families, a directory that is above another directory or file (towards the root) is called a parent. MyDisk is the parent of Drawer1, and Drawer1 is a child, or subdirectory, of MyDisk. Just as your parent is also your grandparent's child, a parent directory can at the same time be a subdirectory.

How to Refer to a Disk

You can refer to a disk in ways other than the volume name. For example, if the disk you want to access is in the internal disk drive, you can issue this command:

```
DIR df0:
```

df0: is the name of the internal disk drive, disk-floppy (df) number zero (0). AmigaDOS identifies external drives by their position in a chain leading outward from the computer. On the Amiga 1000 and the Amiga 500, the first external drive is called df1:, the second is df2:, and the third is df3:. On the Amiga 2000, the first internal drive is called df0:; external drives are numbered consecutively beginning with df2:.

The colon is an important part of the name because it alerts AmigaDOS that a name either refers to a device or is somehow specially defined in its filing system. The colon also indicates that you are referring to something in the root directory of the device. When you refer to a root directory, you can use either its hardware name (df0:, df1:, df2:) or its root-directory name, also known as its volume name.

How to Look Below the Root Directory

Like other CLI commands, DIR accepts parameters. The DIR command lets you look at the contents of other directories. Simply type the

directory name on the same line as the DIR command and DIR shows you the contents of that directory instead of the contents of the current directory. On MyDisk (Figure 1.1), if you enter:

```
DIR Drawer1
```

the CLI would respond:

```
note1     test
program1
```

Now for the Workbench disk. When you first open the CLI, the current directory will be the root directory of the Workbench disk. Try these commands, one after the other:

```
DIR
DIR df0:
DIR "Workbench 1.3:"
```

(or insert the name of your Workbench disk)

```
DIR :
```

You should see the same output for each of these commands. The first command, DIR, lists the contents of the current directory, in this case the root level of the Workbench disk. The second command instructs the system to list the contents of the root level of the disk in the internal disk drive. Because this is the Workbench disk, again you get the same result. The third command asks the system to list the contents of a specific volume name, Workbench 1.3. The fourth command uses only the colon to refer to a directory. The colon refers to the root directory of the current disk, again, the root of Workbench. If you request a disk by a name that is not in one of the system disk

drives, AmigaDOS displays a little window called a requester with the message:

```
Please insert volume
named
name-you-specified
into any drive.

Retry    Cancel
```

When you insert a disk into a drive AmigaDOS reads the volume name on the disk. If it is the volume name that AmigaDOS was waiting for, AmigaDOS automatically retries the current command and the requester vanishes. If the volume name is incorrect, the requester will flash and again demand the proper disk.

A good habit is to write the actual volume name on the disk label so that when AmigaDOS requests a specific disk you can find it easily. Giving each disk a unique name can save you a lot of frustration later on, as well. Because it reads the identifying information on a disk (such as the date and time it was made), AmigaDOS can tell the difference between disks with identical volume names and contents. While you might insert a disk with the same volume name as AmigaDOS requested, AmigaDOS can tell that the disk is not exactly the same as the one it was expecting and will not accept it.

Showing Contents of Subdirectories

Try the following commands to look at the subdirectories below the Workbench's root directory:

```
DIR fonts
DIR fonts/ruby
```

The first command shows you the contents of the fonts directory, which contains the names of fonts (sets of letters and numbers of a particular style and size) that you can use in Notepad and the directories that house font data. A font file (for example, ruby.font) contains a description of the font, as well as the path names to the font data. In the second example, you specify the relative path name from the current

directory (the Workbench root), through the intermediate directories (in this case only fonts), to the destination subdirectory (ruby). Here at ruby, the last item in the path, AmigaDOS will list the contents of this subdirectory.

In the path name (fonts/ruby) you separate steps along the way with slash marks (/). Slash marks in a directory path determine the method AmigaDOS uses to get from a parent directory to the child (subdirectory).

Moving from Directory to Directory

The DIR command lets you stand in place, and look down the filing tree to see the contents of directories anywhere down the tree. Here you will learn how to move yourself down or across the branches of the tree, enter a directory, and then list its contents. To move from one directory to another, you use CD, the Current Directory command.

Type the command:

```
CD
```

AmigaDOS replies:

```
Workbench 1.3:
```

(or whatever your boot disk is named).

When you type CD without parameters, AmigaDOS returns the name of the current directory. It tells you where you are on the disk. When you use DIR by itself, you see only the contents of the current directory. With DIR you can see what you have and where you can go, but not where you are. CD tells you where you are. Between DIR and CD, you can get a complete picture of where you are and what is around you in the filing system.

Like DIR, CD accepts a parameter, the name of the directory you want to enter. Try these commands:

```
CD DF0:
CD fonts
CD
```

After the last command, AmigaDOS responds:

```
Workbench 1.3:fonts
```

Continue with:

```
CD ruby
CD
```

AmigaDOS answers:

```
Workbench 1.3:fonts/ruby
```

Finally, enter:

```
DIR
```

Notice that this combination gives you exactly the same list as when you typed DIR fonts/ruby from the root directory. The difference is that you have moved from one directory to another.

How AmigaDOS Searches Directories

The method AmigaDOS uses to search directories to find a specific file is important because it affects the way AmigaDOS performs your other commands. When you press Return, AmigaDOS looks at the first word on the command line and tries to find a file in its current search path that corresponds to that name. Normally, AmigaDOS will search the current directory first, then the C: directory. Other directories to search can be added to the search path using the AmigaDOS PATH command (see Chapter 7). If the command is found in one of these paths, AmigaDOS runs that program.

Possibilities for Error

If you ask the system to list the contents of something that is not a directory, AmigaDOS will tell you about your mistake. If you ask the system to change directories to a file name instead of to a directory, AmigaDOS tells you your error. Try:

```
CD DF0:
DIR fonts/ruby/12
```

AmigaDOS responds:

```
fonts/ruby/12 is a file not a directory
```

If you ask for a listing of a directory that does not exist, you are told it could not be found:

```
CD DF0:
DIR fonts/ruby/3
```

AmigaDOS responds:

```
fonts/ruby/3 not found
```

Many times, you may try to move to a directory that you know exists, only to be told that it is "not found." For example, if you type the command CD fonts from inside the devs:printers directory, the system will not be able to locate the fonts directory because there is no fonts directory inside of devs:printers. If you try CD df0:fonts or CD :fonts instead, AmigaDOS will find the directory.

Specifying a Directory Change

Up to now, you have only changed directories down the tree, from parent to child. In addition, you can specify the CD parameter to be an absolute path name that includes the disk name and all of the directories that you would have to pass through to get to the directory you want. An absolute path will take you anywhere on a disk, but you must begin at the root. A relative path name begins at the directory you are in and can only go down the tree. Finally, the CD parameter can be a special character, either a colon (:) or a slash (/). Using a slash moves the current directory back up one level to the parent directory. Using a colon moves you to the root directory of the current disk.

Starting at the root of the disk, look at the various ways you can use the CD command. Enter the following:

```
CD DF0:
CD fonts/ruby
CD
```

AmigaDOS responds:

```
Workbench 1.3:fonts/ruby
```

You have used CD to move down the directory structure. Now enter:

```
CD /
CD
```

AmigaDOS responds:

```
Workbench1.3:fonts
```

The slash told AmigaDOS to move to the parent of the current directory.

Now, try:

```
CD DF0:
CD fonts/ruby
CD //
CD
```

AmigaDOS responds:

```
Workbench1.3:
```

For every slash you type after the CD command, AmigaDOS moves one level toward the root directory of the disk.

Now from the root level type:

```
CD /
```

AmigaDOS reports:

```
Can't find /
```

This means that you are already located at the root of the disk and there is nowhere to go above you.

If you want to go directly from a lower-level directory to the root of the current disk, you use a colon (:).

```
CD df0:devs/printers
CD :
CD
```

The response is:

```
Workbench 1.3:
```

Finally, from the root directory, move to the printer directory with an absolute path name:

```
CD devs/printers
```

Now, to move to the fonts directory, which is not below the printer directory in the structure, enter the complete path name of the directory you want to move to:

```
CD df0:devs/fonts
```

As you can see, CD is very versatile; you can use it to move about your disks at will.

Interactive Directories

You can specify some options to use in conjunction with the DIR command by typing the keyword OPT after the DIR command. Entering DIR OPT I lets you view a directory listing interactively. The CLI shows you the names of every file and directory in the current directory, one at a time, with a question mark after each. You have many options at the question mark.

If DIR OPT I shows you an entry that has (dir) after it, you can enter that directory by pressing E. When you have viewed the last entry, DIR returns to the parent of that directory and shows you the next directory or file in the parent directory. To go back up toward the root before seeing all the files in a directory, type B.

At any point, you can view the next item in the directory by pressing Return. You can delete files and empty directories by typing DEL (the letters D, E, L—not the Delete key) and pressing Return. View the contents of files by pressing the T key. Note that you will get funny characters if you try this in anything other than a text file. To abort the DIR OPT I listing, press the Q key. If you forget what the OPT I commands are, type a question mark and the system will tell you what the letters stand for.

To get a feel for DIR OPT I, try the following sequence. Enter a Return after each question mark except where noted.

```
CD DF0:
DIR OPT I
Trashcan(dir) ?
c (dir) ?
demos (dir) ?
System (dir) ?
l (dir) ?
devs (dir) ?
s (dir) ?
t (dir) ?
fonts (dir) ? e
ruby (dir) ? e
12? b
opal (dir) ? e
diamond.font ? b
libs (dir) ? q
```

With DIR OPT I you can examine the directory structure of your disks at your leisure.

The Big Picture: DIR OPT A

If you are tired of seeing just the contents of the current directory or viewing the contents of a disk interactively, you can see everything that is on your disk with DIR OPT A. Think of it as DIR OPT "ALL." Try the following:

```
CD df0:
DIR OPT A
```

AmigaDOS returns a listing of every file and directory on the Workbench disk, indented nicely to show you the various levels in the directory structure of Workbench. DIR OPT A is not limited to the root

directory; it will show all files and directories in or below any directory. Compare the previous output with the following:

```
DIR devs OPT A
```

Unlike the previous output, this example shows only the files and directories in or below the devs directory.

One last DIR option you may find useful is DIR OPT D. This option lists only the directories in the current directory; it ignores the files.

Conclusion

So far, you have learned how to start the CLI. You have learned how to get a directory listing of your disks and how to move through the disk structure using CD. You have learned the importance of the AmigaDOS directory structure. You have also learned how AmigaDOS uses names. For instance, do you remember all the ways you can refer to a disk in your internal drive? Here they are:

- by the generic system name "df0:"
- by the actual name of the disk, such as "Workbench 1.3:"
- by the name of the root directory, ":"
- by no name at all, if your current directory is the root directory of the disk in the internal drive.

You have seen how the DIR command can be used in interactive mode, or how it can be made to show selectively the contents of the current directory, a subdirectory, or all directories and files. Most importantly, you have issued your first CLI commands. The rest of the book will take you deeper into AmigaDOS and the CLI.

2 AmigaDOS Shortcuts

Command Templates, Redirection, and Pattern Matching make AmigaDOS easier to use by making it easier for you to enter commands, redirect the input and output streams, and access multiple files with one command.

The AmigaDOS Command Template

Commands in the C directory of your Workbench disk have a unique feature known as a "command template." This is an English-like single-line description of what parameters the command takes, the order of those parameters on the command line, whether some of them are optional, and so on. To view the command template for any command, type the command, followed by a space and a question mark (?). Here is an example. When you enter:

```
DIR ?
```

the CLI responds:

```
DIR,OPT/K: _
```

and waits for you to enter parameters for the command. The underscore is where you enter the parameters that the command accepts.

The parameters that make up the command template are listed in the order in which the command expects you to enter them. Parameters are separated by commas in the command template. This is simply a convenient notation used in AmigaDOS. When you actually enter parameters or keywords, you separate them with spaces, not commas. The DIR command, therefore, expects a DIR parameter first, followed by an OPT parameter. In the command template, DIR and OPT are called *keywords*.

The DIR keyword—not to be confused with the DIR command—means that the command expects you to enter the name of a directory as a parameter. This name is the directory you want to list. The OPT keyword indicates that special display options exist for this command. You can access these options by using the OPT parameter.

In addition to keywords, the command template can also contain special characters that modify the meaning of keywords. Here is a list of the AmigaDOS parameter modifiers and their meanings:

/A indicates that the preceding parameter is required by the command; it must be present as part of the command line.

/K indicates the preceding keyword *must* be used with a parameter. Often, keywords are optional and a command will determine that a word on the command line is a parameter by its position on the command line. However, when a keyword is followed by the /K qualifier in the template, you must use the keyword in order to use the corresponding parameter.

/S means the preceding keyword is a switch. When used, it turns on some option. When used a second time, it turns that option off. We'll examine commands that use keyword switches later.

We can now make sense of the template for the DIR command. Because the DIR keyword is not followed by a /K, it is optional. In fact, no one ever uses this keyword. Also, because no /A follows DIR, the parameter itself is optional. You do not have to enter the name of a directory when you use the DIR command. If you enter DIR without parameters, the command still gives you a listing of the current directory.

For the next parameter—options—notice that OPT is followed by a /K in the template. This means that you cannot access the DIR options without using the keyword OPT. Also, you cannot use the keyword OPT without entering one or more of the options. Finally, note that because OPT/K is not followed by /A, the entire options parameter is optional!

Other commands use other keywords besides DIR and OPT. For an

example, let's examine and decipher the DISKCOPY command template. Enter the following:

```
CD DF0:SYSTEM
DISKCOPY ?
```

The system responds:

```
FROM/A,TO/A/K,NAME/K:
```

Here's what this template means:

FROM/A Because this parameter is followed by a /A, you must provide a FROM parameter when you invoke this command. The FROM parameter is the name of a disk. However, because FROM is not also followed by a /K, the keyword itself is optional. The FROM parameter can be indicated solely by its position on the command line. If no keyword appears on the command line before the first parameter, then the first parameter is assumed to be the FROM parameter by the command.

TO/A/K Because the TO keyword is followed by both /A and /K, this keyword and its associated parameter *must* be included on the command line. The TO parameter, which follows the TO keyword, is the name of a disk drive.

NAME/K NAME is an optional parameter. By using the keyword NAME, you can indicate the name of the new volume you create with the DISKCOPY command. You must use the NAME keyword to access this parameter.

To summarize, the command template shows you the parameters and keywords used by a command and the positions these normally occupy on the command line. The command template shows the parameters and their associated keywords, separated by commas; but when you enter the command arguments, *no commas are used*. Please remember this when you are entering commands. The command template is there to give you information about what the command

expects to see as its parameters; it is *not* a literal description of the method that you would use to enter the parameters.

Command Templates and a One-Drive System

The command template is printed by the command itself. When you type a command followed by a question mark, the command is loaded and takes over the CLI. The command is now waiting for input, not the CLI.

This is an important point to note, especially if you have only one disk drive. It means that it is not necessary to keep the disk from which the command has been loaded in the drive in order to have the command work properly! After you ask for the template, the command is in memory and is executing, waiting for you to enter its parameters. So, if you wish to get the directory of a disk that is not currently located in the disk drive, you could do the following:

```
DIR ?
```

The system loads the DIR command and responds:

```
DIR,OPT/K:
```

Now, you eject your Workbench disk from the internal disk drive and insert the disk whose directory you want to see. Then enter:

```
DF0:
```

You now get to see the directory of the new disk in the internal drive instead of the directory of the Workbench disk that was originally in the drive. Using the fact that you can enter command parameters after viewing a template, you can do a lot to overcome the limitations of a one-drive system.

For those of you familiar with other operating systems, such as CP/M or MS-DOS, a note here is in order. These operating systems have many of their commands built into the operating system itself. AmigaDOS, however, has virtually *all* of its commands external to the operating system itself. Thus, to get a directory listing, a disk containing the DIR program must be mounted on the system, and the CLI must know where to find the DIR program. Owners of single-drive systems will

often copy commonly used commands into RAM: and add RAM: to the search path for commands. More on this technique later.

Altering the Sequence of Parameters

AmigaDOS commands expect their command parameters to be entered in the same sequence as is shown in the command template. This is the default sequence. You may, if you choose, alter that sequence by altering the sequence of the keywords and parameters on the command line. As an example, remember the template of the DISKCOPY command. You enter:

```
DISKCOPY ?
```

and the command responds:

```
FROM/A,TO/A/K,NAME/K:
```

From the template, DISKCOPY requires two parameters (and permits an optional third). The keyword FROM is optional, while the keyword TO is required. The keyword NAME is required when you use the optional name parameter. The default sequence of parameters the command expects is:

DISCOPY	**DF0:**	**TO DF1:**
(command)	**(FROM parameter)**	**(TO parameter)**

Keywords explicitly define each parameter for the AmigaDOS commands. In other words, the keywords tell the command which of its parameters appears next on the command line. Thus, you can use keywords to enter the parameters out of the normal sequence. In the example above, you could enter:

```
DISKCOPY TO DF1: FROM DF0:
```

and get the same results as when you order the parameters according to the template.

When a command encounters a keyword and then reads the corresponding parameter, it expects the leftover parameters to be in the

same sequence (reading left to right) as they would have been in the template itself. Once again, from the above example, you can enter:

```
DISKCOPY TO DF1: DF0:
```

and get the same results. In this case, DISKCOPY first encounters the TO keyword. It then expects to find the leftover paramters (FROM and, optionally, NAME) in the order they appear in the template.

Thus, the use of keywords in the command line alters the method used by AmigaDOS to accept data as input values. The ability to alter the position of parameters lets you enter commands in a way that makes sense to you.

Let's look at some more examples of altering the command line. Examine the following:

```
DIR OPT A fonts
```

This command line for the DIR command uses the positional keyword OPT, telling DIR to consider the next parameter on the command line as an option instead of as a directory name. If we were being totally consistent, using all keywords, the command would look like this:

```
DIR OPT A DIR fonts
```

Here we are using the DIR command's keyword DIR to specify explicitly which directory we want to examine. The DIR command interprets the above line to read "accept 'A' as an option and accept 'fonts' as the DIR parameter." (Once the OPT parameter has been defined, there is only one other parameter to define, and that is the DIR parameter, so its keyword need not be used.)

There may be some instances in which you may want to use the keyword, not because it is needed, but just because it is comforting. For example, the COPY command (detailed in Chapter 6) has the template:

```
FROM,TO/A,ALL/S,QUIET/S:
```

This is a case where the TO keyword has a /A after it, meaning that the TO parameter is required (some name for the destination of the

copy is needed) but the keyword TO itself is not required. Personally, rather than issue the command as:

```
COPY SourceFile DestinationFile
```

I feel more comfortable using the form:

```
COPY FROM SourceFile TO DestinationFile
```

This is just a matter of personal taste. Some people would rather leave out the FROM keyword and simply type the command as:

```
COPY SourceFile TO DestinationFile
```

This alternate form sounds and feels "right." Because AmigaDOS has this flexibility, you can use whichever form that AmigaDOS allows, and that also makes you comfortable.

When I use an alternate form, I am not changing how a command works or what AmigaDOS does. I am just making sure, in my own mind, that I am issuing the command correctly by issuing it in a way that makes sense to me.

Template Summary

A command template is available for most commands, simply by typing a question mark after the command name in the CLI window. The template specifies the parameters and keywords expected by the command. After viewing the template, you can perform the command by entering the parameters, and possibly some of its keywords, and then pressing Return.

Command templates specify keywords, indicating the positions that the parameters associated with those keywords are to be entered. To modify that sequence, simply specify a keyword preceding the appropriate parameter, and AmigaDOS will accept automatically the parameter associated with that word.

If you have the command template showing in the CLI, it means that the command has been loaded from disk and is now executing. This means you can actually remove the disk from which the command

was loaded and, if you wish, substitute a different disk for the rest of the command.

Command Redirection

In addition to commands, parameters, and keywords, another important item may appear on an AmigaDOS command line—the command input and output redirection symbol. Redirection gives you the ability to force the output (or input) from a command to go to (or come from) somewhere other than where it is normally expected.

Most programs (including AmigaDOS commands) have a "standard input" location, a "standard output" location, and a "standard error" location. Normally, standard input is supplied by the keyboard, while standard output and standard error locations are the CLI window. In other words, the program receives input from what you type, while what the program types—whether normal output or error messages—becomes visible in the CLI window. Note that very few AmigaDOS commands allow redirection of the standard input. This feature is described here for the sake of completeness.

Let's say you would like to create a file that contains all of the output from a command. You can "redirect" the output of the command from the CLI window to the file. This is done by specifying the redirection—along with the file name—on the command line, immediately following the command itself. Here is an example of sending a directory listing to a file named 'dir.list' instead of to the CLI window:

```
DIR > dir.list "Workbench 1.3:"
```

The greater-than (>) symbol means redirect the output. From a physical viewpoint, the symbol looks like an arrow pointing out of the command and into a file name. If you wanted to redirect the input, so as to feed the contents of a file to a program instead of getting input from the keyboard, you use the less-than (<) symbol, where the symbol looks as though the file name is pointing towards the command.

For those of you familiar with the Unix operating system, AmigaDOS redirection is only slightly different. Under Unix, redirection is specified at the end of the command line; in AmigaDOS *it must directly follow*

the command itself. Here is an example that shows redirection of both input and output:

```
TYPE < input.file > output.file
```

This command types the input it gets from input.file into a file called output.file.

Keyword Equivalents

Note that the greater-than symbol can also be represented by the word "TO" and the less-than symbol by the keyword "FROM." So, for example, the COPY command, which has the command template:

```
FROM,TO/A,ALL/S,QUIET/S:
```

is actually copying things from its standard input to its standard output and wants you to tell it where to connect the input and output.

To see if redirection works, try the following:

```
DIR > dir.list
```

You will notice that nothing is printed to the screen, but you will now have on your disk one more file than before, a file called dir.list. To examine its contents, issue the command:

```
TYPE dir.list
```

and you will see the same directory listing you have seen many times before. The difference is that you can now use your word processor to edit this listing if you wish. You can even combine many such files to form an index of the contents of all of your disks in order to find what you are looking for in a batch of disks.

If you use the forward and backward arrows for redirection, AmigaDOS will also allow you to attach the arrows directly to the file names, with no intervening space, such as:

```
DIR >dir.list
```

Command Redirection Summary

Input redirection means taking the input to a program from somewhere

other than the expected source (normally the keyboard). You specify input redirection by using the less-then sign (<) or the keyword FROM on the command line. You specify output redirection by using the greater-than sign (>) or the keyword TO on the command line. What follows the output-redirection symbol keyword is normally a file name or an output device, such as the printer device (PRT:).

Command redirection works only if the redirection keywords immediately follow the command name (in either order: input . . . output, or output . . . input), before any other parameters.

Pattern Matching in AmigaDOS

When you get serious about computing, you often name files in a way that suggests something about the contents of the files. For example, you might give ".c" endings to all C language source-file names and ".asm" endings to all assembly-language file names. Similarly, you might want to find (or copy or delete) files whose names have something in common. Rather than work on each file individually, you can access multiple files at once using pattern matching.

Some AmigaDOS commands let you access files, not by entering the entire file name but by specifying a pattern to locate. This pattern can be composed of normal characters, special characters, or normal and special characters combined. Note that just as during normal file-name searches, AmigaDOS does not differentiate between uppercase and lowercase letters during pattern matching. When you search for an "X," an uppercase or a lowercase letter will match your request.

AmigaDOS compares file names with the pattern you enter and tries to find a match. The special characters used by AmigaDOS in pattern matching are #, ?, %, |, ( ), and ' .

Interpreting Patterns

Here is how AmigaDOS interprets patterns. In the explanation below, <n> represents any combination of characters and special characters. <n1> and <n2> represent different character and special-character combinations.

? matches any single character. If, for example, you wish to type

the contents of all files beginning with the letters "cli" and having exactly four characters, you would enter the command:

```
TYPE cli?
```

#<*n*> matches zero or more occurrences of a particular pattern. If you wish to type all files in the current directory starting with the letter "c"—no matter how many other letters are contained in the file name—you could specify the pattern as:

```
TYPE c#?
```

where the # says match any and all occurences of the following pattern and the ? says "any letter" is the pattern. This is the most common use of pattern matching in AmigaDOS.

<*n1*><*n2*> You can form a sequence of patterns to look for simply by writing them directly adjacent to each other (no intervening blanks). AmigaDOS will look for the file by trying to match the first pattern; then using whatever is left of the file name, it will attempt to match the second pattern. Actually, this two-in-a-row pattern is treated simply as a single pattern. As an example:

```
TYPE c#?a#?
```

will type files whose names begin with "c" followed by zero or more occurrences of any letter, followed by an "a" followed by zero or more occurrences of any letter.

While the following names match this pattern:

Commodore-Amiga
CA
California

here is one that doesn't:

vacation-slides

The latter doesn't match because even though there is a "c" and an "a" in the name, the pattern says "the match must start with a c" and then match the rest of the specifications.

(<n1><n2>) shows parentheses are used to group patterns together.

<n1>|<n2> demonstrates how a vertical bar is used to separate alternate patterns to match. If the file name matches either pattern, it is considered a match.

% matches the null string. A null string has no characters in it. You probably would use this notation in combination with other patterns. For example, if you wanted to delete the files named

test.13
test12
test25
test.10

but retain files named

test431
test...999

the following pattern would indeed specify the match correctly:

```
DELETE test(%|.)??
```

This command line specifies that you delete files based upon the following rule:

"test"—match the first four letters exactly.
(%|.)—match either a null string (no letter at all) or a period.
??—match two single characters of any kind.

' A single prime (or single-quote mark) instructs AmigaDOS not to treat the next character as a special character, but instead to regard it as part of the string to match. This allows you to include special characters as part of your file names and still be able to use pattern

matching. Here are a couple of examples (other patterns also work; these simply show two possible alternatives):

File To Find	A Pattern	What It Says
MyFiles?	M#?'?	Start with M, allow anything, ends with ?
Any#Of#s	#?'#?'##?	anything, then a #, then anything, then a #, and then anything.

Pattern Matching Summary

Some AmigaDOS commands allow you to specify patterns as well as file names as parameters; the command then performs its operation on all files that match a particular pattern. Pattern matching is performed on all paths that are currently set. AmigaDOS treats the characters ( ) # ? ' | as special characters when specifying patterns.

3 Tools on the Workbench

As you can see from the gas gauge in the disk window, the Workbench disk is pretty full. This chapter will detail the contents of the Workbench disk and help you determine what you can delete to make your working copy of Workbench more efficient.

What's on Workbench 1.3?

Here is a DIR listing of the Workbench 1.3 Gamma release (the prerelease developers' version of 1.3). The contents of your Workbench disk may vary slightly from this listing:

```
Trashcan (dir)
C (dir)
System (dir)
L (dir)
devs (dir)
S (dir)
T (dir)
fonts (dir)
libs (dir)
Empty (dir)
Utilities (dir)
Expansion (dir)
.info                   Clock
Clock.info              Disk.info
Empty.info              Expansion.info
Preferences             Preferences.info
Shell.info              System.info
Trashcan.info           Utilities.info
```

Directories, Icons, and Workbench

Files whose names end with .info are icon files that belong to the Workbench interface. In the above listing, some files whose names end in .info have corresponding directory files. For example:

```
Expansion (dir)     Expansion.info
Utilities (dir)     Utilities.info
```

These .info files describe the icons for Workbench drawers. The .info file has a description of the picture that Workbench draws for the drawer, the drawer's alternate picture (some programs draw the drawer opened when the icon is selected), the icon's position within the Workbench window, and the data that define the size and position of the drawer's window when it opens.

When you open the icon for a drawer, Workbench automatically moves into the directory that has the same name as the drawer, opens its window, and then draws any icons that it finds in the drawer.

You will find several kinds of icons on the Workbench disk—drawer, trashcan, tool, disk, and project. When you click on a **drawer** icon, Workbench enters the associated directory and shows you its contents. Similar to a drawer icon, a **trashcan** icon adds the capability of deleting files from the disk. You can delete all items moved into the trashcan by selecting Empty Trash from the Workbench Disk menu.

Opening a **tool** icon, such as Clock or Preferences, calls up the program of the same name. Examples from the Workbench directory list are:

```
Clock       Clock.info
CLI         CLI.info
Preferences     Preferences.info
```

Using **disk** icons, you can ask to have a disk initialized or copied. You can also drag a tool icon on top of a disk icon to file the tool in the disk's main window.

A **project** icon represents the data that resulted from using a tool. For Notepad, the Notepad icon represents the text of the document. For an Amiga Basic program, the project icons represent the Basic programs. When you open a project icon, it looks in its search path

and tries to find the program that created the project. If it finds the program, for example Notepad, AmigaDOS loads the program and loads the text into it. If AmigaDOS cannot find the program in its search path, the screen flashes and nothing loads.

Directories with Icons

The following directories have corresponding icons and Workbench drawers:

Empty (dir)
System (dir)
Demos (dir)
Utilities (dir)
Expansion (dir)
Trashcan (dir)

As its name implies, the Empty directory is empty. It is included as a template for making new directories with icons. When you select a drawer icon and select Duplicate from the Workbench menu, you copy not only the drawer, but also its contents. With the Empty drawer you can easily create new directories and icons without duplicating unwanted files.

The System Directory

The System directory contains tools that you can call from the Workbench—DiskCopy, FastMemFirst, NoFastMem, FixFonts, Format, InitPrinter, GraphicDump, and SetMap.

DiskCopy lets you make backup copies of your disks. Selecting the DiskCopy icon, however, does not start the copying function. You have to use the CLI DISKCOPY command or drag one disk icon on top of another.

FastMemFirst rearranges the system memory lists so that autoconfigured memory (expansion memory that automatically tells the operating system, via special routines in Kickstart, what it is, where it is, and what it needs for system resources) appears on the system list

ahead of any other non-autoconfigured memory. When we examine recoverable RAM disks in Chapter 4, you will see why this tool is useful.

Rather than rearranging, **NoFastMem** disables all memory beyond the 512K that reboots your system and that the custom chips (Agnus, Denise, and Paula) can access. Some early programs written for a 512K Amiga assume that there is no other memory available and will not run on expanded systems. Using NoFastMem you can run many of these early programs on Amigas with more than 512K of memory.

If you delete or add any fonts to your fonts directory, the internal list of available fonts may not reflect the current contents of your disk. Running **FixFont** patches the system list to match the disk configuration. You need to run this tool only if you change the contents of the fonts directory after a boot. AmigaDOS updates the system's font list automatically when you boot the system.

Format lets you prepare a new disk to store information. You can call this function with Initialize on the Workbench menu or with the CLI FORMAT command.

Many printers require the computer to send an initialization sequence (a special string of characters) before sending a document. If your software does not transmit such a sequence, you can use **InitPrinter** to initialize your printer.

GraphicDump lets you take a snapshot of your screen and "dump" (send) it to your printer.

SetMap lets you specify how the Amiga interprets the keyboard. You can change the keyboard to use American, British, French, or any of eight other international character sets.

I use DiskCopy and Format almost every day and keep them on my work copies of the Workbench disk. Because I use the rest of the programs in the System directory less often than DiskCopy and Format, I store them on my backup copy of Workbench. You can rearrange the contents depending on your needs.

Be careful to *never* modify your original Workbench disk. Always experiment with a *copy* of the disk, so you can return to a clean, complete copy of Workbench if your experiment goes awry. From now

on, when I say you can delete a file or directory from your Workbench disk, I am referring to a working copy, not the original.

The Demos Directory

If you want to show off your Amiga, but have investigated only as far as the Workbench disk, click open the **Demos** drawer. Inside are four simple graphics programs (Boxes, Spots, Dots, and Lines) that demonstrate the computer's multitasking, speed, and graphics capabilities.

The Utilities Directory

Like your desk drawer that holds paper, pencils, adding machines, and other necessities, the Utilities directory contains programs to make your computer desktop neater and more efficient. The following are only capsule descriptions.

While it will not fit in your pocket, **Calculator** is a calculator program that requires some of the math library files to run.

ClockPtr turns the mouse pointer into a digital clock. Although tiny, the versatile clock can show the time, the date, or an interval timer that ticks off seconds from the time the program begins to run.

The **Cmd** program lets you capture the output of the printer device (PRT:) in a disk file. Instead of sending the information directly to the printer, the serial or parallel device writes to a file. This command is particularly helpful in debugging printer drivers, because you can see if the correct sequences are being sent to the printer.

More lets you see the contents of text files one page at a time, scrolling forward or backward.

Notepad is a rudimentary word processor that can use the fancy fonts in the fonts directory. The note-taker also requires the diskfont.library.

The **Say** program lets you demonstrate the Amiga's speech capa-

bilities, by repeating aloud what you type. To recite, it requires coaching from the narrator.device and the translator.library files.

The Expansion Directory

Initially empty, the Expansion directory is mainly for the convenience of Amiga hardware developers. To make a particular piece of hardware talk to the Amiga, AmigaDOS may have to run a program from the Expansion directory to recognize the device. Each time you do a warm boot (simultaneously press the Ctrl, Left-Amiga, and Right-Amiga keys, or the Ctrl, Commodore, and Right-Amiga keys on some A500s and A2000s), AmigaDOS looks in the Expansion directory for programs. If it finds programs, AmigaDOS runs them and can then access your expansion hardware.

All you have to do to add hardware to your system is drag a copy of the companion software's icon into the Expansion drawer, and perhaps add a device driver to the devs directory. Unless you plan to add on to your system immediately, you can delete the Expansion directory from Workbench until you need it.

The Trashcan Directory

The Trashcan directory's only purpose is to serve as a temporary holding place for files and their icons. At any time you can select the Trashcan icon and Empty Trash from the Disk menu to delete all the files in the directory. Because the Trashcan is the only directory that allows mass deletions from Workbench, you should keep it on all your copies of Workbench.

Directories Without Icons

Because they contain no tool or project icons and you cannot directly access them from Workbench, some directories have no drawer icons attached to them. These directories contain system commands you

can run from the CLI. From the simplest to the most complex, the directories are:

T (dir)
S (dir)
libs (dir)
fonts (dir)
L (dir)
devs (dir)
C (dir)

The T Directory

In the T (Temporary) directory, AmigaDOS builds temporary files that it uses to perform certain commands. For example, the EXECUTE command (described in Chapter 8), creates files that have names such as Command-T-01, Command-T-02, and so forth.

These files exist only for the duration of the command you are performing; AmigaDOS deletes them automatically when the command is completed. When you use the EDIT command (see Chapter 6), AmigaDOS stores a backup file, edbackup, in the T directory. The file remains after you complete a session with EDIT, so that you can refer back to a previous version of your text file. The file is replaced by the backup of the next file you edit. You must leave the T directory on the Workbench disk to allow the EXECUTE command to function. If you delete it, you will get an error message, such as:

Unable to create :t/Command-T-01

You can usually tell whether a file is needed by the programs you run if, after deleting a Workbench file, a program does not load or run properly. Simply copy the deleted file back into its proper directory on the Workbench disk, and try running the program again. If the program runs, then it probably needed the file.

The S Directory

The S directory contains the **startup-sequence** file. A startup-sequence is a text file that contains a series of commands that you want AmigaDOS

to perform automatically when you turn on or warm boot the system. If you want to see what your startup-sequence contains, type:

```
TYPE :s/startup-sequence
```

The S directory startup-sequence file must be present on all your work disks to bring up the Workbench. The startup-sequence also contains commands that are stored elsewhere on the Workbench disk.

The libs Directory

The libs directory contains libraries of routines. A library is a collection of individual routines and functions that can be used by independent programs. Most programs are self-contained and need take information only occasionally from the Amiga's ROM (read only memory). Some programs, however, use functions that are within the libs directory's libraries.

When a program opens a library, it reads the library into the system memory. As long as the program is running, the library is part of the system. If you start another program (task) that needs functions in the library before the first program is finished, the second program will ask the system to open the library. Multiple tasks can access the same library; the operating system keeps one copy of the library in memory at a time and monitors which tasks are using it. When all tasks using the library are finished, the system removes the library from memory, freeing up the space it used.

When you type DIR :libs AmigaDOS will list:

diskfont.library	**icon.library**
info.library	**mathieeedoubbas.library**
mathieeedoubtrans.library	**mathtrans.library**
translator.library	**version.library**

You use the **diskfont.library** when you wish to load a font from a disk. The routines in the diskfont.library look to the Workbench fonts directory to determine where and how to find the fonts requested.

Workbench uses the **icon.library** to handle its icons. Programs can use functions in the icon.library to create or modify icons, as well.

The **info.library** also contains routines used by the Info menu item on Workbench.

The **mathieeedoubbas.library** contains the IEEE math double-precision functions. The **mathtrans.library** holds single-precision transcendental math functions (such as sine), while the **mathieeedoubtrans.library** houses their double-precision counterparts. If you are not running any programs that use floating-point math functions, you can save some space by deleting these files.

The **translator.library** contains the translate function, which changes English text into phonetic text for the narrator device. This lets your Amiga "talk" to you by repeating files aloud.

The routine in **version.library** lets Workbench and other programs determine which version of the system software they are running under.

If you do not run programs that call the math functions or the translators, the associated libraries might be fair game to delete from your disk.

The fonts Directory

The fonts directory contains many different fonts. If you type:

```
DIR :fonts
```

you will see the list of font styles you can choose for your screen display and word processor:

```
ruby (dir)
opal (dir)
sapphire (dir)
diamond (dir)
garnet (dir)
emerald (dir)
topaz (dir)
diamond.font        emerald.font
garnet.font         opal.font
ruby.font           sapphire.font
topaz.font
```

If you give the command:

```
DIR :fonts/ruby
```

you will see:

```
12    15
 8
```

Each of the directories within the font directory contains numbered files. The numbers represent the point sizes (a typographical measure equivalent to $^1/_{72}$ of an inch) of the type and the files contain the actual bit patterns of the font. Each of the .font files describes the font's characteristics and contains, among other things, the names of the numbered files within that font's directory.

Because the two Topaz fonts (8 and 9) reside in Kickstart (or the system ROM of the A2000 and A500), you can remove all of the fonts and the fonts directory, and still have the Topaz fonts available. Before you start blindly deleting, note that some programs use fonts other than Topaz.

The L Directory

When you type DIR :L AmigaDOS will list:

Aux-Handler	**Disk-Validator**
FastFileSystem	**Newcon-Handler**
Pipe-Handler	**Port-Handler**
Ram-Handler	**Shell-Seg**
Speak-Handler	

The **Aux-Handler**, AUX:, is similar to a serial device. Using AUX: you can connect a terminal or another Amiga to your machine. You can then run CLI-based commands from the second terminal, turning your Amiga into a multi-user machine.

AmigaDOS uses the **Disk-Validator** to make sure that it knows about all of the files on your disk. When AmigaDOS creates a file, it uses sections of the disk, called sectors, in different areas of the disk. AmigaDOS keeps lists of used and unused sectors. Sometimes when a program exits, it may fail to close a file that it has opened. In this

case, AmigaDOS loads the Disk-Validator to look at all of the files on the disk and make sure that it can still find each of them and all the related pieces. If AmigaDOS can find all the files, all is well. Files that were opened and never closed may simply disappear once the validator has finished its work. If there is some other problem (possibly because of physical damage to the disk), the validator will be unable to repair the damage and will report:

Unable to validate disk
Run DiskDoctor to repair

or a similar message. You must then run DISKDOCTOR (found in the C directory) to salvage as many of the files as the repair program can find. Such errors seldom happen, but you should be prepared for emergencies. You must keep the Disk-Validator available simply to check the filing system on each disk as you first insert it into the drive.

A file system is a group of routines used by applications and system software that matches a file name with a physical location on a disk, allowing you to list, read, and write disk files. The default AmigaDOS file system resides in Kickstart.

FastFileSystem is a new innovation in AmigaDOS 1.3. Through version 1.2 of AmigaDOS, the system was slow in finding, reading, and writing files on disk. The FastFileSystem is a reworking of the AmigaDOS file system that makes access to disk files much faster while maintaining compatibility with all Amiga software. Operation of the FastFileSystem is completely transparent to the user and the programmer.

One limitation of the current version of FastFileSystem is that it does not work with Amiga 3½-inch floppy drives. To use FastFileSystem, a disk drive must be described in the system mountlist and be explicitly mounted. Because the 3½-inch drives are automatically mounted when you start up your system, they are restricted to using the old file system contained in ROM. When, in a future release of the operating system, the FastFileSystem is put into ROM, the floppies will be able to use it also. For now, it is available to hard-disk systems and the recoverable RAM disk that comes with AmigaDOs 1.3.

The **Newcon-Handler** provides additional capabilities to the console device, such as command-line editing and command-line history for

use with the Shell, an alternate CLI. Using this handler saves keystrokes for the typical CLI user.

The **Pipe-Handler** lets you connect the output of one program to the input of another, a very handy way to do several functions in a single command.

AmigaDOS loads the **Port-Handler** when you use the serial or parallel port through the SER: (the serial port), PAR: (the parallel port), or PRT: (the printer device). This separate handler controls the input or output, including opening and using the serial.device or parallel.device (from the devs directory, described below). You set the characteristics of the devices with Preferences. The Port-Handler simply uses these characteristics when you ask AmigaDOS to transfer the data to the ports.

The **RAM-Handler** controls the RAM disk, a pseudo-disk that resides in memory and acts just like a hardware disk (such as df0:). You can copy files to the RAM disk, list its directory, or find out information about its files just as with a "normal" disk. The main advantage of a RAM disk is speed; it is much faster than a floppy-disk drive. Because it is all electronic and does not have to turn a motor or spin a disk to read and write, it can operate at the speed that the system can read from or write to memory. Think how fast your system would be if AmigaDOS called your CLI commands from a RAM disk instead of accessing the hardware-driven Workbench disk.

A RAM disk is an especially good friend to owners of single-drive Amigas. By copying files from a source disk to the RAM disk, and from there to the destination disk, you can save a lot of the usual disk swapping. I advise keeping the RAM-Handler on your Workbench disk.

If you wish to use the Shell, you must copy the **Shell-Seg** into memory. In previous versions of the operating system, the CLI was the only type of text-based command interface supplied with the system. The Shell is an advanced version of the Command Line Interface. Anything you can do with a CLI you can do with the Shell, but the Shell gives you more capabilities. See Chapter 10 for more information about the Shell.

The **Speak-Handler** is an alternative to the SAY command, allowing

you to redirect not only files and lines of text, but also the output of files to the narrator device.

The devs Directory

When you type DIR:devs AmigaDOS will list:

```
keymaps (dir)
printers (dir)
clipboards (dir)
clipboard.device          mountlist
narrator.device           parallel.device
printer.device            ramdrive.device
serial.device             system-configuration
```

Used by the SetMap program, the **keymaps** directory contains several files that are the remapping of the keys for American and international character sets.

The **printers** directory contains drivers for many different printers. You can keep only the driver for the printer you normally use and remove the others from Workbench, until you need them. Just because you do not see the specific brand or model name of your printer in the list does not mean you cannot use that printer. Many printers are not Epsons, for example, but are Epson compatible (behave like Epsons), so you can use the Epson printer driver.

The **clipboards** directory is where the **clipboard.device** (also contained in devs) writes its files. The clipboard.device provides a means of transferring data between applications. Your program can cut something out of one application's window and paste it into another application.

A device is a group of functions that manage a hardware item. When a program opens a device, those functions become available to the rest of the program, just as with libraries. When a program wants to talk to the serial port (parallel port or printer), for example, the program forms commands and sends them to the **serial.device** (or **parallel.device** or **printer.device**). The device, in turn, performs the commands, then tells the calling program the results. Through the Amiga's multitasking the program can go on to do something else while the

serial (or parallel or printer) input and output continues independently. A second advantage is that programs need not independently create routines for manipulating the hardware each time they require the serial.device, because the serial.device (the serial-port routine library, so to speak) already contains functions that handle the most common uses of that hardware.

Talking programs, such as in the SAY command, use the **narrator.device**. If you do not have programs that access the Amiga's built-in speech capability, you can delete this file from Workbench.

The **ramdrive.device** is a recoverable RAM disk, named RAD:. Unlike standard RAM disks, programs and data files located in a recoverable RAM disk will usually be available after a system reset.

The CLI command MOUNT uses the **mountlist**. It provides a list of the characteristics of various devices that are not recognized automatically by AmigaDOS. For example, if you attach an external 3½-inch disk drive to your system, AmigaDOS would recognize it and read its disk at system reset. A 5¼-inch disk would not be recognized, so you must issue a MOUNT command before AmigaDOS can use the disk.

Each time you reset the system, AmigaDOS looks in the **system-configuration** file to find an encoded version of the settings you saved in Preferences. For example, it saves your screen colors, the size and shape of your mouse pointer, which printer driver you have chosen, and so on.

The C Directory

The C Directory houses many of the CLI commands. Here, I have simply listed the commands, grouped by type. I will describe each in depth over the next several chapters in roughly the same order as they are presented below.

Commands that give you information about:

- your disks **INFO, DIR, LIST**
- the filing system **ASSIGN**
- CLI's now executing **STATUS**
- why a command failed **WHY**
- the version of the operating system **VERSION**

- the time and date **DATE**
- the contents of a file **TYPE, MORE**
- the current directory **CD**

Commands that let you modify file characteristics to:

- attach an explanatory note to a file or directory **FILENOTE**
- change the name of a file or directory **RENAME**
- join two files together **JOIN**
- change the name on a disk **RELABEL**
- remove a file or an empty directory **DELETE**
- set the protection bits in the file description **PROTECT**

Commands that you use to control the filing system to:

- move from one directory to another **CD**
- create a new directory inside the current directory **MAKEDIR**

Commands that are system utility programs to:

- discover how much memory is available **AVAIL**
- duplicate disks **DISKCOPY**
- copy files **COPY**
- repair a filing system so you can copy files to a new disk **DISKDOCTOR**
- send a special signal to a running task **BREAK**
- create another CLI **NEWCLI**
- close a CLI **ENDCLI**
- turn the mouse pointer into a clock icon **CLOCKPTR**
- edit a text file **EDIT**
- act as a data filter (similar to editing but allows you to read from an input file and write to an output file while executing editing commands) **ED**
- locate text patterns in a group of files **SEARCH**
- sort lines of a file in various ways **SORT**
- use the Amiga's speech (and the SPEAK: device) **SAY**
- create a bootable disk **INSTALL**
- load and display the Workbench if it is not present **LOADWB**
- start a program running as a separate process **RUN**
- start a batch file running **EXECUTE**

Commands that are normally used within Execute (batch) files to:

- set a value to test against as an exit condition **FAILAT**
- test something and decide what to do next **IF**
- do something if the test does not succeed **ELSE**
- define the end of an if-else block **ENDIF**
- deliberately define a failure value **FAULT**
- skip forward in a batch file to a specified label **SKIP**
- define a label for SKIP to find **LAB**
- send a message to the user while performing a particular batch-file step **ECHO**
- make the batch file wait a specific time before continuing **WAIT**
- get input during a script file **ASK**

Commands that change the way AmigaDOS works to:

- specify the current time and date to AmigaDOS **DATE, SETDATE**
- provide more space for certain kinds of programs **STACK**
- modify the CLI prompt **PROMPT**
- install new drivers for devices that are not part of the basic operating system **BINDDRIVERS**
- modify the priority of a running task **CHANGETASKPRI**
- provide more memory for AmigaDOS and speed up disk access **ADDBUFFERS**
- tell AmigaDOS that you have changed a disk in a 5¼-inch drive **DISKCHANGE**
- tell AmigaDOS where to search for the commands you ask it to perform **PATH, ASSIGN**
- add a new item to the filing system **MOUNT**
- change the system memory lists **FASTMEMFIRST, NOFASTMEM**
- change system font lists **FIXFONTS**
- initialize a printer **INITPRINTER**
- change the Preferences **PREFERENCES**

Commands associated with the command Shell to:

- get the value of an environment variable **GETENV**
- set the value of an environment variable **SETENV**
- make commands Resident **RESIDENT**
- invoke a shell **SHELL**

In Summary

You have surveyed both the icon-driven and the invisible, but powerful, programs on the Workbench disk. You learned the capabilities and limitations of each of the files, as well as which files are interdependent. Based on your new understanding of Workbench's contents, you can modify your working copy to contain only the directories and files essential to you. Remember, never alter the original disk; keep it as a master to make copies from. With your first exposure to the commands in the C directory, you are halfway to mastering the CLI. You know what the commands are and what they can do; the next step is to learn how to use them. You will start, in the next chapter, with commands that give you information about the operating system.

4 Information Commands

This chapter describes all of the commands that return information about AmigaDOS and the files on your disks, with the exception of the DIR and CD commands covered in Chapter 1.

The information commands include: DIR, CD, LIST, INFO, ASSIGN, STATUS, WHY, VERSION, DATE, and TYPE. Some of these commands not only provide information, but also control certain characteristics of the system. Having already examined the DIR and CD commands in depth, let's begin with LIST.

The LIST Command

While DIR provides a simple list of file names and directories, the LIST command gives much more information about those files.

When no parameters are given, LIST outputs information about the current directory. If any of the files has a comment (called a filenote) attached to it, that filenote appears on the line immediately following that file or directory in the listing, preceded by a colon in column 1.

Here is an example of the output of the LIST command, produced by entering LIST in the CLI:

```
Trashcan                Dir    rwed   04-Mar-87    12:38:36
cli.2                 11654    rwed   Sunday       16:38:55
cli.3                 28661    rwed   Sunday       16:38:51
cli.1.rev             26837    rwed   Sunday       16:38:44
:This file note is attached to cli.1.rev.

cli.4                   860    rwed   Sunday       16:38:44
Trashcan.info           430    rwed   04-Mar-87    12:38:36

5 files — 1 directory — 147 blocks used
```

The first column of the output tells you the name of the file or the directory, just as it would have appeared in the DIR listing, except the DIR command sorts all files alphabetically and does not display the file names until it has finished sorting. The LIST command starts listing file names immediately.

The second column tells you the size of the file in bytes. If it is a directory rather than a file, then "Dir" appears in this column. You can use the CD command to move into that directory, or you could list the contents of that directory by simply specifying the complete path name (in this case, you would type LIST df0:trashcan).

The third column lists "rwed" for all of the files in this example. These are flags that can be set by you or by a program to provide information to other programs or users about the status of the file.

The "d" flag, the only one to which AmigaDOS pays any attention, indicates that this file can be deleted. You can protect your file by using the PROTECT command to clear the d flag; AmigaDOS will then be unable to delete or overwrite that file. A protected file will have "rwe" listed in this column. (AmigaDOS version 1.2 and earlier does nothing with the other three flags.)

The "r" flag indicates that the file is readable, while the "w" flag notes that the file can be written to; you can write over the file or append something to the end of it. The "e" flag indicates that the file is executable. It tells AmigaDOS and other programs that this file either contains a program image or other executable entity, or is a text file that is a script for some scripting language to follow, such as the EXECUTE command.

The fourth column lists the day or date the file was created, and the fifth column lists the time of creation. This information is helpful when you are revising a file, giving it a new name each time.

Some files, such as cli.1.rev in the example, have informational notes attached to them. These notes are listed on the line following the file information. Later, you will learn how to create filenotes.

Using LIST With Parameters

LIST is a versatile command. In addition to using it with no parameters to display the contents of the current directory, you can specify the

name of a directory or the name of a file, thereby getting a different kind of output. You can view its parameters by entering:

```
LIST ?
```

AmigaDOS responds with the following command template:

```
DIR,P = PAT/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K,SINCE/K,UPTO
/K,QUICK/S:
```

The keywords in this template indicate the following:

DIR lets you specify the directory you want to list. If you specify no directory name, the current directory is listed.

P = PAT indicates that either the keyword P or the keyword PAT can be accepted here. PAT tells LIST that the next word should be treated as a pattern against which the file names are to be matched; all file names in the specified directory that match this pattern are listed.

The following example lists all files that have the character sequence "la" anywhere in the file name:

```
LIST df0:c PAT #?la#? NODATES
```

Output:

```
Directory "df0:c" on Wednesday 04-Nov-87

Relabel                    860   rwed
Lab                         40   rwed
FailAt                    1072   rwed

3 files — 9 blocks used
```

The pattern specifier #?la#? can be translated as "any number of anything preceding or following 'la'."

KEYS lists the block numbers in the AmigaDOS filing system at which each file begins. Information such as this is important only to

those who are using sector editors to get directly to disk data. Here is a sample command:

```
LIST df0:utilities SINCE 06-nov-86 KEYS
```

Output:

```
Directory "df1:utilities" on Wednesday 04-Nov-87

.info              [ 1308]   35   rwed   06-Nov-86   12:35:37
Calculator.info    [ 1304]  218   rwed   06-Nov-86   11:49:08
Notepad.info       [ 1318]  302   rwed   06-Nov-86   11:49:16

3 files — 6 blocks used
```

DATES specifies the date and time of file creation. This is the default mode, but it is deleted when you select the QUICK option. Note that QUICK also deletes the information about the file size and the rwed status. Using QUICK and DATES, you can customize your listing output. Example:

```
LIST df0:utilities QUICK DATES
```

Output:

```
Directory "df0:utilities" on Wednesday 04-Nov-87

.info              06-Nov-86   12:35:37
Calculator         04-Nov-86   15:39:02
Notepad            04-Nov-86   15:39:20
Calculator.info    06-Nov-86   11:49:08
Notepad.info       06-Nov-86   11:49:16

5 files — 149 blocks used
```

NODATES produces a display with no date or time information. Example:

```
LIST df0:utilities S lat NODATES
```

Output:

```
Directory "df0:utilities" on Wednesday 04-Nov-87

Calculator              13240   rwed
Calculator.info           218   rwed

2 files — 31 blocks used
```

TO is used to redirect LIST to a specified output file. Example:

```
LIST TO RAM:mylist
```

Unlike redirection using the > character, which must be placed immediately following the command, TO can be placed anywhere on the command line. Instead of typing:

```
LIST >RAM:mylist df0:
```

you can type:

```
LIST df0: TO RAM:mylist
```

S means "substring." It lets you list all files that contain a particular string of characters. The following example lists all files that contain the character string "lat":

```
LIST QUICK S lat df0:utilities
```

Output:

```
Directory "df0:utilities" on Wednesday 04-Nov-87

Calculator

Calculator.info

2 files — 31 blocks used
```

SINCE will specify a date, in the format DD-MON-YY, and then list the files and directories created or last modified since (but not including) that date. Example:

```
LIST df0:Utilities SINCE 04-NOV-86
```

Output:

```
Directory "df0:utilities" on Wednesday 04-Nov-87

.info              35   rwed   06-Nov-86   12:35:37
Calculator      13240   rwed   04-Nov-86   15:39:02
Notepad         54676   rwed   04-Nov-86   15:39:20
Calculator.info   218   rwed   06-Nov-86   11:49:08
Notepad.info      302   rwed   06-Nov-86   11:49:16

5 files — 149 blocks used
```

Instead of specifying an actual numeric date, you can also specify days of the week by name or by the special keywords "today" or "yesterday," as shown in the examples below:

```
LIST SINCE today
LIST SINCE monday
LIST SINCE thursday
LIST SINCE yesterday
```

Although this is an extremely handy feature, be aware of its limitations. "LIST SINCE a week ago monday," for example, is not a valid parameter—at least not in the current version of AmigaDOS!

UPTO also specifies a date, in the format DD-MON-YY; it lists the files and directories created or last modified prior to and including that date. Example:

```
LIST df0:Utilities UPTO 06-Nov-86
```

Output:

```
Directory "df0:utilities" on Wednesday 04-Nov-87

Calculator      13240   rwed   04-Nov-86   15:39:02
Notepad         54676   rwed   04-Nov-86   15:39:20

2 files — 143 blocks used
```

QUICK lists only the names of the files and the total of all blocks (of disk storage) that are used by the selected files. (Each block is 512

bytes long.) An AmigaDOS diskette that stores 880K contains 1758 blocks in which data can be stored. When you specify QUICK, the LIST command shows how many blocks are in use by the files you selected. In the command template, the qualifier shown with QUICK is /S. This means that the QUICK flag setting simply tells LIST how to output something; it needs no other parameters. Example:

```
LIST QUICK df0:Utilities
```

Output:

```
Directory "df0:utilities" on Wednesday 04-Nov-87

.info
Calculator
Notepad
Calculator.info
Notepad.info

5 files — 149 blocks used
```

The INFO Command

Unlike the Workbench Info menu item, which gives you information about an individual file or directory, the CLI INFO command provides you with information about the filing system. For instance, INFO will tell you how much space you have on all disks known to AmigaDOS, how many blocks of memory your RAM disk is currently using. The INFO command takes no parameters.

Here is the typical output from an INFO command on a single-drive system with only the Workbench disk in the internal drive:

```
Mounted disks:
Unit Size     Used    Free     Full   Errs      Status        Name
DF0: 880K     1667      91      94%      0  Read/Write  Workbench 1.3
RAM: 12K        22       0     100%      0  Read/Write
```

Volumes available:

Workbench 1.3 [Mounted]

RAM Disk [Mounted]

The headings on each column of information listed in this output can be defined as follows:

Unit—is the device name of the unit whose info is being given. Floppy disks here are indicated as DF0:, DF1:, and so forth. The RAM disk is listed as RAM:. A hard disk might be listed as DH0:, DH1:, or JH0:, and so forth.

Size—is the total size of the disk or device. When a hard disk is partitioned into one or more segments, the INFO command will list the size of each partition here along with the Unit name under which each is mounted.

Used and Free—indicate the number of "blocks" of space either in use or free on this disk or device.

Errs—shows how many "soft" errors have occurred on this device since the last hard reset. A soft error is one that is corrected or overcome when the command reporting the error is retried by AmigaDOS. If you see a lot of soft errors on different disks, your disk drive may need to be serviced.

Sometimes a slight scratch or a bit of dust on your disk may cause the drive to read the data incorrectly. Such "hard" errors afford no recovery; in these cases, your data should be copied to a new disk.

Status—indicates Read/Write or Read Only. Setting the write-protect tab on your disk so that you can see through the slot will result in a Read Only status.

Name—shows the name that you have given to a disk volume (if any).

Volumes Available — provides a list of all disk volumes known to the system. If you open a file but then remove the disk from the drive, that disk's volume name will show up in the list, but no Status info will

be given for it. Only disks currently present (mounted) in your disk drives will be given Status info.

The ASSIGN Command

The ASSIGN command, when typed in alone, provides information about directories that have been assigned to certain "logical" names. These logical names act as both shortcuts and forwarding addresses. They are shortcuts because they let you access certain directories with a short, abbreviated word or sometimes merely a single letter. For example, to find the FONTS directory, you do not have to know its absolute location—MYWORKBENCH:Fonts; you need only use the logical name FONTS:.

Certain assignments are more than shortcuts or substitutes for absolute addresses. AmigaDOS provides special treatment for directories assigned to the names S:, SYS:, and C:. If AmigaDOS cannot find the command you want to perform in the current directory, it will automatically search the C: directory, and, not finding it there, the SYS: directory. AmigaDOS searches the S: directory for script files (used by the EXECUTE command) if it cannot find it in the current directory.

When AmigaDOS first starts up, it automatically makes certain assignments, called defaults.

Here is the output of the ASSIGN command, exactly as it appears when typed without parameters. For this example, I am using a two-drive system; I have the Workbench disk in drive 0, I have copied something from it into the RAM disk, and there is a disk named CLI.BOOK.WORK in drive 1:

```
Volumes:
CLI.BOOK.WORK [Mounted]
RAM Disk [Mounted]
Workbench 1.3 [Mounted]
```

```
Directories:
S           Workbench 1.3:s
L           Workbench 1.3:l
C           Workbench 1.3:c
FONTS       Workbench 1.3:fonts
DEVS        Workbench 1.3:devs
LIBS        Workbench 1.3:libs
SYS         Workbench 1.3:

Devices:
DF1   DF0   PRT PAR SER
RAW   CON   RAM
```

First, we see the list of volumes; all three volumes are currently mounted. Next are the system default ASSIGNs. These directories are:

S: AmigaDOS will search this directory to locate the startup-sequence file. It will also search the S: directory automatically when you perform an EXECUTE command. (But remember, it will search the current directory before it searches S:.)

L: contains device handlers and the Disk-Validator.

C: contains the majority of the AmigaDOS commands. AmigaDOS searches this directory automatically if it cannot find the command you have entered in the current directory.

DEVS: contains devices.

FONTS: contains fonts.

LIBS: contains libraries.

SYS: AmigaDOS will search this directory to locate commands if it cannot find them anywhere else; it is a last resort before reporting to you "Command Not Found."

In the default listing shown above, all of these special names are assigned to specific directories on the Workbench disk; you can use the ASSIGN command to assign them to other directories.

For example, if you wanted to use fonts other than those contained on your Workbench disk, you could use the ASSIGN command to assign FONTS: to another disk's directory where your font files are located. Then, when you use Notepad or a similar program, it will look there for fonts instead of in your Workbench directory.

Let's say you want to use the directory called NewFonts on your Whiz.Bang disk. You would type:

```
ASSIGN FONTS: Whiz.Bang:NewFonts
```

If you do not currently have this disk in a drive, AmigaDOS will ask you to insert the disk into any drive. (AmigaDOS will not make an assignment to a volume name that it cannot locate.) From that time on, until you make a different assignment, any new request for fonts will be taken from this new FONTS: directory.

The same applies to the C: (CLI commands) directory. Let's say you have two customized Workbench disks, and, using a single-drive system, you would like to remove WB-1 and insert WB-2. You would like the system to use the directories on WB-2 instead of continually asking you to reinsert WB-1 just so it can read the commands directory. With customized Workbench #1 in the internal disk drive, you can switch disks by typing:

```
ASSIGN ?
```

AmigaDOS responds with the command template:

```
NAME,DIR,LIST/S
```

Remove Workbench #1 and insert Workbench #2, then type:

```
C: df0:c
```

From now on AmigaDOS will search the C directory of the disk now in the internal drive. Only if it cannot find the command would it ask you for the original disk; it would then search the directory assigned to SYS: before giving up entirely with "Command Not Found."

NOTE: Be careful when you enter an ASSIGN command. Before you press Return, be sure that what you typed is what you want. If ASSIGN

can find the target directory, the assign happens exactly as you requested.

If you want to switch entirely to the new customized disk, insert a script file in the C: directory of the new disk that contains the ASSIGN commands listed below. You create a script file by using an editor, such as ED or MicroEMACS (on the Extras disk). (I keep such a file, named "thisdisk," on my own system.) Here is the file:

```
ASSIGN C:       df0:c
ASSIGN S:       df0:s
ASSIGN L:       df0:l
ASSIGN SYS:     df0:
ASSIGN LIBS:    df0:libs
ASSIGN DEVS:    df0:devs
ASSIGN FONTS:   df0:fonts
```

Here we are assuming that these directories are actually available on the new disk that you are assigning. If you have done the initial ASSIGN (that is, ASSIGN ?, followed by typing C: DF0:C), you can simply type:

```
EXECUTE C:thisdisk
```

If you have not done the initial ASSIGN, but have the file named "thisdisk" containing all of those ASSIGN statements in the C directory on your new disk called WB-2, you can also use the following command before removing WB-1 from the internal drive:

```
EXECUTE WB-2:C/thisdisk
```

AmigaDOS will ask you to insert the WB-2 disk, and will execute the ASSIGNs, making that the disk from which everything will be accessed. This allows easy swapping of customized Workbench disks in a single-drive system.

Using ASSIGN With Parameters

If you type:

```
ASSIGN ?
```

as noted above, you will get the command template for ASSIGN, which is:

```
NAME,DIR,LIST/S
```

NAME is a logical name that either AmigaDOS defines (for example, C: and DEVS:) or you define. DIR is an AmigaDOS path name that the operating system substitutes whenever you use the corresponding name.

The "/S" following the LIST parameter means that it is a "switch." In this case, adding the word LIST to ASSIGN means "perform the ASSIGN, and then show a list of everything that has been assigned." Actually this produces the same list you would normally see if you entered the ASSIGN command with no parameters at all.

Devices

The last thing shown in the ASSIGN command output is Devices. These are the devices with which AmigaDOS can communicate. To be able to work with AmigaDOS, devices must be mounted (some mount as a default when AmigaDOS starts up). Being mounted means simply that AmigaDOS knows how and where to find the device driver (in the DEVS: directory) for the device.

Serial, Parallel and Printer Devices

The device list shows devices named PAR, SER, and PRT. These are the parallel port (referenced as PAR:), the serial port (referenced as SER:), and the printer (referenced as PRT:), which may be connected to either the serial port or the parallel port, depending on your Preferences settings.

PAR: is a raw output device. Any data copied to PAR: is copied directly to the device connected to the parallel port, with no translation whatsoever.

SER: is also a raw input or output device. Again, whatever you send to SER: is sent directly to the device connected to the serial port.

PRT: is a device that controls rudimentary printer operations. When you set your Preferences, PRT: receives information from a program,

and then uses either SER: or PAR: as its output device. Commands sent to PRT: take the form of what might be called a "generic" printer-control command sequence. Every printer driver takes this generic control sequence and translates it into the sequence of characters that it takes to explicitly control the printer for which the driver was written. This means that an Amiga program needs to support only one set of printer codes, the generic printer sequences, in order to support any Preferences printer. Because the PRT: driver does the translation work, the software vendor need not provide support for every make and model of printer.

Disk Devices

The device list shown above contains disk drives DF0 and DF1. These are the devices that drive the floppy disks in the Amiga. Floppy-disk devices have read and write capabilities and can carry a filing system.

The RAM Disk

Notice that RAM is listed as a device. Like floppy-disk devices, it also has a filing system (can hold files and directories). The size of RAM: is not fixed, however; it can grow and shrink depending upon its contents. (The INFO command always shows it as full.)

Console Devices

There are two console devices in the ASSIGN list above, RAW and CON. CON: opens when a program wants a text-oriented window to which cursor-move commands and text-styling commands can be sent. To open a console window, you have to specify both the device type (CON:) and the size and placement of the window that AmigaDOS opens for you in response to a request for a CON: output or input. Here are two specifications that could be used when opening a console window:

```
CON:0/10/640/80/Sample

"CON:0/10/640/80/Multiple Word Title"
```

where CON: is the name of the device.

Note: You can substitute RAW: for CON: depending on the type of input/output you are expecting. RAW: is seldom used because in input mode, it transmits the actual numeric value of the downstroke and the upstroke of the keys and not the "normal" key values you might expect to see. RAW: is useful primarily through a programming language rather than from the CLI interface.

Interpreting the first example, we see that:

0 is used to represent the X coordinate of the position at which AmigaDOS should put the upper left-hand corner of the window that it opens. A value of 0 is the leftmost side of the screen.

10 is used to represent the Y coordinate of the position at which AmigaDOS should put the upper left-hand corner of the window. Beginning with 0 as the top of the screen, increasing Y values move the window farther down the screen.

640 is used to represent the width of the window.

80 is used to represent the height of the window.

When a console device opens, the open area of the window is treated as though it has discrete positions for the characters of text in the font you have selected in Preferences. The number of lines of characters that you will see in the window will be an even multiple of the text font size.

If you are using the normal system fonts, this number will be a multiple of eight lines. Ten lines are removed from the top of the window area for the title bar and the frame around the window, and one line is removed from the bottom of the window (again the frame line). Thus, if you open a window that is 80 lines high (as shown in the example), only eight lines of text will fit. (We get this by taking the number of lines, subtracting the borders, dividing by the lines per character, and then rounding off to the nearest whole lower multiple of the system font. Thus $80 - 11 = 69$; $69 \div 8 = 8.625$, or 8.)

Sample occupies the place reserved for the name that will appear in the window title bar. In the second example, where you have a multiple-word title, the entire specification for the console device should be placed in quotes.

Find a long text file and try the following two exercises to assure yourself that you can control the console size and title:

```
TYPE >CON:0/10/640/80/MyTitle your_textfile
```

```
TYPE >"CON:30/0/200/120/New Title" your_textfile
```

In these examples, when the TYPE command hits the end of a file, it closes the console window automatically. Because CON: knows how to write text into windows, the specification:

```
CON:x/y/width/height/name
```

is treated like the name of a file. You can substitute a console device specification anywhere that you would normally use a file name.

The NIL Device

Although it does not show up in the ASSIGN command output, NIL: is a perfectly legal device available for use in special situations. Use NIL: when you want to throw something away. When you redirect the output of a command to NIL:, it will not be printed or saved in any way. This comes in handy now and then, particularly during execution of script files.

NOTE: Before moving away from ASSIGN and on to the next command, I would urge you to be careful when choosing names for disks that you format. Do not choose names that are the same as any of those assigned or reserved by the system (i.e., command keywords, Workbench directory or file names, and so forth). ASSIGN lists things in the following sequence:

Volumes
Directories
Devices

This is the sequence in which AmigaDOS looks for those things in the system. For example, if you have a disk named "C," AmigaDOS would expect to find its commands in the root directory of that disk, because it looks at volumes first. Therefore, AmigaDOS would be unable to

find any of its commands. In another example, if you are using a disk named "SER," you will not be able to use the SER: (the serial device) until you remove the "SER" disk from the system.

The STATUS Command

STATUS gives you information about processes that have been started under control of the CLI; such procedures will show up in the status list. If you start operations from the Workbench, however, even though they are running as separate processes, they will not show up in the list.

From the CLI, type:

```
STATUS
```

You should get a listing such as:

```
Task 1: Loaded as command: df1:emacs
Task 2: Loaded as command: status
```

As you can see, I have one CLI-launched task, which is "emacs" from drive df1:, while in a second CLI, I have issued the command "status."

Task 1 corresponds to CLI number 1. When you start another CLI, its prompt will normally show a different number, for example 2.

The STATUS command also has parameters it can accept, although these are probably of interest only to the programmer and are not covered in detail here. (Consult the *AmigaDOS 1.2 Manual* for more information about STATUS parameters.)

The WHY Command

Use WHY after or within an EXECUTE script, or to find out why the last command failed.

If the command you perform sets a return code in the event of an error, the WHY command can access it and attempt to translate that code number into an understandable English translation. From that, you may be able to determine better what you did wrong.

If you type:

```
WHY ?
```

AmigaDOS responds with:

```
Type RETURN to know. . . :
```

If you type:

```
WHY
```

and press Return, AmigaDOS tries to translate for you. Here is an example. Type:

```
DIR xxx:
```

When the requester comes up and asks "Insert XXX into any drive," select CANCEL. The requester will disappear and the CLI will report:

```
Could not get information for XXX:
device (or volume) not mounted.
```

Now if you type:

```
WHY
```

and press Return, AmigaDOS responds:

```
Last command failed because device (or volume) not mounted.
```

Here the translation is the same as that reported by the command, but this is not always the case. Another common report from WHY is:

```
Last command did not set a return code.
```

This forces you to do a bit more work to figure out what you did wrong. AmigaDOS can translate only those errors that it can recognize. (See Appendix B for a list of the AmigaDOS errors.) Applications programs often generate their own error messages and may not provide any information for WHY to examine for its report.

The VERSION Command

Of interest primarily to programmers, the VERSION command provides a line of output that tells the programmer what version of the system the program is running under. If the programmer has been keeping track of changes to the operating system, he or she can make adjust-

ments to the program to ensure that it will function correctly under new versions of the operating system.

Here is the typical output of VERSION:

```
Kickstart version 33.180. Workbench version 33.47
```

The VERSION command does not accept any parameters.

The DATE Command

When used without parameters, The DATE command reports the date currently known to AmigaDOS. The date command's output with no parameters looks like this:

```
Sunday 15-Nov-87 20:12:00
```

DATE reports the current weekday name, then the day, month, and year, followed by the time in a 24-hour format as hours:minutes: seconds.It is important that you keep your Amiga's clock up to date because the date and time recorded for files are the actual date and time at which the file was last written. You should keep track of the file creation times so that if you have two versions of a file with the same file name, you can look at the creation times and see which one is the most current.

Note: If you make a copy of a file, its date will be when you created the *copy*, not the original file.

How to Set the Date

The DATE command can also set the date and time of the Amiga's internal clock. Its template is:

```
DATE, TIME, TO=VER/K
```

At the DATE parameter, you enter the current date in DD-MMM-YY format. For example, to set the date to August 1, 1988, you would enter:

```
DATE 01-aug-88
```

Note that leading zeros are significant when you set the date.

Set the current time with the TIME parameter. The Amiga's internal clock uses 24-hour notation; if you want to set the time to 6:35 PM, you would enter:

```
DATE 18:35
```

The TO parameter lets you set the date from a disk file.

NOTE: The DATE command works on the Amiga's internal clock. It does not affect the battery-backed clock/calendar that is standard on the Amiga 2000 and optional on the 500.

The TYPE Command

The ultimate informational command is TYPE, which lists the entire contents of a file. From the CLI, this command simply types the contents of the file into your CLI window. For example, with your Workbench disk in the internal drive, from a CLI issue the command:

```
TYPE df0:s/startup-sequence
```

You can use TYPE to display the contents of any file.

NOTE: If you use TYPE to display the contents of a file containing any non-ASCII characters, they may cause your screen to flash and strange things to appear. You should limit your use of TYPE to files that you know for certain contain only text material.

Using TYPE With Parameters

TYPE's command template is:

```
FROM/A, TO, OPT/K:
```

You can use TYPE as a kind of filter. It expects you to specify which file is to be typed, and it also allows you to specify where the output of TYPE should go.

The OPT keyword can accept one of two possible options:

H gives you a hexadecimal listing of a file instead of an ASCII listing.

N adds a sequence number to each line it types.

You may add line numbers to a copy of a file simply by issuing the following command:

```
TYPE FROM s/startup-sequence TO start.numbered OPT N
```

You can also employ TYPE as a cheap printer text-file dumper by simply redirecting its output to the printer. Example:

```
TYPE >PRT: mytextfile
```

You may find that you will begin using the TYPE command quite frequently, especially when you start working with script files (Chapter 8). TYPE is a great deal faster than a text editor or word processor for examining the contents of text files such as script files.

5 Modifying Files Using AmigaDOS Commands

This chapter covers two topics: modifying files and making new directories. Both are important functions that will increase your range of options as you become more familiar with your system.

Below is a list of commands—with a brief description of the operations they perform—you would use to make changes to files and to create new directories. We will examine each of these commands in detail and work through some examples to help you become familiar with using them.

Filenote annotating a file

MakeDir adding to the filing system tree

Rename changing a file's name or location in the tree

Relabel changing the name of a disk

Join making a new file out of one or more old files

Delete deleting a file or directory

Protect setting information bits (some of which "protect" the file)

Annotating a File

AmigaDOS allows you to use file names up to 30 characters long, giving you room to describe just what the file is and what it does. By the same token, if you always start your files from the CLI, typing in long names can become tiresome.

Assume that you are using only the CLI and have some files that you would like to describe more fully, but because you use them fairly

often, you do not want long names to type each time. Here are several options for handling the problem:

1. Keep the long, descriptive file name and suffer through extensive typing each time you want to use the file.

2. Copy the file, give the copy a shorter name, and use the short name to perform the function. This, however, takes up precious space on your disks.

3. Give the file a shorter name and add a filenote of up to 80 characters to fully describe the file's purpose. The filenote will appear when you examine the file using the Workbench INFO tool or the LIST command. This alternative offers the best of both worlds; it requires little disk space and lets you use a short file name for quick typing.

What should you put into a filenote? If you are using several versions of one program, include the creation date and perhaps a version number. Adding the former is a good idea because the date you see in the LIST command refers to the time you copied the file onto that disk, not the date the file was created. Therefore, you can wind up with a number of dates for the same file name on different disks. Without filenotes, you may find it hard to tell them apart.

The template for the FILENOTE command is:

```
FILE/A,COMMENT/A
```

The first argument AmigaDOS expects is a file name. As usual, this can be a complete path name. The second parameter you supply should be the comment you want attached to the file. You can include spaces in the comment if you enclose it in quotation marks.

The following example shows you how to attach a comment to one of the files in the Workbench fonts directory:

```
FILENOTE fonts:diamond/12 "This is a system font"
```

Now to see the filenote, simply use the LIST command. Try the following:

```
CD fonts:
CD diamond
LIST
```

AmigaDOS responds:

```
12 1948 23-Nov-85 18:07:11
: This is a system font
1 file — 5 blocks used
```

When you use the LIST command, any filenotes appear on the command line immediately following the corresponding file name.

Note that the AmigaDOS 1.2 COPY command copies a file, but not a filenote. AmigaDOS 1.3 copy has an option that lets you copy the filenote when you copy the file. See Appendix A.

Making New Directories

In Chapter 1, we used the CD command to move around in the directory tree on the Workbench disk. We also used CD without parameters to find our current location.

In this section you will learn how to create new branches on the directory tree using the MAKEDIR command. The following sequence creates several new directories. Use the DIR OPT A command to view these directories.

```
CD RAM:
MAKEDIR a1
CD a1
MAKEDIR b1
MAKEDIR b2
CD b2
MAKEDIR c1
CD c1
CD
```

AmigaDOS should respond:

```
RAM:a1/b2/c1
```

Entering:

```
CD /
```

results in:

```
RAM:a1/b2
```

Now enter:

```
CD :
```

To view what you have created, type:

```
DIR OPT A
```

And you should see:

```
a1 (dir)
  b2 (dir)
    c1 (dir)
  b1 (dir)
```

What good is this ability to create new directories? Let's say that a1 represents a book I am writing. The directories labeled b1 and b2 may represent the chapters, and the c1 directories will accommodate sections within each chapter. Within each directory I can then keep separate text files relating to that chapter or section.

Here is how the book you are holding was arranged on my disk:

```
a1 (dir)
  preface
  b2 (dir)
  introduction
c1 (dir)
  part1
  part2
  part3
```

and so on. You can expand your directories to any depth and breadth, and assign them any meaning you wish.

The reason many people like to use hierarchical filing systems such as this (so called because they are built in levels) is that organizing files this way leaves fewer names at each individual level, making it easier to find a particular file. (It can also be an easy way to lose files if you do not design your subdirectories and filing system well!)

Renaming a File

You use the RENAME command to change the name of a file or directory. For example, if you have a file whose name is too long or is not as descriptive as you might wish, you can change its name in the following way:

```
RENAME "Not Very Descriptive File Name" NewName
```

The template for RENAME is:

```
FROM/A,TO = AS/A:
```

The first parameter the command expects is a file name (or a complete path name to a file), and the second parameter is the new name (or a complete path name).

The TO = AS notation means that either TO or AS can be used as keywords.

Here are some examples of RENAME:

```
RENAME FROM oldname TO newname
RENAME TO differentname oldname
RENAME thisfile AS thatfile
```

The third example is my favorite way to express a RENAME command; it really tells you what is going on.

You can also use the RENAME command to move a file or a directory to a different part of the directory tree. Back to my book disk example:

Let's say that I saved a section called infostuff under Chapter 2, but that it actually belongs in Chapter 5. Here is what my directory tree (from a DIR OPT A command) might look like before I make the change:

```
Chapter2(dir)
  commands
  templates
  infostuff
Chapter3(dir)
Chapter5(dir)
  renaming
```

Issuing the command:

```
RENAME Chapter2/infostuff AS Chapter5/infostuff
```

will accomplish the move, and now the result of a DIR OPT A will be:

```
Chapter2(dir)
  commands
  templates
Chapter3(dir)
Chapter5(dir)
  infostuff
  renaming
```

When you use RENAME within a device, that is, when the source file is on the same device (df0:, dh0:, RAM:, or any other) as the destination directory, RENAME deletes the file from the original directory and puts it in the target directory, just as if the command were "MOVE"; it does not just copy the file.

Here is an example that you can try to see how the RENAME command works. This example introduces the COPY command (we will examine it more closely in the next chapter). I use COPY here to put a program into RAM so we can work from there and avoid altering

your disk. First, insert your Workbench disk into the internal drive. Now type:

```
MAKEDIR ram:f
MAKEDIR ram:f/f2
MAKEDIR ram:f/f2/f3
CD DF0:
COPY Clock ram:f/f2/f3
DIR ram: OPT A
```

The last entry prompts AmigaDOS to display:

```
f (dir)
  f2 (dir)
    f3 (dir)
      Clock
```

Continue by typing:

```
RENAME ram:f/f2/f3/Clock AS ram:f/f2/Clock
RENAME ram:f/f2/f3 AS ram:f3
RENAME ram:f/f2 ram:f2
```

To see the result of running the last three RENAME commands, type:

```
DIR ram: opt a
```

AmigaDOS replies:

```
f2 (dir)
  Clock
f3 (dir)
f (dir)
```

You cannot use the RENAME command to move a file or directory from one device to another. For example, the sequence:

```
RENAME df0:Clock AS RAM:Clock
```

is not legal because AmigaDOS sends the command to the device itself.

Moving things within a device is feasible because each device can maintain its own methods of storing and finding files and directories.

Shortcuts for RENAME (and other commands)

Thus far, we have specified a complete path name with each RENAME command, but as we saw earlier in the book, we can use shortcuts to refer to path names. For example, to move some files into a directory named RAM:f/f2/f3, I can shorten my typing by using the ASSIGN command, as:

```
ASSIGN D: RAM:f/f2/f3
```

Then the example above might have looked like this:

```
MAKEDIR ram:f
MAKEDIR ram:f/f2
MAKEDIR ram:f/f2/f3
CD DF0:
ASSIGN D: ram:f/f2/f3
COPY Clock TO D:
```

As a result of issuing the ASSIGN command, the f3(dir) has moved to a different level in the RAM: directory. When you define an object using ASSIGN, AmigaDOS puts a lock on it, preventing you from renaming it, moving it, or deleting it.

You can create a conflict in the system if you ASSIGN the same name to a file or directory that you have already used as a volume name (a "label") for a disk. When you perform a command, AmigaDOS tries to find a path to the named item by first looking in the list of disk volume labels. If it cannot find the name there, it will look at the list of ASSIGNs you have performed.

Harv Laser, chairman of American People Link's Amiga Zone, related the following story to me. It illustrates the types of problems you can run into if you are not aware of how ASSIGN works:

"I recently formatted a stack of disks, and named them in sequence A:, B:, C: and so forth. Later, while I had the disk named C: in a drive, I tried to run some AmigaDOS commands. I kept getting mes-

sages such as 'Unknown command DIR' and 'Unknown command INFO.' The name C: on the disk overrode the assignment I had made for the C: directory. AmigaDOS was searching the C: disk instead of the C: directory for its commands!"

To avoid such problems, when you name your disks, choose ones that are different from those names you have ASSIGNed.

If you find you need to delete an assignment that you have already made, simply ASSIGN the item to nothing, like this:

```
ASSIGN D:
```

This removes the lock and allows all operations to work again.

You can use the directory terms "/" (slash) and ":" (colon) as specifiers, instead of using full path names, to move items from one directory path to another. Each slash entered will move the item one level up from your current position. Entering a colon moves the item to the topmost level on the device. The following examples show the use of slash and colon for renaming:

```
MAKEDIR ram:f
MAKEDIR ram:f/f2
MAKEDIR ram:f/f2/f3
ASSIGN D: RAM:f/f2/f3
COPY df0:Clock D:
CD D:
RENAME Clock AS /Clock
DIR ram: OPT A
```

AmigaDOS responds as follows to the last command:

```
f (dir)
  f2 (dir)
    Clock
    f3 (dir)
```

Continue with:

```
RENAME /Clock :Clock
DIR ram: OPT A
```

You will then see:

```
f (dir)
Clock
  f2 (dir)
    f3 (dir)
```

The colon serves to move the clock to the topmost directory of the device, that is, RAM:.

Relabeling a Disk

The Workbench has a menu item called RENAME that allows you to change the name of any disk or file that you click on. Workbench can distinguish between disks and files; it tests each item before renaming it. You can use the CLI to change the name of a disk through the RELABEL command (do not use RELABEL, however, to change a file name).

You can change the name of a disk by referring to the drive's system name (such as df0: or df1:), or by using the current disk name (such as "Workbench 1.2" or "Myfiles"). If you forget the disk's name, simply use the INFO command in the Name field. The following example shows how you can perform the RELABEL command. With the Workbench disk in the internal drive, type:

```
INFO ?
```

The CLI will respond:

```
none:
```

This means that the command is already loaded into memory and ready to execute. Pop the Workbench disk out. . . insert the disk whose name is changing, and then press RETURN. The CLI responds by

listing the volume name of that disk. Suppose the name is "MyLists." To change the name to "MyLists.old," type the following:

```
RELABEL ?
```

The CLI responds first by asking that you reinsert the Workbench disk so it can load the RELABEL command. Then it displays the RELABEL command template:

```
DRIVE/A,NAME/A
```

The first thing it expects to see is the name of a drive and then the new name. You can specify a drive name as df0:, df1:, or df2:, and so forth, or as a volume name followed by a colon.

The name in the second parameter *cannot* have a colon following it; if it does, you will have great difficulty later in trying to specify path names for the files on that disk. The correct way to specify a RELABEL (assuming you have followed the instructions above and the command is now waiting for parameters) is:

```
MyLists: MyLists.old
```

or, if you have simply formulated it from the CLI command line directly:

```
RELABEL MyLists: MyLists.old
```

To repeat, the drive parameter must have a colon after it; the name parameter must not have a colon after it. Thus, RELABEL DF0: STUFF is right, but RELABEL DF0: STUFF: is wrong.

When you execute this command from Workbench, AmigaDOS relabels the disk and changes the icon name automatically. If you perform the command from the CLI, Workbench will not know you have modified the label unless you eject that disk from the drive and then pop it back in, allowing Workbench to reread the volume name on the disk. (This is a result of multitasking; the CLI and the Workbench are separate tasks, running individually and simultaneously.)

Joining Files Together

Sometimes I create text files in separate chunks to encapsulate separate ideas. When I am satisfied that all of the files are complete, and I want

to make a chapter or paper out of them, I can use the AmigaDOS JOIN command to link the files, instead of going into my word processor and adding each file to the end of the previous one. JOIN is much faster and does not limit me to the final size of a file that will fit in memory. (I can join files whose combined size is larger than the memory on the computer, whereas some word processors can edit only files that do not exceed the amount of available memory.)

Here is an example that demonstrates the JOIN command. Before trying it, put your Workbench disk in the internal drive. The command sequence uses the AmigaDOS redirection commands to create two different text files in RAM:. It then joins them to create yet another file, and types that file. (Your output may be different if you are using a later version of the Workbench disk than mine.)

```
DIR >ram:lib.dir libs:
LIST >ram:lib.list libs:
JOIN ram:lib.dir ram:lib.list AS ram:both.lists
TYPE ram:both.lists
```

The AmigaDOS output is:

```
diskfont.library icon.library
info.library mathieeedoubbas.library
mathtrans.library translator.library
Directory "libs:" on Wednesday 30-Dec-87
mathtrans.library 7428 rwed 23-Nov-85 18:02:51
icon.library 11880 rwed 23-Nov-85 18:02:57
translator.library 12920 rwed 23-Nov-85 18:03:02
info.library 16932 rwed 23-Nov-85 18:03:09
mathieeedoubbas.library 4472 rwed 23-Nov-85 18:03:13
diskfont.library 4268 rwed 23-Nov-85 18:03:19
6 files — 128 blocks used
```

The template for the JOIN command is:

```
,,,,,,,,,,,,,,,AS/A/K
```

The 15 commas indicate that you can use this command to join as many as 15 files at one time. The "AS" keyword is necessary for the command to function. It tells the system what name should be applied to the files you are joining. AmigaDOS does not allow the target name to be the same as any of the names of the files that you join.

How to Delete a File or Directory

DELETE is a useful command in that it lets you get rid of the files and directories you no longer need. It is also a dangerous one, however, because it does not give you a chance to change your mind. Once you have made your choices about what to delete and have pressed RETURN, your files are irrecoverable. Unlike the equivalent functions in MS-DOS or CP/M, AmigaDOS does not ask, "Are you sure?" It just obeys your command.

The command template for DELETE is:

```
,,,,,,,,,,ALL/S,Q = QUIET/S:
```

The set of commas in the template indicates that DELETE can accept up to 10 file specifications, each of which can include a wild card. AmigaDOS will attempt to delete files in sequence. If it finds that one of the files in the sequence does not exist, that a file is protected from deletion, or that a name represents a non-empty directory, DELETE will fail at that point and will not attempt to delete any more files in your request.

Normally, DELETE reports each file name that it completes, but if you do not wish to see the deletion reports, use the QUIET (or Q) keyword on the command line.

The ALL command deletes all files within a specified directory, and then deletes the directory itself. For example, if you have several files in RAM:stuff, such as "item1, item2, item3," and you type:

```
DELETE ram:stuff ALL
```

then DELETE will report:

```
ram:stuff/item1 . . . deleted
ram:stuff/item2 . . . deleted
ram:stuff/item3 . . . deleted
ram:stuff . . . deleted
```

Before we delete something from one of your disks, let's practice with the RAM: disk. Start by putting a couple of things into RAM:, and get rid of them from there. Start with your copy of Workbench in the internal disk drive.

```
CD df0:
COPY Clock ram:
MAKEDIR ram:nn
COPY Clock ram:nn
CD ram:
DIR OPT A
DELETE Clock
DELETE #?
```

AmigaDOS responds to the last command:

```
ram:nn Not Deleted — directory not empty
```

Notice that with that last DELETE command, I did not specify "ALL." If I had, the files in the directory and the directory itself would be gone. AmigaDOS's refusal to delete a non-empty directory is one you can take advantage of.

If you have a directory that contains many files, and you want to save just one or two and delete the others, you can create a new directory inside the current one. Use the RENAME command to move the files to be saved into that new directory, and then issue the DELETE command with a wildcard to get rid of everything else. (In the next section, which deals with the PROTECT command, you will find an alternative method for preventing deletion of a file or directory).

Let's make some room on a copy of the Workbench disk by deleting the printer driver files you are not likely to use. Here is what I did to

get rid of all files except the driver for my own printer, a Diablo-630. Be sure that you are using a copy of your Workbench disk and not the original. You are about to get rid of a few files and unless you have kept a backup, or have access to an UNDELETE command, the files will be gone forever. With a copy of Workbench in the internal disk drive, type:

```
CD df0:devs/printers
MAKEDIR save.these
RENAME diablo_630 AS save.these/diablo_630
DELETE #?
```

AmigaDOS now lists all the printer drivers deleted. It also reports:

```
save.these Not Deleted — directory not empty
```

Now to move the driver back to the printer directory and get rid of the temporary-storage directory, type:

```
RENAME save.these/diablo_630 AS diablo_630
DELETE save.these
```

Cleaning out unused files in this way makes room for other utilities. Yet another way to delete files is to use the DIR OPT I command, as shown in Chapter 1.

Setting Informational (Protection) Bits

In our discussion of the LIST command in Chapter 4, we mentioned protection bits, including the "rwed" produced in the LIST command output. Here we will see how they are used. There are four protection bits that you can set using the PROTECT command.

The command template for PROTECT is:

```
FILE, FLAGS/K
```

For the FILE parameter, enter the name of the file or directory to be

protected. For the FLAGS parameter, you provide a string that contains any combination of the characters r, w, e, and d (which stand for readable, writable, executable, and deletable). When you ask AmigaDOS to delete a file or directory that is delete-protected, it responds:

```
Not Deleted — file is protected from deletion
```

The alternative to using the DELETE command I mentioned earlier in my printer-driver example can be applied here. I can simply protect that file instead by typing:

```
CD df0:devs/printers
PROTECT diablo_630 rwe
DELETE #?
```

After reporting a number of messages about deleted files, AmigaDOS will reply:

```
diablo_630 Not Deleted — file is protected from deletion
```

The bits you set allow whatever action they stand for. For example, if you set the D (deletion) bit, it allows the file to be deleted. If you do *not* set the D bit, it does not allow the file to be deleted. The easiest way to reset all of the protection bits at the same time is to simply feed a null string to the command, such as:

```
PROTECT filename ""
```

or

```
PROTECT filename FLAGS ""
```

If you LIST a file that has been protected in this way, you will see "- - - -" in place of "rwed", which means "not readable, not writable, not executable, and not deletable." Because only the "d" flag has any

kind of effect, using the blank string is simply shorthand for resetting all of the bits.

Although using this shorthand method saves time, you may still want to specify flags individually so that when and if the operating system changes to make use of other bits contained in the directory and file-entry data, your commands will still function.

6 The Heart of the CLI

This chapter presents the system utility programs. These allow you to manipulate AmigaDOS files and disks, and to create new bootable disks and fill them with files.

The system utility commands (DISKCOPY, FORMAT, COPY, INSTALL, LOADWB, RUN, NEWCLI, ENDCLI, EXECUTE, SEARCH, SORT, ED, EDIT, DISKDOCTOR) are the CLI's core of power. I will discuss the most frequently used commands first, then those that are less popular.

Copying Disks

An important command for new users, DISKCOPY lets you make backup copies of AmigaDOS disks. Backing up disks may seem a tedious chore, but it is invaluable if you destroy a file or corrupt a disk.

From the Workbench you copy a disk by selecting its icon, then selecting Duplicate from the Workbench menu. Workbench in turn calls the DISKCOPY program from the System drawer. To copy a disk from the CLI, you can call DISKCOPY directly.

To call the program from the CLI, without entering its path name, you must change your CLI defaults to allow AmigaDOS to find commands in the System drawer. To find a command, the CLI searches the current directory first, then the C directory. To access DISKCOPY, type:

```
CD System
```

(Alternately, you could add the system directory to the AmigaDOS search path with the PATH command. See Chapter 7 for details.)

DISKCOPY copies the contents of one disk, track for track, onto another disk of the same size and type. In other words, you cannot

use DISKCOPY to copy from a 5¼-inch disk to a 3½-inch disk, or vice versa. Because the command works only when the number of tracks and sectors per track are identical, you cannot set up an 880K hard-disk partition and use DISKCOPY to copy a disk from or to that partition. The floppy disk and the hard-disk partition have a different number of tracks and a different number of sectors per track.

DISKCOPY's command template is:

```
From/A,To/A/K,Name/K:
```

The normal sequence of parameters is the device name from which you will copy (the source disk drive), followed by the device name for the destination drive, followed by the name, if any, you wish to give the destination disk. From/A means that you must provide the device name from which you are copying. The word "From" is optional. To/A/K means you must specify a destination disk and include the word "To." Name/K indicates you may provide a destination disk, if you include the word "Name."

To copy from the internal drive to an external drive, type:

```
DISKCOPY From df0: To df1:
```

or you can abbreviate it as:

```
DISKCOPY df0: To df1:
```

because DISKCOPY expects the source disk to be specified first and the destination disk second.

When you run DISKCOPY, it asks you to insert the source disk into the source drive and the destination disk into the destination drive. You need not format your destination disk before copying; just pop a brand-new disk into the drive. DISKCOPY formats each track before writing to it. For safety's sake, you should write-protect your source disk before copying.

If you are copying a disk on a single-drive system, you can specify the same drive as the source and destination drive. For example:

```
DISKCOPY From df0: To df0:
```

The program will use the computer's memory to hold the data, and

will ask you to insert the source disk and then the destination disk alternately until DISKCOPY finishes.

If you specify the parameters as shown above, DISKCOPY will give the copy the same name as the source disk. If you want DISKCOPY to give the copy a new name, you must instruct it with the Name parameter. All of the following are legal formats for copying from df0: to df1:

```
DISKCOPY From df0: To df1:
DISKCOPY To df0: df1:
DISKCOPY From df0: To df1: Name "My New Name"
DISKCOPY df0: To df1: "My New Name"
```

DISKCOPY cannot make copies of copy-protected disks. If DISKCOPY fails, it may be that the destination disk is write-protected or that the source or destination disk is damaged. The error message that DISKCOPY issues will help you track down the problem.

Formatting a Disk

If you want to use a disk as a data disk to which you save and copy occasional files, you must prepare it with the FORMAT command. FORMAT can also format hard-drive partitions. Workbench calls FORMAT from the System drawer when you use the Initialize menu item. To access the command, make the System directory the current directory by using the CD command. FORMAT is somewhat fussy with its command parameters, as the command template shows:

```
FORMAT Drive <disk> Name <name> [NoIcons]
```

The words "Drive" and "Name" *must* appear on the program's command line. To initialize a disk called MyDisk inserted in df0:, type:

```
FORMAT Drive df0: Name MyDisk
```

FORMAT automatically creates a Trashcan directory and icon on the newly formatted disk. If you do not want a Trashcan, you must use the NoIcons modifier:

```
FORMAT Drive df0: Name MyDisk NoIcons
```

When FORMAT starts, it instructs you to put a disk into the selected drive, then to press RETURN. FORMAT *will destroy all the data on the disk*, so be sure that you have the correct disk before you press RETURN.

FORMAT will verify each track by rereading it after writing to it. This can give you a bit more confidence about a disk's ability to hold data.

Copying Individual Files

To copy individual files or directories (and all of the files they contain) instead of an entire disk, use the COPY command. Note that the destination disk must already be formated. The command template for COPY is:

```
From,To/A,All/S,Quiet
```

If you do not specify a source directory or file name in the optional From position, COPY assumes that you wish to copy from the current directory. If you name a directory as a source, then COPY will copy everything within the directory to the destination directory. The destination must be a directory because the program assumes you will be moving multiple files. You must indicate an existing directory as the destination, because COPY will not create one.

For example, on a two-disk system, you insert a formatted disk into the external drive (df1: or df2:, depending on your system), and type:

```
CD df0:
COPY To df1:
```

AmigaDOS will copy all the files from the source directory (the root directory of df0:) to the destination directory (the root directory of df1:). The cloned files in df1:'s root will have the same names as the original files in the source directory. Only files within the directory are copied; the program ignores nested directories and their contents.

If you wish to copy nested directories and their contents, include the parameter All:

```
COPY <source> TO <destination> All
```

To make a backup copy of a disk with COPY, type:

```
COPY df0: df1: All
```

Using COPY to duplicate an entire disk is less efficient than using DISKCOPY, which works on tracks sequentially, stepping through each track until it finishes. With COPY, however, the system follows the individual files that are stored randomly on the disk. Because the disk head must move more often and skip about, the COPY method takes longer.

If you wish to copy a file into a directory, the file retains the name it had on the source disk. For example:

```
COPY df0:CLOCK To RAM:
```

creates a copy of CLOCK in the root directory of the RAM: disk, while

```
COPY df0:CLOCK To RAM:newclock
```

copies the same file to RAM:, but renames it newclock in the destination directory.

Be careful: If you attempt to copy a file that has the same name as a file currently in the destination area, COPY will replace the original file. COPY will not warn you if it is about to overwrite a file and offers no way to reverse your actions. You can escape, however, while copying a directory or a disk with CTRL-C.

To keep you informed, COPY writes messages to the CLI each time it completes a duplication, such as File1. . .copied, and so on. To terminate the messages, add the switch Quiet, with no spaces, as in:

```
COPY df0:mycommands RAM: All Quiet
```

Copying from the Keyboard

A simple method of loading text into a file is to use the COPY command to connect keyboard (console) input to a file. You refer to the console with the * character (note that this works both from the CLI and from the AmigaDOS 1.3 SHELL). To copy keyboard input to a RAM-disk file called Test, type:

```
COPY * To RAM:Test
```

From now on AmigaDOS directs everything that you type into the file as well as displaying it on the screen. The CLI will not respond to your keystrokes until you finish running the COPY program.

To stop copying input into the file, you must issue an end-of-file character; press the Ctrl key and the Backslash (\) key simultaneously. Hereafter, I indicate the end-of-file key combination as Ctrl-\. Try typing the following:

```
COPY * To RAM:Test
This is a quick way to create a text file.
CTRL-\
```

To view your new file type:

```
TYPE RAM:Test
```

AmigaDOS responds:

```
This is a quick way to create a text file.
```

Printing a File with COPY

A versatile command, COPY not only duplicates and creates files, but also lets you print them as well. To copy your test file to the printer (PRT: device) type:

```
COPY RAM:Test To PRT:
```

AmigaDOS sends the contents of the source file directly to the printer.

The INSTALL Command

To make a bootable disk that satisfies the system's request for Workbench, you must run the INSTALL command. The command template for INSTALL is:

```
Drive/A:
```

The word "Drive" is optional. You can install a formatted disk in df0:, df1:, df2:, and df3:. To install a disk in df1:, for example, insert the disk in df1: and type:

```
INSTALL df1:
```

Be careful when installing disks on a single-drive system. If you simply type INSTALL df0:, AmigaDOS will immediately install your Workbench disk. Instead, use the command template and disk-swap method discussed in Chapter 2.

When the disk-drive light goes out, the installation is complete and you can remove the disk. Now you can reset the machine. If the disk was entirely blank (freshly formatted) when you ran INSTALL, you should see a 60-column CLI after you reboot.

Never run INSTALL on commercial software. Manufacturers frequently use the boot blocks, the areas of the disk to which INSTALL writes, to copy-protect a program. Running INSTALL on one of these disks could render the disk unusable.

Creating a Minimally Configured Workbench Disk

You now have a bootable disk with a CLI on it, but no C directory of commands. You need to copy some files onto this new disk to make it useful. To begin, insert your work copy of Workbench, reset the system, and bring up a CLI. Type the following to copy a set of commands and files to the RAM disk:

```
MAKEDIR RAM:devs
MAKEDIR RAM:c
MAKEDIR RAM:libs
MAKEDIR RAM:L
MAKEDIR RAM:S
COPY CLI To RAM:
COPY CLI.info To RAM:
COPY devs:system-configuration To RAM:devs
COPY c:CD To RAM:c
COPY c:info To RAM:c
COPY c:ASSIGN To RAM:c
COPY c:PATH To RAM:c
COPY c:COPY To RAM:c
COPY c:DIR To RAM:c
COPY c:RUN To RAM:c
```

```
COPY c:NEWCLI To RAM:c
COPY c:ENDCLI To RAM:c
COPY c:LOADWB To RAM:c
COPY L:ram-handler To RAM:L
COPY libs:icon.library To RAM:libs
COPY libs:info.library To RAM:libs
COPY ?
```

Remove the Workbench disk from df0: and insert your newly formatted, newly installed disk. To COPY all the files and directories from RAM to the disk, type:

```
RAM: df0: All
```

When the command finishes, reset the system again. You can now call the INFO, CD, COPY, and DIR commands from the new disk. You can also use the RAM: device, because you copied the ram-handler to the disk.

When you called Preferences from your original disk, it created a system-configuration file of all the settings you made (80-column CLI, screen colors, custom mouse pointer, and so forth). When you copy the system-configuration file, you transfer these specifications to your new disk.

The LOADWB command and the two libraries allow Workbench to work. The NEWCLI and ENDCLI programs allow you to open and close CLI windows. RUN lets you use a single CLI to start multiple programs. EXECUTE allows you to create a startup sequence to go along with this new disk.

The minimally configured disk you created provides only basic capabilities. I will leave it up to you to determine whether this is enough or if you need more files from the original Workbench.

Starting the Workbench

When you boot with the minimally configured disk, the system opens an AmigaDOS window. To then enter the Workbench environment,

you enter the LOADWB command. Reboot the machine with your new disk. At the CLI prompt type:

```
LOADWB
```

Workbench loads and starts to run. Shrink the CLI window with the sizing gadget. Workbench will appear; the CLI window is a window on Workbench.

To Run a Program

The RUN command lets you load and start multiple programs in a single CLI. For example, with a standard Workbench disk in df0: type:

```
RUN demos/LINES
```

The CLI prompt returns almost immediately, and the Lines program runs. Now type:

```
RUN CLOCK
```

CLOCK starts running and again the CLI prompt returns. While the programs continue to run, the CLI will continue to accept new commands. If you had typed:

```
demos/LINES
```

instead of:

```
RUN demos/LINES
```

the LINES program would have monopolized the CLI and the CLI would not have executed the CLOCK command until the first program, LINES, was finished.

Programs have a standard input source and a standard output area. When you start a program directly from a CLI (and when its window is active), the program takes over the CLI's window for its input and output. The CLI does not interpret succeeding keystrokes, it just passes the keyboard input directly to the program. Because the CLI is dormant while the program runs, it will not accept any input or provide any output.

If you type RUN and the program's name instead of only the

program's name, AmigaDOS creates a separate process for the program. The CLI and the program share the same output device; both send output to the console window. Because you used RUN, however, the CLI remains active and will accept commands while the other program is running. Because the CLI does not pass keyboard input to the program you started with RUN, the program has no standard input. Use RUN for programs that do not need external input or that create their own input windows. If a program needs to receive both typed input and to provide output, you will have to start a new CLI.

Starting a New CLI

If you want to open additional CLIs from an existing CLI, type:

```
NEWCLI
```

The command will open a new CLI window in the default width and position on the Workbench screen. The NEWCLI program can also accept parameters. Its command template is:

```
Window, From:
```

If you specify a file name in the From parameter, the NEWCLI command will read and execute a script file with that name. With From, you can start any script file you could run with EXECUTE. The EXECUTE command, however, must be present in the C: directory for this feature to operate.

To demonstrate this feature, type the following script file:

```
CD df0:
COPY * To RAM:junk
DATE >RAM:todayis
ECHO "Today's Date is: "
TYPE RAM:todayis
df0:CLOCK
CTRL-\
```

Now type:

```
NEWCLI From RAM:junk
```

A second CLI with the default title New CLI will open and type the current date. Because the CLI is running a program, the cursor will be on a line by itself, without a prompt. The prompt will return when the program exits.

The CLOCK program will be running. If you close the clock, the new CLI will show a prompt character string.

NEWCLI's Window parameter lets you control the position, size, title, and type of window that is opened. Normally, you will want to open a CON: window, which looks like a CLI console. See Chapters 4 and 9 for more information about CON: windows.

A sample Window specification string is:

```
"CON:10/20/250/120/My New Window"
```

The first item, CON:, specifies a console window. The next two items are the X (10) and Y (20) pixel positions (relative to the screen's upper-left corner) at which the NEWCLI command should open the window on the Workbench screen. The next two items are the window's Width (250) and Height (120) in pixels. The last item is the title you wish to give the window.

Continuing from the previous example, type:

```
NEWCLI FROM RAM:junk WINDOW "CON:10/20/250/120/My New Window"
```

While the date will be displayed and the clock will run, the CLI window's size, shape, and title will be different from those in the previous example.

Closing a CLI

To close a CLI window, type:

```
ENDCLI
```

If you launched a satellite task with RUN within the CLI you are trying

to close, the CLI will not shut down until the other task finishes or you terminate it. (For more information, see RUNBACK in Chapter 11.)

EXECUTE and Script Files

The EXECUTE command allows you to create and run script files. Unlike other programs, script files will not run if you type only their names. You must tell AmigaDOS explicitly that a file is a script file by calling the EXECUTE function. If you try to run a script file without specifying EXECUTE, AmigaDOS will tell you "not an object module." See Chapter 8 for a thorough discussion of this complex and powerful command.

Searching for Text Strings

The SEARCH command looks through files to find an indicated character string. If SEARCH finds the string, AmigaDOS reports the line number of each occurrence. SEARCH examines only the first 255 characters of a line, then looks for an end-of-line indicator, and resumes the search on the next line. The command template is:

From,Search/A,All/S:

From, the first parameter that SEARCH expects, is the file (or complete path name) specification. This can be a pattern, file name, or directory name. If you specify a directory in the From position, SEARCH will look at all the files in that directory. If you specify All as a keyword, then SEARCH will look in all subdirectories as well. To search every file in every directory of a disk, specify the name of the disk or the drive it is in.

The parameter associated with the Search keyword is the string that you want to locate. SEARCH is not case-sensitive; you can specify a string and SEARCH will report all instances, whether they are in upper- or lowercase. You cannot substitute wildcards for the Search string.

Sorting Text Files

The SORT command reorders the lines of a text file into increasing alphabetical order, based on the characters' ASCII (American Standard Code for Information Interchange) values. Because computers under-

stand only numbers, not letters and special characters, each character on the keyboard has a standard numeric value—or ASCII value. Internally, the computer manipulates the character's ASCII value, not the character itself. For a table of ASCII values, see Appendix A of *The Amiga Basic Manual*. Like SEARCH, SORT is not case-sensitive.

The command template for SORT is:

```
From/A,To/A,Colstart/K
```

With the parameters, you specify the file that SORT is to take its input From and the file that SORT should direct its output To. As an option, you can specify the number of the column on which you wish to index the sort, Colstart.

Use COPY to create an input file to sort. Type:

```
COPY * To RAM:junk
MakeDir
Diskcopy
FORMAT
INSTALL
Run
cd
Search
Protect
Ctrl-\
```

Now, enter the following to sort the file and display the result:

```
SORT From RAM:junk To RAM:junk.sorted
TYPE RAM:junk.sorted
cd
Diskcopy
FORMAT
INSTALL
MakeDir
Protect
Run
Search
```

Because SORT orders characters based on their ASCII values, sort data can include non-alphanumeric characters.

To specify the column on which you wish to index your sort, use the Colstart variable. For example:

```
SORT From RAM:junk To RAM:junk2 Colstart 2
```

To see the result, enter:

```
TYPE RAM:junk2
MakeDir
cd
Search
Diskcopy
INSTALL
FORMAT
Protect
Run
```

A Pair of Text Editors

AmigaDOS offers two text editors—ED, a full-screen editor, and EDIT, a line editor. Text editors let you create script files, program listings, and other unformatted document files. You can also quickly check and alter the contents of a data file.

ED uses the no-frills approach, offering only the essential commands. You can move the cursor; insert and delete characters; search backward and forward for a character string; move, delete, and write blocks of text; modify a line of text and undo the changes; save the file; exit; and quit without saving the file. With only these functions, you can create quite lengthy text files.

Call the editor by typing:

```
ED RAM:ed.test
```

ED opens a window and creates a new file, in this case ed.test. See Appendix D for a list of ED commands. Remember to press RETURN after each one. When searching and replacing, ED is not case-sensitive.

For ordinary operations such as creating script files, EDIT is a poor

substitute for a page-oriented editor. If you absolutely must edit binary information, refer to *The AmigaDOS Manual*, 2nd ed., for a thorough explanation. If you do not need a line editor, I suggest you delete EDIT from your working copy of Workbench.

Doctoring a Disk

If you accidentally corrupt a disk, you can recover its data with DISKDOCTOR. AmigaDOS will alert you to the problem with the message "cannot validate this disk." To run DISKDOCTOR, put your damaged disk in df1: and type:

```
DISKDOCTOR df1:
```

If you own a single-drive system *be sure* to use the command template and swap method!

When DISKDOCTOR finishes examining the disk, it will instruct you to copy your files onto a new disk and to reformat the damaged disk. Put the damaged disk in df0: and a new disk in df1:, then type:

```
COPY df0: To df1:
```

Single-drive owners should first copy the contents of the damaged disk to RAM: (to activate the RAM disk, type DIR RAM:), then copy it from RAM: to the new disk.

Even if the directory area of a disk is damaged, the directory and file structures of an AmigaDOS disk are organized so that you can recover most files. DISKDOCTOR looks at all of a disk's files and tries to reconstruct them. Because DISKDOCTOR modifies the disk to let you copy the files to a new disk, you must reformat the disk before using it again. You must use COPY to transfer the reconstructed files. Because the disk is corrupted, DISKCOPY will not function correctly. In fact, because DISKDOCTOR has modified the disk so thoroughly, you cannot use any other commands (CD, DIR, LIST, and so on) on the disk.

For more information on a public-domain program—DISKSALV—that doesn't modify a corrupted disk, see Chapter 11.

7 Configuring AmigaDOS

This chapter details the commands that let you change the hardware or software configuration of your system. The commands are covered in order of complexity, beginning with the simplest.

AmigaDOS supplies many commands that let you change how your Amiga system operates. These commands affect how AmigaDOS works and how it interfaces with hardware peripherals. The commands covered here are SETDATE, PATH, PROMPT, ADDBUFFERS, BINDDRIVERS, MOUNT, DISKCHANGE, CHANGETASKPRI, and STACK.

The SETDATE Command

When you use the COPY command to copy a file, the date associated with the new file is the date it was copied, not the date the original file was created. You can change the date on a file or a directory with the SETDATE command. Its command template is:

```
FILE/A,DATE/A,TIME:
```

The template indicates that SETDATE expects both a file name (the file whose date stamp you want to change) and a date value such as an actual date, day name, or descriptor like TOMORROW or YES-

TERDAY. The command also accepts an optional time value. Examples of SETDATE are:

```
SETDATE myfile TOMORROW
SETDATE myfile FRIDAY 3:00
SETDATE myfile 17-Jan-88 12:15
SETDATE DATE TUESDAY TIME 11:07 myfile
SETDATE DATE 1-Jan-88 myfile 7:45
SETDATE myfile TODAY 4:27
```

AmigaDOS interprets words like TODAY and TUESDAY and converts them into its standard day-month-year format.

Setting the Search Path

When you enter a command in the CLI, AmigaDOS must find the command and load it into memory before the command can be executed. AmigaDOS searches not only the current directory to try to locate the command, but also searches the C: directory. Thus, the current directory and the C: directory comprise the AmigaDOS search path. You can add other locations to the AmigaDOS seach path by using the PATH command.

When you set a path, AmigaDOS will still search the current directory first, but instead of then searching the C: directory if the command is not in the current directory, AmigaDOS next searches the places you specify with the PATH command. Only after searching your paths without success does AmigaDOS search the C: directory. In other words, AmigaDOS will continue searching for the command until it finds it or until it runs out of search locations. Only in the last case will you get the dread "file not found" message.

Searching a large path may seem like a long process, but it is not. AmigaDOS can search quite a long path in much less time than it takes to read this sentence. So do not worry about creating long search paths—I doubt you will notice a slowdown as AmigaDOS performs its search.

You use the PATH command without parameters to display the search path that is currently in effect. If you have not changed the path

since booting, or if the path isn't changed in the startup-sequence, and you enter:

```
PATH
```

AmigaDOS will respond:

```
Current directory
C:
```

Now, try adding a couple of locations to the search path with PATH ADD. The names after the ADD parameter are directory names that are added to the path. Enter the following:

```
PATH ADD fonts: devs:
PATH
```

AmigaDOS responds:

```
Current directory
Workbench 1.3:fonts
Workbench 1.3:devs
C:
```

Whenever you add an item to the path, it goes at the end of the list, just ahead of C:. This is the order the system uses in its searches. The path list always begins with the current directory and always ends with the C: directory. Anything you add to the path shows up between these two locations.

In the startup-sequence file that is executed by the standard Workbench 1.3 disk, the system adds sys:system and sys:utilities to the path. Thus, your initial path may vary, depending upon the disk you use to boot your system.

You can add up to ten directories to the search path with a single PATH command. The directories are added in the sequence you specified on the command line.

Note that each of the directories you specify in the PATH command

must exist and be accessible to AmigaDOS. The CLI "puts a lock" on each specified directory by reading it and remembering where it is located. If you add a directory that is not currently on any of the disks in your drives, the CLI will ask you to insert the disk that contains the added directory. Subsequently, the CLI may prompt you to insert the disk during a path search.

You can remove all the directories you have added to the search path by issuing the PATH RESET command. For example, if your current path is:

```
Current directory
Workbench1.3:system
Workbench1.3:utilities
C:
```

and you enter:

```
PATH RESET
PATH
```

AmigaDOS responds:

```
Current directory
C:
```

There is no way to remove only one directory or to shuffle the path list. To change the path you must issue PATH RESET, then define a new path. The placement of the current directory and the C: directory in the path list cannot be changed.

Each CLI has its own search path, so if you open up multiple CLIs, each CLI can have a different path to search for commands. If you use one CLI to create another CLI, using the NEWCLI command, the new CLI will inherit the search path from the current CLI.

PATH Summary

When you give a command, AmigaDOS looks in the current directory to see if a command by that name is present in that directory. If the

command cannot be found there, AmigaDOS looks at each of the directories that you have added to the command search path using the PATH command. If the search is still unsuccessful, AmigaDOS searches the C: directory. The system always looks in the current directory first, and in the C: directory last. At any time, you can reset the path and get back to the default of current directory and C:.

Changing The Prompt

Normally, when you open a CLI, you will see the following:

```
1> _
```

This is called the prompt, and it is just something the CLI puts there to tell you that it is waiting for you to enter a command. Some computers make the cursor flash on and off; the Amiga CLI uses the prompt.

The "1" in the example above indicates which CLI process number is running the CLI. Because the Amiga can run several jobs at one time, it is helpful to be able to identify the CLI by the prompt that it presents.

When you ask for the PROMPT command template, AmigaDOS responds:

```
PROMPT:
```

The command is asking for the string that it should display while waiting for input. In response, you could enter a name as a prompt, such as:

```
Rob>
```

or even try something long and silly, such as:

```
Command Me, Oh Master -->
```

You now have a customized prompt.

Notice that if you enter a blank space after the PROMPT request, you get a forward arrow (>) as the prompt. AmigaDOS will supply you with a default prompt if you enter nothing at all. The only way to get a blank prompt is to enter:

```
PROMPT " "
```

The quotes around the blank character make it an entity that AmigaDOS must deal with.

If you want the CLI process number to appear within the prompt string, you have to provide the special character combination "%N," meaning "in this position, display the process number." Here is an example. If you enter:

```
PROMPT "Rob's CLI %N>"
```

your new prompt will be:

```
Rob's CLI 1>
```

The number will change whenever you open a new CLI.

An addition to AmigaDOS 1.3 is the ability to display the current directory as the prompt. To print the current directory in the prompt, you use the "%S" character sequence. For instance, if from the root directory of Workbench 1.3 you enter:

```
CD fonts
PROMPT %S>
```

your prompt will now be:

```
Workbench1.3/fonts>
```

If you now enter:

```
cd ruby
```

your prompt changes to:

```
Workbench1.3/fonts/ruby>
```

The %S option can be very useful, but it can result also in some absurdly long prompts.

Faster Floppy Drives

The ADDBUFFERS command can make certain AmigaDOS operations faster by giving the disk operating system more working room. Normally, when you ask AmigaDOS to run a program (either by clicking an icon or by running it from the CLI), the program and data from the disk

are read from the disk one sector at a time (a sector holds 512 bytes of information). Each sector is loaded into memory through an input buffer that AmigaDOS maintains for each disk drive. Once a sector is moved from the buffer into main memory, the next sector of data is loaded into the buffer.

The upshot of this is that AmigaDOS does not "remember" the data it has recently read from disk. If, for example, you perform the DIR command twice in a row, AmigaDOS does not remember the information that it displayed the first time; it must perform the command the second time exactly the same way it performed it the first time. You can improve AmigaDOS' memory with ADDBUFFERS, which lets you give AmigaDOS more working space by providing a place for AmigaDOS to store disk sectors in memory. If you provide enough buffers, AmigaDOS could store entire files in memory. Thus, if you perform multiple operations on the same file, AmigaDOS searches its memory copy of the disk, and if the file is in the buffer, AmigaDOS performs its operation in memory rather than actually accessing the disk. Because working within memory is much faster than accessing the disk, using ADDBUFFERS can result in a significant improvement in speed.

To use ADDBUFFERS, you enter the command, the drive you want to add the memory to, and the number of 512-byte buffers that you wish to reserve for the drive. For example, to add a 10K buffer for use with your internal drive, you would enter:

```
ADDBUFFERS df0: 20
```

There are drawbacks to using ADDBUFFERS. Once given to the system, the memory is reserved by AmigaDOS until you reboot the system—there is no corresponding REMOVEBUFFERS command available. Another drawback is that ADDBUFFERS takes memory from the custom chips—the same memory used to store graphics and sound

data. Even if your system contains expansion RAM, ADDBUFFERS will not use it.

Configuring Expansion Hardware

Much of the expansion hardware available for your Amiga, such as hard-disk drives and memory-expansion cards, contains special auto-configuration circuitry that indicates what type of hardware the device is and how much memory space the device needs to operate. Once the device has identified itself and specified its requirements to the Amiga, the system assigns the device a place in the Amiga memory map. A device handler will then normally make all accesses to this area of memory.

Auto-configuration circuitry (auto-config, for short) eliminates the need for user-selectable configuration switches and enables devices from different manufacturers to operate without conflict.

Even though auto-config hardware can configure itself into the memory space of your Amiga during power-up, many devices require additional software called device drivers (or device handlers) that let the devices communicate with the Amiga system through its memory space. Device drivers are normally supplied by the maker of the hardware expansion device. To get your Amiga to recognize a device driver and to connect a driver with the correct device on the auto-config list, you must run the BINDDRIVERS command.

BINDDRIVERS needs no parameters. To run it, you simply enter:

```
BINDDRIVERS
```

Now, if you have added an auto-config device to your system that requires a software driver (a hard-drive controller, for instance) and you have copied the driver from the disk supplied by the device manufacturer into the Expansion drawer of your boot disk, BINDDRIVERS will add the device and its controlling software to your system. BINDDRIVERS is normally the first program you run in your startup-sequence.

Mounting Hardware Devices

Some types of expansion devices, such as external floppy disk drives, can be driven directly from existing system hardware. For example, a

Commodore 5¼-inch disk drive connects directly with the external disk-drive port on the Amiga. The problem with using the 5¼-inch drive, however, is that its configuration (the size of the disk and the number of tracks and sectors it contains) differs from the configuration of the standard 3½-inch drive. You use the MOUNT command to tell the system about the existence and characteristics of the drive so that AmigaDOS can access it correctly.

The format of the MOUNT command is:

```
MOUNT device_name
```

Device_name is a label that matches an entry in the text file DEVS:Mountlist. Entries in the Mountlist file list the device name, its device driver or handler, and its characteristics. For example, here is the Mountlist entry that would let AmigaDOS access a 5¼-inch disk drive as the first external disk drive.

```
df1:    Device = trackdisk.device
        Unit = 1
        Flags = 1
        Surfaces = 2
        BlocksPerTrack = 11
        Reserved = 2
        Interleave = 0
        LowCyl = 0; HighCyl = 39
        Buffers = 5
        BufMemType = 3
#
```

The pound (#) sign is not a typographical error. It signifies the end of a particular Mountlist. If a hardware manufacturer sells you a device that requires a Mountlist, the manufacturer normally provides the information for the Mountlist.

The entries in the Mountlist are things that the particular device driver needs to know in order to use the device. The Device entry specifies the driver or handler name, while the other entries give the driver

specific information. Thus, by changing the entries, you can use one driver for different devices. For example, the trackdisk.device will drive both 3½- and 5¼-inch drives.

Note that 3½-inch floppy drives do not require a Mountlist entry. They are the default for the trackdisk.device and are mounted automatically by the system at power-up.

Always be sure to read the owner's instruction manual that comes with any new hardware. Some add-on devices require a Mountlist entry, and some do not. Some manufacturers provide a disk with the Mountlist already written for you, so you need only copy it over to your own Workbench. Much frustration can be avoided by reading the manuals that come with your new hardware.

Indicating Disk Swaps

Oftentimes you get a message from the system that asks you to insert a particular disk into any drive. The Amiga 3½-inch disk drives have a built-in switch that tells AmigaDOS whenever you insert or remove a disk from the drive. The clickity-clack you hear from an empty drive is just AmigaDOS checking to see if you have inserted a disk. When you insert a disk, AmigaDOS turns on the disk-activity light and reads the disk momentarily, just to find out what disk you have now inserted.

The Amiga 5¼-inch disk drives have no such switch. The system has no way of knowing automatically when you swap disks in the drive. You use the DISKCHANGE command to inform the system that you have changed a 5¼-inch disk.

The format of DISKCHANGE is:

DISKCHANGE <drive>

The <drive> parameter (usually df1:, df2:, or df3:) must match the Mountlist entry for the 5¼-inch drive.

If AmigaDOS asks that you insert a specific disk into a 5¼-inch disk

drive, use DISKCHANGE to signal the change; otherwise the task that asked for the disk will not continue.

Changing Task Priorities

You can load many programs into your Amiga at the same time, and the system seems to execute them simultaneously. Actually, the Amiga works on one task at a time, but switches between tasks so quickly that they all appear to be running at once.

Each task running in the system has a priority associated with it. The priority value ranges from −128 to 0 to +127. The most common priority value assigned to a task is priority 0. When the Amiga is interrupted by its internal timer, it searches for the highest priority task that is currently ready to run. If the priority of the current task is less than or equal to that of the next ready task, then the current task lies dormant and the system begins to work on the new task. Tasks of equal priorities will be executed in what is called a round-robin, where each will get roughly equal processor time.

For tasks that you start from the CLI, you can specify the priority of a task by using the command CHANGETASKPRI. For example, you can set the priority of the current CLI to 4 by entering:

```
CHANGETASKPRI 4
```

The AmigaDOS Manual cautions that although you can set any priority between −128 and 127, you should restrict your changes to values between −5 and 5 so as to avoid interfering with certain system tasks that use higher priorities. This ensures, for example, that the task that updates the output display runs when it should and gives you a stable display screen.

Why would you want to change task priorities? Let's say you have two CLI windows active. From one, you have started a word processor, while in the other, you are compiling a C program. If the compiler CLI has a priority value of 1 and the word processor task has a priority value of 0, the compiler has the highest priority and will get the lion's share of the system running time. Consequently, you could find your word processor unable to keep up with your typing—a very frustrating experience.

To end your frustration, you could change the priority of the individual CLIs before you start your word processor and compiler. When running more than one task at a time, you should set the priorities so that interactive tasks—ones that require a lot of keyboard or mouse entry, such as a word processor—have a higher priority than non-interactive ones, such as a language compiler. Note that when you set the priority of a CLI, you are also setting the priority of any CLI you start from the original or any task you run from the CLI.

Use CHANGETASKPRI with caution. It is usually best to let all tasks run at priority zero. Even small differences in priority may result in large differences in the running time allotted to two tasks. Misusing CHANGTASKPRI could negate the effects of multitasking by letting one task run all the time.

Changing the Stack Size

Every program you run on the Amiga requires a certain amount of workspace memory called the "stack." The stack is a block of memory that the program uses for temporary storage and other internal bookkeeping matters. Some programs simply assume that enough stack space has been reserved to allow them to run successfully. If a program actually requires more stack memory than has been allotted, the program will probably crash the machine.

You can increase the stack size for those programs that require a large stack (the SORT command is one of them) by using the STACK command. The format of this command is:

```
STACK stack_size
```

Stack_size is a number that indicates the number of bytes reserved for the stack. For example, if you enter:

```
STACK 10000
```

you increase the stack size for any task launched from the current CLI to 10,000 bytes.

The default stack size for any CLI is 4000 bytes. If an application

requires a larger stack, the manufacturer will specify this fact in the progam documentation. For example, the Amiga Assembler recommends a stack size of 16,000 bytes. SCULPT 3-D benefits from a stack as large as 30,000 or 40,000 bytes.

Although you could give each task a stack of 50,000 or 100,000 bytes to ensure that it never runs out of stack space, this is a big waste of memory. The idea is to provide exactly as much stack workspace as a program needs, and no more. The default size of 4K is enough for most programs.

8 AmigaDOS Command Scripts

This chapter shows you how to create text files that can be used as AmigaDOS command scripts. The chapter also details the most important script file on your system, the startup-sequence file.

Introduction to Script Files

If you have ever spent the considerable amount of time it takes to type a long series of commands you wish to execute, you will appreciate AmigaDOS command files. For instance, when you set a RAM: disk you must issue a command to create the RAM: disk, a series of COPY commands to populate it, and an ASSIGN or PATH command to tell AmigaDOS to look for commands in the RAM: disk. If you use this RAM: disk often, you will become very tired of typing all these commands whenever you start up your system.

With command files, however, you can put commands into a text file and get them all to run, in the sequence you entered them, by using the EXECUTE command. A text file that contains AmigaDOS commands is called alternatively a command file, an execute file, a batch file, or a script file. In this book, I refer to these text files as command scripts or simply script files.

Types of Command Scripts

Command scripts can be simple or complex. A simple script may consist of a single command such as DIR. Thus, you can use your word processor, text editor, or even the COPY command (copy from

the keyboard to disk) to create a text file consisting of one line—DIR. If you name this file dir.txt, you would execute the script as follows:

```
EXECUTE dir.txt
```

The EXECUTE command reads the file dir.txt and runs the commands it finds. In this case, it gives you a listing of the contents of the current directory.

Creating single-command script files serves little purpose, but when you set up script files containing hundreds of CLI commands, they can save you an enormous amount of time and help you avoid numerous typing errors.

Command scripts can be very complex. In addition to simple scripts, we will examine scripts that use parameter substitution and interactive scripts that accept user input as they execute. We will also look at command scripts that branch based upon logical decisions.

Running Command Scripts

You can call command scripts in various ways. One program on your Workbench disk—XICON—lets you run a command script with a simple click of your mouse. From the CLI, the easiest method of executing a command script is to use the EXECUTE command, as in:

```
EXECUTE myfile
```

where myfile is a command script in the current directory. An alternative method of executing scripts from the CLI is to simply type the name of the script file. This method works only under Workbench 1.3 and only if the "S" flag is set on the file. For example, if you create a script file called myfile, and you set its "S" flag with the PROTECT command, you can execute the script by entering:

```
myfile
```

There are also two indirect methods of executing a script file. In the first, you invoke a startup file for a new CLI. For example, the line:

```
NEWCLI CON:10/10/300/200/NEWCON FROM myfile
```

will start the new CLI and execute the script contained in myfile. This method is useful for setting special paths or starting a Shell-like program.

The second method involves resetting your Amiga. Anytime you power up or reset your computer, AmigaDOS executes the command script called startup sequence located in the S directory of your boot disk. Startup sequence is probably the most commonly executed Amiga command script. (If you are familiar with MS-DOS, df0:s/startup sequence serves the same purpose as autoexec.bat.)

You use startup sequence to customize your working environment. For example, many people use it to add to the AmigaDOS search path, to copy commands to RAM:, to start various utility programs, or to make assignments required by various applications programs.

EXECUTE and the T Directory

When EXECUTE is given a command script, it reads the script line by line and performs each command it finds in the sequence in which it appears in the script file. When EXECUTE first begins to run, it produces a temporary file in the T directory of the current disk. If there is no T directory in the root of the current disk, the EXECUTE command fails and reports an error. The same error occurs if the current disk is write protected. The reported error is:

```
Cannot create: ":t/Command-0-T01"
```

Note that in version 1.3, if you have assigned a directory to T:, EXECUTE will use that directory, not :T.

Locating Script Files

When you run EXECUTE, AmigaDOS looks in two places for your script file. If it does not find the file in the current directory, AmigaDOS then looks in S:. Therefore, the most logical place to put scripts you use frequently is in S: so that you will not have to think about where

they are located. (This is analagous to moving often-used commands to your C: directory.)

Modifying Commands with EXECUTE

You may also use EXECUTE to redefine the way a command is performed in order to save typing time. For example, let's say you prefer the information you get from the following command to the standard DIR and LIST output:

```
LIST NODATES PAT #?
```

You can avoid the tedium of entering all those parameters each time you want to examine the contents of a directory by creating a script containing exactly the command line LIST NODATES PAT #? and storing it in a text file, perhaps one called LDIR. To further ease your typing burden, you might rename the EXECUTE command as X. Now, to get your preferred form of directory listing (assuming LDIR is in either the current directory or the S: driectory), enter:

```
X LDIR
```

and you will get the same output as if you had typed LIST NODATES PAT #?.

Script-File Parameter Substitution

One of the more powerful features of EXECUTE is its ability to substitute parameters entered on the command line for key or dummy names found within the command script. This feature is called command-line parameter substitution.

To demonstrate, let's use the LDIR example above. At present, the LDIR script lists only the current directory. To get it to work on any directory, you must enter a command line such as:

```
X LDIR df1:
```

Here, however, we need command-line parameter substitution, which requires that you use the .KEY statement in an execute script. .KEY takes a parameter you enter on the command line and puts it into a

variable. (In the example below, the variable is called dirname.) Then, whenever the variable appears in the script, the parameter is used instead.

Here is a modified LDIR file that accepts a directory parameter:

```
.KEY dirname
LIST NODATES PAT #? "<dirname>"
```

Now, when you enter:

```
X LDIR df1:
```

EXECUTE reads the LDIR file and immediately finds a period in the first column of the first line. EXECUTE now knows to watch for parameter substitution. EXECUTE also finds that only one keyword is defined—dirname. On the next line, EXECUTE finds this keyword used with a command. It copies this line to T:Command-1-01, while making the parameter substitution (df1: for dirname). The file now appears as:

```
LIST NODATES PAT #? df1:
```

The notation <dirname> does not appear in the temporary file. EXECUTE then reads the temporary file and performs the commands in the file. If you do not supply a parameter on the command line, EXECUTE substitutes a blank space during parameter substitution. You should be sure that a blank space will not mess up the operation of your script when you create the script.

Although the .KEY statement need not be the first statement in the script file, it should always begin at the left of any line on which it appears. This alerts EXECUTE to its special function.

To specify more than one keyword parameter, separate them by commas, as shown below. Note that only the first .KEY statement will be recognized and used in any command script:

```
.KEY parameter1,parameter2,parameter3
```

You must not use spaces between parameters. If you try:

```
.KEY parameter1, parameter2, parameter3
```

the parameter substitution will fail.

When parameters are specified on the command line, they are assigned to the names in the .KEY statement by position. That is, the first parameter is assigned to the first name, the second parameter to the second name, and so on. EXECUTE, however, allows you to specify a keyword on the command line and thereby change the order of the parameter assignments. Named keyword assignments are made first, and the remaining parameters are assigned in sequence to whatever keyword positions are left over.

For example, create the script file below, name it test, and store it in your S: directory:

```
.KEY p1,p2,p3
ECHO "First parameter is: <p1>"
ECHO "Second parameter is: <p2>"
ECHO "Third parameter is: <p3>"
```

Now, try the following commands:

```
EXECUTE test X Y Z
EXECUTE test p2 Y p3 Z p1 X
EXECUTE test p3 Z X Y
```

Notice that the result of all of the above commands is the same. This shows you that not only does the .KEY statement establish the order of the parameters expected on the command line, but it also allows the user to specify the keyword name to change the sequence in which the parameters are accepted. If you recall Chapter 2, this is the same way most AmigaDOS commands operate.

Testing Conditions in Command Scripts

You can create test conditions in script files and branch to one command or another based upon the result of the test. This gives you the ability

to create flexible command scripts you can use in different situations.

You use the control word IF to create test conditions within your execute scripts. In fact, IF can be used only inside execute scripts. When you form an IF statement, you must tell EXECUTE where that statement ends by using an ENDIF. When you have something else that EXECUTE should do when the results of the IF statement are found to be false, you can also add an ELSE statement to the IF. . . ENDIF block. The two possible ways to construct an IF statement are:

```
IF <condition is true>
<perform these commands>
ENDIF
```

or:

```
IF <condition is true>
<perform these commands>
ELSE <if condition is false>
<perform these commands>
ENDIF
```

For every IF, there must be one ENDIF. The ELSE is optional. Remember also that all statements must be on separate lines in the script file.

The conditions that IF can test are:

- Does a particular file or directory exist? (EXISTS)
- Does a parameter have a particular value? (EQ)
- Did the last command return a WARN value? (WARN)
- Did the last command return an ERROR value? (ERROR)
- Did the last command return a FAIL value? (FAIL)
- Reverse the meaning of any of the above. (NOT)

Testing Whether a File Exists

Here is a short example that tests to see if a file exists. If the file exists, the script types the file and then deletes it. If the file does not exist, the script creates the file so that it will exist the next time the script is run. Thus, if you run the script twice, you will be back where you

started (assuming that a file by the same name does not exist the first time you ran the script).

Here is a listing of the script. Give it the name iftest and save it in your S: directory.

```
IF EXISTS shortfile
  ECHO "Found shortfile, typing contents:"
  TYPE shortfile
  DELETE shortfile
  ECHO "shortfile deleted"
ELSE
  ECHO "Creating shortfile"
  ECHO >shortfile "Here I am!"
ENDIF
```

Reversing a Test Condition

You use the word NOT right after the word IF to reverse the conditions of a test. Whereas:

```
IF EXISTS <filename>
```

is true if a file exists, the following:

```
IF NOT EXISTS <filename>
```

is true if the file does not exist. As mentioned above, the NOT control word can be used with any of the conditions described for the IF test.

Testing for Equality

Use equality tests to check that you have entered the correct parameters for a script. Here is a short example that uses the EQ test.

Enter the following script and save it as check.

```
.KEY p1,p2
IF "<p1>" EQ "ALL"
  ECHO "Found 'all'"
  QUIT 0
ENDIF
```

```
IF "<p2>" EQ "ALL"
  ECHO "Found 'all'"
  QUIT 0
ENDIF
ECHO "Did not find 'all'"
```

To try the example, try the following test statements. You should get the same results as those listed below. When you enter:

```
EXECUTE check this that
```

AmigaDOS responds:

```
Did not find 'all'
```

Enter:

```
EXECUTE check all that
```

and AmigaDOS responds:

```
Found 'all'
```

Finally, if you enter:

```
EXECUTE check you all
```

AmigaDOS responds:

```
Found 'all'
```

Setting Default Values for Parameters

If you do not enter a parameter value on the command line, the parameter assumes the value " "—in other words, it becomes a null string. You can establish a default value for parameters in your script files with one of two methods. The first way to establish defaults is by using the .DEF statement. Here is a script file that will always give you

a listing of the contents of your root directory unless you pass a parameter:

```
.KEY direc
.DEF direc ":"
DIR <direc>
```

Note that only one .DEF statement per parameter name is accepted in a command script. Any attempt to redefine a parameter using a second .DEF will be ignored.

In addition to establishing defaults, you can use .DEF as a form of shorthand in your command scripts and specify more parameters in the .KEY statement than you may actually need. For example, if you frequently use the string ".library" in an execute script, but do not want to type .library at every occurrence in your script, you may use "x" as shorthand for ".library." Enter the script and save it as findlib:

```
.KEY which,x
.DEF x ".library"
.DEF which ""
ECHO "Looking for <which><x> in LIBS:"
IF EXISTS LIBS:<which><x>
  ECHO "found <which><x> in LIBS:"
ENDIF
```

In the same vein, if you have an extremely long command such as IF NOT EXISTS df0:fonts/ruby/8, you can use .DEF to form an alias for the command itself.

The second method to enter a default for a parameter in a script file is to specify the default value within the parameter substitution brackets. Thus, when you use the parameter, it takes the form:

```
<parameter$default>
```

Here, you first specify the parameter name, then use a dollar sign ($) to indicate that a default follows, and, finally, indicate the default string.

When EXECUTE sees this form of string, it first looks to see if the user has provided a value for the parameter. If so, it uses the user-defined value. If the user has supplied no value on the command line, then the parameter uses the default listed after the dollar sign. Here is an example:

```
.KEY p1
IF "<p1$novalue>" EQ novalue
  ECHO "There was no parameter typed with the command."
ELSE
  ECHO "The value of the first parameter is: <p1>."
ENDIF
```

Save the script as deftest and try the following commands to see what the above script does:

```
EXECUTE deftest
```

AmigaDOS responds:

```
There was no parameter typed with the command.
```

Now, enter:

```
EXECUTE deftest somevalue
```

AmigaDOS responds:

```
The value of the first parameter is: somevalue.
```

Changing Delimiters

You can change the $ character that EXECUTE uses to separate the parameter name from its value by using the .DOLLAR command. For example, using:

```
.DOLLAR ,
```

in a command script changes the separator to a comma, letting you use the dollar sign as a normal character in your parameter definition.

You can also substitute other characters for the brackets that set off parameters in the script. Set the left bracket with the .BRA statement,

and the right one with the .KET statement. (Notice that if you put these two statements together, they seem to spell BRACKET.)

You can also change the "." that precedes .KEY, .DEF, .BRA, and other command-script statements by using the .DOT statement to change the period to some other character.

Finally, note that you can use a semicolon (;) to put comments into a script file. Comments are useful if you want to modify a script later to change its operation slightly.

WARN, ERROR, and FAIL Conditions

AmigaDOS commands and command scripts often return information that you can test in a file. These return values indicate whether a command or script had any problems during execution. You test return values with the control words WARN, ERROR, and FAIL.

Normally, if a return value is 10 or greater, an execute script will stop on its own and not continue. You can change the default failure value, however, by using the FAILAT command. Here are three examples that illustrate how the control words WARN, ERROR, and FAIL are interpreted in a command script.

Example 1:

```
FAILAT 25
TYPE  ;deliberate error, no arguments
IF WARN
  ECHO "Return code is 5 or greater"
ENDIF
```

Example 2:

```
FAILAT 25
TYPE  ;deliberate error
IF ERROR
  ECHO "Return code is 10 or greater"
ENDIF
```

Example 3:

```
;deliberately not specify FAILAT
;default is FAILAT 10
TYPE  ;deliberate error
IF FAIL
  ECHO "Return code is 20 or greater"
ENDIF
```

A deliberate error is built into each of the three example script files. The command TYPE is used without a parameter—it has no file to type. When this occurs, the TYPE command issues the response "bad args" and returns an error value of 20.

In the first example, FAILAT is set at 25. Thus, when TYPE returns an error value of 20, the script does not fail. The IF WARN test then tests to see if the error code is 5 or above (the higher the code, the more severe the error). In this case, the code is 5 or above, so the message is printed.

In Example 2, FAILAT is once again set at 25, so the script does not fail when TYPE returns a value 20 error. The IF ERROR statement then checks to see if the return code is 10 or above. Since this statement is true, the message is printed.

The third example illustrates what happens when an error code is above the FAILAT threshold. In this case, FAILAT is left at its default of 10. Because the return value from TYPE is greater than 10, the EXECUTE command controlling the script fails and the message is not printed. The script does not get past the TYPE command.

These examples illustrate a couple of other points. First, FAILAT lasts only for one command—you must explicitly reset the FAILAT value if you want it to be other than 10. Next, note that the IF tests for WARN, ERROR, and FAIL also last only for one command. In other words, if you want to test for a return (possibly an error) value from a command, you must do the test immediately after the command itself is executed. Each command sets the return value, including the IF test itself, so you must perform your tests with care.

Each of the tests in the examples is formulated correctly, with FAILAT placed before the command to be tested, and the IF test immediately

following it. FAILAT prevents EXECUTE from aborting the script when a value over 10 is encountered. The IF statement tests the return value from TYPE.

Note that it is not possible to use the QUIT command to pass back an error code other than its own. The QUIT command will cause a particular command script to exit, and to pass its own return code, but it will not pass an error code generated by any other command to another command script.

The best advice I can offer is to not nest command scripts. Make one script perform all the functions you need, using IF and SKIP to move from item to item, thus avoiding having one execute script call another except where there is no need to pass values between them.

Labels in Your Command Scripts

You can insert a label anywhere you wish in your command scripts. The label serves no purpose other than to act as a target for a SKIP statement. To establish a label, you use the LAB statement. For example:

```
LAB thisLabel
```

Labels must be unique. You cannot use the same label name twice in the same script file. Also note that as with all other AmigaDOS commands, the labels will be treated as though you entered them in all uppercase letters. Labels are not case-sensitive.

Skipping to a Label Statement

The SKIP statement lets you skip all the statements and commands between the SKIP statement itself and the LAB statement to which it skips. Here is an example:

```
;other statements here
SKIP go_west
;more statements here
LAB go_west
;next group of statements
```

In this example, SKIP reads each statement in turn and ignores all statements until a "LAB go_west" is found. SKIP skips forward only

in a command script. A direct backwards skip is not possible, although one can be simulated. See the menu script below for an example of a script that skips backwards by calling itself.

Returning an Exit Value

To exit an execute script at any time, you use the QUIT command, which allows you to specify an exit value. The form that QUIT takes is:

```
QUIT value
```

where value is a return value such as:

```
QUIT 5
```

or:

```
QUIT 20
```

The quit value is returned to the system and can be tested inside the execute script.

Practical Uses—A Menu Script

It's time to put some of this stuff to work. Here is an example based on a command script created by Andy Finkel of CATS, the Commodore-Amiga Technical Support staff. The example simulates branching backwards using the skip command, primarily by having the command script call itself.

I would suggest that after you have run this script, you modify it to suit your own tastes. This example is provided to show you the basic principles of complicated scripts. Store this script under the name menu in your S: directory

```
.BRA {
.KET }
COPY s:menu to RAM:menu
ECHO "CLI Menu Example"
```

```
ECHO "Would you like info, a list, a directory, cd, or quit?"
ECHO "Type I, d, c, or q, followed by RETURN"
SKIP <* >NIL: ?
;
LAB i
INFO
SKIP done
;
LAB l
LIST
SKIP done
;
LAB d
DIR
SKIP done
;
LAB q
quit
;
LAB c
ECHO "To change directories, type a new directory name."
ECHO "If you just press RETURN, no change takes place."
CD <* >NIL: ?
ECHO "The current directory is now:"
CD
SKIP done
;
LAB
ECHO "Try again"
;
LAB done
WAIT 1  ;Let user see results
EXECUTE ram:menu
```

Now, from any directory, you simply enter:

```
EXECUTE menu
```

and this command script takes over. Try it.

There many interesting aspects to the above example. Notice first that the .BRA and .KET are changed. The execute script uses redirection symbols so this is the safest way to make sure that EXECUTE does not get confused.

Also notice the SKIP command:

```
SKIP <* >NIL: ?
```

Let's look at this command one portion at a time, starting with:

```
SKIP ?
```

The question mark prompts the user for input. The SKIP command displays the following:

```
LABEL:
```

and waits until the user enters a label name. In the example, however, the SKIP and the question mark are separated by:

```
>NIL:
```

meaning that the LABEL: prompt is sent to the NIL: device instead of to the screen; the user never sees this prompt. Finally, between the SKIP and the >NIL:, you see:

```
<*
```

These characters redirect the input for the SKIP command to the console. In other words, it hooks your keyboard to the SKIP command. Normally, when you execute a command script, there is no connection between the console and the commands in the script file. The programs get all their input from the script itself.

Also of note are the lines:

```
LAB
ECHO "Try Again."
```

A LAB statement normally has a label associated with it. Because the one in the example does not, it becomes the place that SKIP goes to if the user just presses RETURN or enters anything other than an already existing label.

This example demonstrates the use of labels, user interaction, and the SKIP command. It also shows how you can string simple commands together to create complex, interactive scripts.

Summary of Command Script Statements

Here is a quick reference guide to the special statements used in command scripts.

- **.KEY (or .K)** specifies keywords that identify variables for parameter substitution.
- **.DOLLAR** changes the character that separates parameter names and labels.
- **.BRA and .KET** change the beginning and ending bracket characters that enclose a keyword used in a command script.
- **.DEF** establishes a default value for a parameter if the user does not provide a value for it.
- **.DOT** changes the character that indicates a KEY, DOLLAR, BRA, KET, DEF, or DOT command.
- **LAB** specifies a label, the target of a SKIP command. Labels must be unique.
- **SKIP** passes over lines in the command script until it reaches the specified label.
- **IF** tests a specific condition. When the result of the test is true, EXECUTE performs the commands that follow the IF statement in the script. If the condition is false, EXECUTE searches forward to the next ELSE or ENDIF statement.
- **ELSE** marks the beginning of the block of commands executed when the corresponding IF statement is false.
- **ENDIF** ends the block of commands whose execution is dependent upon the corresponding IF statement.
- **NOT** reverses the conditions of an IF test.

- **EQ** tests if two values are equal. If they are, the IF condition is true.
- **EXISTS** tests if a file or directory exists.
- **WARN** tests if a return value is 5 or above.
- **ERROR** tests if a return code is 10 or above.
- **FAIL** tests if a return value is 20 or above.
- **FAILAT** sets the return value at which the command script will abort.
- **QUIT** ends the execution of the current command script and returns a value to the calling program.

Startup Sequences

Now that we've talked about what goes into the a command script, let's look at the most frequently used command script, the startup sequence. As mentioned at the top of this chapter, the startup sequence file executes automatically every time AmigaDOS starts up, whether from a power-up reset (turning on the power switch) or from a warm-boot reset (hitting Ctrl-Left Amiga-Right Amiga). Startup sequence resides in the S:, which by default is df0:s.

Command scripts let the computer do the dirty work of typing a sequence of commands. The startup-sequence script helps you create the computing environment that works best for you, without ever typing a command (after you modify the file, of course). You might want special utilities, such as a virus-detection program or a font-acceleration program, running in the background whenever you reset your machine. You might want to have special names assigned to particular directories or disks, or you may simply want to have the same application program up and running whenever you fire up your computer. Your startup sequence does all this and more for you. It can be as simple or as complex as you want to make it. Best of all, you create and edit it just as you would any other command script file.

The Simplest Startup Sequence

Some people say the best startup is no startup at all. I do not agree, but resets certainly take a lot less time without a startup-sequence file.

To try this one out, enter your Workbench 1.3 S directory and type:

```
RENAME startup-sequence startup-sequence.save
```

Now, reset your computer. Pretty fast, eh? Back to the CLI in record time. When you don't have a file named s/startup sequence on your boot disk, AmigaDOS boots into the CLI and stops. But if you make this CLI window smaller and look behind it, you will discover that not even Workbench is running.

Booting to Workbench

A minimal startup sequence that boots you directly into the Workbench interface contains just two lines:

```
LOADWB
ENDCLI > NIL:
```

The first line brings up Workbench and its icon interface. The second line closes and deletes the CLI window. The redirection command sending the output of the ENDCLI command to NIL: simply avoids printing the message "CLI task 1 ending."

If you want to have Workbench loaded but keep the original CLI active when your startup is completed, just delete the line ENDCLI > NIL:. You still have the CLI available after rebooting, but you also have access to Workbench, which is behind the CLI window.

Moving Commands to a RAM: Disk

Startup sequences are often used to create a RAM: disk and to move commonly used commands onto the disk. This can be a tedious process, but it greatly speeds up the performance of your system. People with more than 1MB of RAM commonly use a startup sequence similar to the one below.

```
ECHO "Workbench 1.3 RAM: startup"
BINDRIVERS
;
MAKEDIR RAM:c
;Any access of RAM: forces the system
;to create the RAM: disk if it isn't present
COPY C:COPY TO RAM:c
COPY C:DIR TO RAM:c
COPY C:LIST TO RAM:c
COPY C:MAKEDIR TO RAM:c
COPY C:ASSIGN TO RAM:c
COPY C:TYPE TO RAM:c
COPY C:EXECUTE TO RAM:c
PATH ADD RAM:c
LOADWB
```

The more files you copy to RAM:, the longer it will take to perform your startup sequence and the less free memory space you are leaving for your programs and data. Having these commands in memory all the time is inefficient in terms of space, but very efficient in terms of speed. If you do not use a particular command very often, you should not load it into a RAM: disk.

Using a Recoverable RAM: Disk

One variation on the RAM:-disk concept is the recoverable RAM: disk. Available in the Amiga public domain as VD0: and on Workbench 1.3 as RAD:, these devices are RAM: disks that do not lose their contents during a warm boot. Once loaded, their contents will not be destroyed by an accidental reset or a guru-meditation error. Using a recoverable RAM: disk is not difficult; it requires simply that you test for the existence of the disk before you start copying files into it. The first time after you power up, the recoverable RAM: disk will not exist—you will have to create it and populate it with files. Subsequently, your startup sequence should detect the presence of the recoverable RAM:

disk and skip past the copy commands that populate it. Here is a startup sequence that does just that, using the 1.3 RAD: device.

```
ECHO "Workbench 1.3, RAD: startup"
;
BINDDRIVERS
;
FAILAT 15  ; If RAD: is already exists, the
; next command results in a return value of 10.
MOUNT RAD:
IF NOT EXISTS RAD:c
  MAKEDIR RAD:c
  COPY C:COPY TO RAD:c
  COPY C:DIR TO RAD:c
  ; copy other important commands to RAD:c
ENDIF
PATH ADD RAD:c
LOADWB
ENDCLI >NIL:
```

If RAD: already exists, the above startup sequence will print a message that "Device RAD: is already mounted" and skip the COPY commands.

Making Commands Resident at Startup

A new AmigaDOS 1.3 command, RESIDENT, forces a particular command to become part of the operating system contained in memory. Thus, when a command is resident, AmigaDOS no longer needs to load it from a disk, even a RAM: disk, in order to run it. The command is always available and can be used by many different CLI tasks at the same time.

I discuss the particulars of RESIDENT in Chapter 10. For an example of how it is used in a startup sequence, take a look at the startup-sequence file on the new Workbench 1.3. Also examine the file StartupII, a command script that is called by startup sequence. These

files demonstrate many of the concepts presented in this chapter. Study them and modify them to your heart's content.

Other Ideas for Your Startup Sequence

In preparing this book, I talked with many people about their startup sequences. One of the most popular startup-sequence programs, SETCLOCK, sets the system clock from the battery-backed clock on the Amiga 500 and 2000. Another popular concept is to launch the virus-detection program VIRUSX from the startup sequence using the RUN command. Programmers are particularly likely to have elaborate startup sequences. They set up their entire development environments when they first power up their Amigas, thus maximizing the time they actually spend programming.

Of all the people I talked with, each had a different idea of what should be included in a startup sequence. As you experiment and grow comfortable with your Amiga, you will gradually develop a startup-sequence file that suits you best.

Conclusion

Command scripts give you a great deal of flexibility and can save you a lot of work at the keyboard. The df0:s/startup sequence file, which executes whenever you reset your system, creates your initial working environment. The startup sequence that is best for you depends as much on personal preference as on the hardware and software configuration of your system.

If you need more help in selecting what to put into a startup sequence, you might consider accessing one or more of the Amiga BBSs and examining the uploaded files. Many people work on a particular startup-sequence file over a span of months, and then share it with others. These files can be a great place to start creating your own version of the perfect startup sequence.

9 Console Control Characters

This chapter demonstrates how you can customize the look of your CLI windows by sending control characters to the console device. You can send these characters from the keyboard or from a script file.

The Customized Console

Whether you communicate with AmigaDOS through the CLI or through the optional Shell supplied with AmigaDOS 1.3, your input is first filtered through a console device. In the case of the CLI, the console device is called CON:. If you use Shell, the console device is NEWCON:. In either case, you can use control-character sequences to customize the look of your output.

Here is a short sampling of the changes you can make to your CLI or Shell output:

- Clear the window
- Move to a specific position in the window
- Change the font to italic or bold or both
- Change the font to inverse video
- Make the cursor disappear and reappear

These and other output changes are effected by sending commands to the console device. Some console commands consist of a single character and are best sent directly from the keyboard. Others, called "escape sequences" (because they always begin with the Escape character) are most easily sent when embedded in a command such as ECHO. The escape sequences are best used to format the output of your command scripts. I will discuss both types of control characters

in this chapter, with the emphasis on the more versatile and powerful escape sequences.

Note that you can use AmigaDOS without ever sending a command to the console device. This chapter is intended for those who want to customize the output of their AmigaDOS commands.

Single-Character Console Commands

The following single-character commands are available to you when you use the CLI and its CON: device. The Shell does not support all these functions, but it does handle the most important ones, such as Form Feed and Line Feed. Try these commands from your CLI window:

Ctrl-H Backspace—perform a destructive backspace
Ctrl-I Tab—move to next horizontal tab
Ctrl-J Line Feed—move cursor down one line
Ctrl-K Vertical Tab—move cursor up one space
Ctrl-L Form Feed—clear the screen
Ctrl-M Carriage Return—cursor to left margin
Ctrl-N Set High Bit—use alternate character set
Ctrl-O Reset from Ctrl-N—standard character set

These are not all the single-character commands that can be sent to the CON: device; I have ignored those that return information from the console and are therefore useful only to programmers. For more information about the commands not covered here, see the Amiga ROM Kernel Manual, Volume II, Libraries and Devices.

With the exception of Form Feed, the single-character console commands are not terribly useful. The real action begins when you send escape sequences to the console device.

Introduction to Escape Sequences

An escape sequence is a string of characters that begins with the Escape character. This is the character you generate when you press the Esc key. Escape is a non-printing character used by many different devices—including most printers—to differentiate between data to be processed and commands to the device itself.

To see how escape sequences affect the output of the console device, enter the following command:

```
ECHO "*Ec*E[1mCLI Menu Example*E[0m*N"
```

This command clears the console window, prints "CLI Menu Example" in boldface, and resets the console output to normal type.

When you use the ECHO or PROMPT command, AmigaDOS handles a pair of character sequences in a special manner, converting them into escape sequences for the console device. The first character sequence is:

```
*E
```

This represents the escape character. When you use this sequence in an ECHO or PROMPT command, the command sends the Esc-key character to the console device. The other special-character sequence is:

```
*N
```

This is the newline sequence. When you include it in an ECHO or PROMPT command, a carriage return is sent to the console device.

Thus, in the command example given earlier, the key sequence *Ec sends the escape character to the console device, followed by a "c." The sequence sent is Escape-c, which is recognized by the console device as the command to "reset the console to its original state." Coincidentally, this command clears the screen.

The second command sequence is *E[1m. This is a complex command that changes the console text mode to boldface. Once again, the *E tells the console that what follows is a command, not something to be output to the console window. The [1m changes the graphics rendition to bold mode. The third escape sequence, *E[0m, resets the graphics rendition to its previous state. By the way, the reset to initial state (*Ec), will set both the text styles and colors back to normal if you had changed them previously.

Note the difference between the character sequences. The first, *Ec,

is the escape character followed by a command character. In the latter two, the escape character is followed by "[," a number, and a command character.

The sequence *E[is called a Control Sequence Introducer, or CSI. Whenever it precedes a command, it indicates that the command can accept numeric values that set either parameters (as in the "set graphics rendition" above) or a counter telling the console device how many times to perform an operation.

The Console Output Commands

Here is a list of the escape sequences for the CLI's CON: device. Many of these also work with the Shell's NEWCON: device. The sequences are printed as you would have to enter them in an ECHO command.

- ***E[@** This sequence inserts a character into a line in the console window by moving all characters one space to the right and inserting a space character. The cursor remains over the space. For example:

  ***E[@**

  inserts one character, while:

  ***E[10@**

  insert 10 characters.
- ***E[A** This command moves the cursor up one line.
- ***E[B** Moves the cursor down one line.
- ***E[C** Moves the cursor one space to the right.
- ***E[D** Moves the cursor one space to the left, thus performing a nondestructive backspace.
- ***E[E** Moves the cursor to column 1 of next line.
- ***E[F** Moves the cursor to column 1 of previous line. Examples of the above commands are:

  ***E[4A**which moves the cursor up 4 lines:

  ***E[12C**

  which moves the cursor 12 spaces to the right, and:

  ***E[3F**

  which moves the cursor up three lines and to the left margin of the window.

- ***E[<row>;<col>H.** Moves the cursor to the specific row and column number. Examples of this sequence are:

  E[1;1H

  which moves the cursor to row 1, column 1 of the console window, and:

  ***E[5;10H**

  which moves the cursor to row 5, column 10.
- ***E[J** Erases the current line from current cursor position to the end, and all lines below the current line.
- ***E[K** Erases the current line from the current cursor position to the end of the line.
- ***E[L** Inserts a blank line just ahead of the current line. For example:

  ***E[4L**

  inserts 4 blank lines above the current line.
- ***E[M** Deletes the current line and moves any lines beneath it up one to fill the gap. For example:

  ***E[3M**

  deletes the current line and the next two below it.
- ***E[P** Deletes the character the cursor is resting on. For example:

  ***E[6P**

  deletes the current character and the next 5 to the right. Any other characters on the line are moved to the left to fill the gap.
- ***E[S** Deletes the top line of the console window, scrolling the remaining lines up.
- ***E[T** Deletes the bottom line of the console window, and scrolls the remaining lines down.
- ***E[<p1>;<p2>;<p3>m** Selects the graphics rendition mode. This command accepts any number of numeric parameters—separated

by semicolons (;)—preceding the "m" that identifies the command. Here are the effects of the different parameters:

0 — set plain text
1 — set bold face
3 — set italic
4 — set underscore
7 — set inverse video
30 — set foreground text color to color 0
31 — set foreground text color to color 1
. . .
37 — set foreground text color to color 7
40 — set background text color to color 0
. . .
47 — set background text color to color 7

On a standard Workbench, the foreground color is color 1 (white), and the background color that surrounds the text is color 0 (blue). Color 2 is black and color 3 is orange.

Thus, if you want to set the graphics rendition to standard, meaning normal white text against a blue background, you would use the following command sequence:

```
ECHO "Setting Normal Graphics Rendition"
ECHO "*E[0;31;40m"
ECHO "Normal Text, FG color = 1, BG color = 0"
```

If you want to use use bold, underline, italicized characters over an inverse video background, try the following:

```
ECHO "Setting bold, inverse, underlined, italics"
ECHO "*E[1;3;4;7m"
```

Note that you have to be careful in which mode you leave your CLI console window. It is perfectly fine to use ECHO to change how characters are output to the console. You will encounter problems,

however, when you try to type characters while the console is in a non-standard mode. You can get some awful looking results.

AmigaDOS and the CLI output each formatted line; the line is formed completely before being output. When you type into a window, however, each character is individually placed in the window. Thus, when the console is in italic mode, for example, the text output functions cannot merge all of the characters before writing them to the window, resulting in some horrible looking output. The best approach is to use special modes for console output, but to reset the mode to normal when a command is finished and the CLI is expecting input.

Console-Sizing Commands

The next four commands tell the console device to change the positioning of output text and the size of the text in the console window. You can use them to define a separate scrolling region in the middle of a window.

When you issue the command to reset the console, the system looks at the font that you have set and figures out how many lines will fit in the window and how many characters will fit on each line. It also determines the position at which to start the first line of text, counted in scan lines from the top of the window. Finally, the system determines a left-offset—a left margin position measured in pixels from the left edge of the window. Using the following commands, you can manipulate these values so that the active area of your window differs from the actual window dimensions.

- ***E[<PL>t** This command sets the page length measured in lines of the current font (PL indicates page length). For example, a full-screen console window normally holds 24 lines of text. To protect the text on the bottom 12 lines and define a scrollable window above that area, you would send the command:

  ***E[12t**

  to limit the text output to the top 12 lines of the window.
- ***E[<TO>y** Sets an offset, measured in scan lines, below which the console window will begin its output.
- ***E[<LL>u** Specifies how many characters of the current font should

be allowed on each line before an automatic line break is printed. For example:

```
ECHO "*E[32u Column Width Is Limited Now"
```

will prevent any disturbance in the console window to the right of column 32 of the text.

- ***E[<LO>x** Causes the console device to set a left margin at the specified location. For example, because the standard system font TOPAZ 8 is about 8 pixels wide, the command sequence:

  ```
  *E[160x
  ```

  causes an indent that is 20 characters from the left border of the window.

If you want to see the effects that these parameters have when used together, try the following execute-script:

```
; store this file as ram:test
ECHO "*Ec Clear the screen"
LIST ;we can protect part of this
ECHO "*E[12t*E[43y*E;bo40u*E[84x Smallish Box"
LIST
```

Open your CLI window to a maximum size and enter:

```
EXECUTE ram:test
```

As you can see, the effect is very interesting. One major problem with this technique, however, is that once you set the margins of a console window, the system can no longer change any of these parameters. Thus, if you or some other user resizes a window you have adjusted, the output may not show up in a visible area of the window! Be careful using these commands.

Interactive Designing of CLI Screens

Thus far, I have listed console-escape sequences in the form you would use when including them in ECHO and PROMPT commands. If you wish, you can create short console-command sequences with your

keyboard, save each sequence as a small file, and output the sequences to the console using the TYPE command.

To create these short files, I first redirect console input to a file. For example:

```
COPY * to RAM:test
```

Now, I enter the escape sequence. Note that I cannot use the special characters *E and *N that are recognized and translated by ECHO and PROMPT. Instead, where I normally enter *E, I now hit the Esc key; where the sequence requires *N, I now hit Return. To stop recording my key presses to a file, I enter Ctrl-\.

Now, instead of having to put sequences into ECHO statements and ECHO statements into command scripts, I can simply record some simple escape sequences to disk and TYPE them when I want. For example, if after entering the above command, I hit Esc[1mCtrl-\, I will have created a file called RAM:test that will turn my CLI output to boldface when I enter:

```
TYPE RAM:test.
```

By stringing lots of these little files together using the JOIN command, I can create escape-sequence files for any occasion.

Conclusion

Although not essential to using the CLI, knowledge of how you can control the console device is fun and rewarding. Experiment with the different commands and add sparkle to your script files.

10 AmigaDOS 1.3

This chapter covers the AmigaDOS 1.3 release. It describes new commands and devices, changes to old commands, and making commands resident with the Shell.

The 1.3 Release

When this book was conceived, it was my intention to integrate information on AmigaDOS 1.3 throughout the book. As time passed, and as Commodore's shipping date for the 1.3 upgrade slipped from April '88 to sometime in the fall (1.3 is still not released as I write this), I decided to consolidate the information on 1.3 into a separate chapter. Thus, if you have yet to upgrade to AmigaDOS 1.3, you can use the information contained in the first nine chapters, and read this one when you obtain AmigaDOS 1.3. If you already have 1.3, you can supplement the information in the first nine chapters with the information presented here. Note that I have integrated the 1.3 commands into Appendix A.

Why 1.3?

OS 1.3 is the fourth major release of the Amiga operating system. It addresses many of the problems users have encountered with previous releases. Commodore's main goals for OS 1.3 were to improve the speed and quality of the printer drivers, vastly improve the speed of hard-disk access, and let users boot from a hard disk instead of from a floppy. OS 1.3 accomplishes all this and goes quite a bit further. For AmigaDOS users, Commodore has provided new commands, new

devices, improvements to old commands, and a new interface. AmigaDOS 1.3 is more powerful and much easier to use than previous versions.

The New Commands

The commands new to the AmigaDOS 1.3 C: directory are ASK, AVAIL, FF, GETENV, LOCK, NEWSHELL, REMRAD, RESIDENT, SETENV, and SETCLOCK. I will deal with NEWSHELL and RESIDENT later in this chapter, and, as GETENV and SETENV are related, I will discuss the remaining commands in alphabetical order before tackling these two.

Like IF and LAB, ASK is designed specifically for command scripts. This command asks you to enter a Y or N, and lets you test the response. The template for ASK is:

```
PROMPT/A:
```

The PROMPT parameter is a string that is output to the console window. Normally, you would include a question in this string. Here is a short script that asks if you want the output of the TYPE command to go to the output window (the default) or to your printer. If you do not specify a file to be typed on the command line, the script types the mountlist file:

```
.KEY filename
.DEF filename "DEVS:mountlist"
ASK "Do you wish to send the <filename> to the printer? "
IF WARN
TYPE <filename> TO PRT:
ELSE
TYPE <filename>
ENDIF
```

In the above script, ASK outputs its prompt and waits for you to enter a reply. If you enter anything but Y, N, or Return, ASK redisplays

the prompt. If you enter Y, ASK sets a return code of 5. If you enter N or Return, ASK returns a 0. Thus, you can use WARN to test the user response and take action based upon that response.

The AVAIL command provides information about the available memory in your system. It does not have a template, but instead returns the following usage message:

```
Usage: avail [CHIP|FAST|TOTAL]
```

AVAIL is most useful without parameters. If you enter:

```
AVAIL
```

AmigaDOS responds with something like this:

Type	Available	In-Use	Maximum	Largest
chip	379680	143552	523232	377688
fast	1944824	669184	2614008	1449264
total	2324504	812736	3137240	1449264

This is the ouput of AVAIL on my 3MB Amiga 2000. The report you get will take the same format but will differ in its particulars.

AVAIL reports, in bytes, the condition of the two types of memory in your system, and of the total of the types. (See Chapter 12 for an explanation of chip and fast RAM.) It reports the amount of memory available (not yet assigned to any program), the amount in use, the total of these two, and the largest contiguous segment available. The parameters CHIP, FAST, and TOTAL force AVAIL to return information on the available memory of the type indicated.

The next 1.3 command is FF. This command was developed by Charlie Heath of Microsmiths, an active Amiga developer and booster. FF stands for FastFonts, and its purpose is to speed up text output to

console windows. This command should be in your startup-sequence file. Entering FF without parameters:

```
FF
```

or with the -0 parameter:

```
FF -0
```

enables text speedup. Entering:

```
FF -n
```

turns off fast text handling. I do not see why you would ever want to use the latter option.

Locks, Clocks, and REMRAD

As I mentioned in Chapter 3, AmigaDOS 1.3 includes a FastFileSystem (FFS) that increases greatly the speed of hard-disk access. The LOCK command can be used only with disks that are mounted using FFS. LOCK lets you write-protect FFS disks and partitions.

The template for LOCK is:

```
Drive/A,On/S,Off/S,Passkey:
```

The DRIVE parameter is the hard-disk drive or partition you wish to protect. ON enables write protection, and OFF turns it off. PASSKEY is a four-character password you can specify when you use ON. If you use a PASSKEY when you enable write protection, you will have to use the same PASSKEY to turn off write protection. For example, if you enter:

```
LOCK Fast: ON tttt
```

to enable write protection on an FFS volume named Fast:, you will have to enter:

```
LOCK Fast: OFF tttt
```

to enable you to write to the volume again.

One problem with the Amiga is that it contains two different clocks:

the system clock found on all models and the real-time clock found in Amiga 2000s and in Amiga 500s equipped with the A501 memory/clock card. Often, you find that these clocks have completely different dates and times. SETCLOCK is a command that lets you synchronize your system clock with your real-time clock.

SETCLOCK does not return a template. It takes two parameters, LOAD and SAVE. The LOAD parameter lets you load the contents of the real-time clock into the system clock. You should include this command in your startup sequence. The SAVE command sets the contents of the real-time clock from the system clock. Thus, to set the real-time clock to the current date and time, you first use the DATE command to set the system clock from the keyboard, and then enter:

```
SETCLOCK SAVE
```

Because the real-time clock has a battery backup and thus continues to work after you power down your computer, you need only set the real-time clock once, and then use it to set the system clock each time you reset your computer.

The REMRAD command is used in conjunction with the RAD: device discussed below. It takes no parameters and displays no template. Its function is to remove the RAD: device from the system and return its memory to the system.

Environment Variables

SETENV and GETENV are two new commands that let you establish environment variables. These variables are designed to be used with the ENV: device, which was not ready for the first 1.3 release. For now, the commands manipulate environment variables stored in the directory RAM:env, which is assigned the name ENV:.

Environment variables are similar to the variables you use in script files, with the added bonus that they are available to any AmigaDOS

command, and not simply to a single script file. You set up your environment variables with SETENV and retrieve them with GETENV.

SETENV has the following template:

```
NAME/A,STRING:
```

NAME is the name of the variable, and STRING is its contents. Once you have set a variable, you retrieve its contents with GETENV. Its template is:

```
NAME/A:
```

To demonstrate SETENV and GETENV, enter the following:

```
SETENV myfile "df0:temp.test"
GETENV myfile
```

AmigaDOS responds:

```
df0:temp.test
```

In the future, when the ENV: handler is released, you will be able to use environment variables with many AmigaDOS commands. Until that time, you will have to be content with simulating their functions.

New 1.3 Devices

Through AmigaDOS 1.2, the MOUNT command has been used primarily to add external hardware devices to the system. The 1.3 mountlist, however, contains some software devices that greatly add to the versatility of your Amiga. These devices are AUX:, NEWCON:, PIPE:, RAD:, and SPEAK:. Unlike hardware devices, which add new hardware to your system, these devices define new functions for existing hardware.

AUX: lets you attach a terminal to your Amiga serial port. By mounting this device, you turn the serial port into an unbuffered input/output device. Then, by issuing the command:

```
NEWCLI AUX:
```

you can control an Amiga CLI process from a terminal attached to the serial port. AUX: turns your Amiga into a multiuser machine.

NEWCON: is an improvement on the Amiga CON: device. NEWCON: is a console handler that supports command-line editing and command history. I will discuss these features in detail in conjunction with the Shell, which uses the NEWCON: handler.

The PIPE: device was developed by Matt Dillon and used by Commodore with his permission. It lets you use the output from one program as the input to another. When you designate a program's output destination as a PIPE: file, the program can write up to 4K of information to the file before output is blocked. When another program uses the same PIPE: file for input, the write block is removed until the 4K buffer is full once again. Thus, a program can read the information from the PIPE: file as fast as it as written. The buffer is used only when needed.

To demonstrate PIPE:, let's assume you want to transfer the output of a directory listing of your entire hard disk directly into your WordPerfect (WP) word processor for formatting and eventual printing. You would enter:

```
RUN DIR >PIPE:myfile HardDisk: OPT A
WP PIPE:myfile
```

Note that DIR is started in a separate process from WP. This prevents DIR from tying up the CLI until WP has read the entire PIPE: file.

By providing direct access between applications, PIPE: eliminates the need for temporary files.

Surviving a Warm Boot

The RAD: device in the 1.3 mountlist is a recoverable-RAM: disk. This disk has all the speed advantages of a RAM: disk, and one thing more: it preserves its contents through a warm boot. Thus, if you encounter a guru-meditation error or accidentally enter Ctrl-Left Amiga-Right Amiga, you will not lose the files you stored in the RAD: disk. You lose the contents of RAD: only if you turn your computer off or issue the REMRAD command. Unlike the RAM: disk, RAD: must be mounted. If you examine its mountlist entry, you will see that it simulates

a standard 3½-inch floppy drive, using 512-byte sectors and 11-sector tracks (or cylinders). The mountlist specifies a LowCyl and a HighCyl entry for RAD. Thus, RAD: has a fixed size; it does not expand and contract as does RAM:. Also, because it is a mounted device, RAD: can use the FastFileSystem. If you change the mountlist to use FFS with RAD:, however, you must add the following to the RAD: mountlist:

```
GlobVec = -1
FileSystem = I:FastFileSystem
```

and then format the drive after mounting it. See the section in Chapter 8 entitled "Using a Recoverable RAM: Disk" for an example of a startup sequence that initializes RAD: if it does not exist, and recognizes it if it does.

The last new 1.3 device is SPEAK:. Like the Workbench Say utility, SPEAK: uses the Amiga's built-in speech synthesis to output text as speech. Unlike Say, however, which can take its input only from the keyboard, SPEAK: can take input from any program that can write to a file because it is a device. For example, to listen to a directory listing, enter:

```
MOUNT SPEAK:
DIR >SPEAK: Df0:
```

You can specify output options with SPEAK:. These determine the qualities of the output. Here are the SPEAK: options:

P### pitch (where ### is from 65-320)
S### speed (where ### is from 30-400)
M male
F female
R robot
N natural
O0 do not allow these options in the input stream
O1 allow these options in the input stream
A0 turn off direct phoneme mode

A1 turn on direct phoneme mode (do not use translator.library)
D0 break up sentences on punctuation alone
D1 break up sentences on punctuation, Return, and Linefeed

SPEAK: uses the OPT keyword to access these options. For example, if you want your Amiga to speak a file called myletter at speed 280 with a robot voice, you would enter:

```
TYPE myletter to SPEAK:OPT R S280
```

Improved Commands

AmigaDOS 1.3 supplies improved versions of many commands. I provide a complete explanation of every 1.3 command in Appendix A. Here, I will concentrate on the most important changes.

AmigaDOS 1.3 has added four new protection bits—H, A, P, and S—that you set with the PROTECT command. The P flag is the most important. It stands for pure, and indicates that a command can be made resident using the RESIDENT command (see below).

The A (for archive) flag lets you, or a disk-backup utility, mark a file that has been backed up. The S or script flag indicates a file is a command script. When using the Shell, you can execute such a script file simply be entering its name, without having to explicitly use the EXECUTE command.

The H bit stands for hidden, making you think it would hide a file name so the name would not show up when you ran DIR or LIST. This, in fact, is its stated purpose, but like the old W, R, and E flags, the H flag is not functional as of this writing.

The COPY command has a new parameter to work with protection bits. NOPRO indicates that a file's protection bits are not to be copied. COPY has three other new parameters: DATE copies a file's date and time stamp, COM copies its filenote, and CLONE copies everything—protection bits, date, and filenote.

Better Lists

The 1.3 LIST command has many new options. With the FILES and DIR parameters, you can list only files or directories, respectively.

BLOCK reports file sizes in disk blocks instead of bytes. You can also now use wildcards to specify the files you want to list.

The biggest addition to LIST is the LFORMAT parameter. Commodore provided this as a simple way to construct script files. The best way to explain this option is to use an example.

Let's say you wanted to add a .bkup suffix to all the files in a data directory. You can use LIST to create a script file for this purpose by entering the command:

```
LIST >S:bkup.script df1:data/#? LFORMAT = "RENAME %S df1:data/%S.bkup"
```

To execute this script, you enter:

```
EXECUTE bkup.script
```

and every file in df1:data now has a .bkup extension.

The first thing LIST does is to redirect its output to a file called S:bkup.script. Then, using wildcards, it selects every file in the directory df1:data for listing. Now we come to LFORMAT, which defines a string into which the listed files will be embedded. In this case, the string is "RENAME %S df1:data/%S.bkup." %S is a special character that, when used with LFORMAT, means "insert the item to be listed here." Thus, in the output file, each %S is replaced with the name of a file. If df1:data contains a file called myletter, it would create a line in bkup.script that reads:

```
RENAME df1:data/myletter df1:data/myletter.bkup
```

Each line of the output file will be a command that renames a file in df1:data. When you execute the output file, the files are renamed. LIST with LFORMAT is a powerful script generator.

Disk Workings

The 1.3 FORMAT command has new options for working with hard disks. Although located in the System drawer, FORMAT is often called from the CLI. Its new parameters are QUICK, FFS, and NOFFS. QUICK formats only the root block, boot block, and bitmap block of a disk. This saves time when you are reformatting a hard disk. FFS

and NOFFS let you override the mountlist and indicate whether or not a drive will use the FastFileSystem.

ADDBUFFERS has been improved so that it now works with FFS drives. INSTALL has two new parameters: NOBOOT will make a disk unbootable while keeping it a DOS disk and CHECK will report a return value of 5 if a disk does not have a standard Commodore-Amiga boot block. This can help you detect viruses.

Many other commands have minor improvements. DELETE will no longer abort when using wildcards if one deletion fails. DIR has synonyms for OPT I and OPT A: INTER and ALL, respectively. NEWCLI now uses a default startup file—S:CLI-Startup—if you do not specify such a file on the command line. SEARCH has some new options that make it more useful in command scripts. DISKDOCTOR simply works better. Check the templates in Appendix A for these and other changes.

Shell and RESIDENT

For AmigaDOS users, one of the biggest changes in 1.3 is the inclusion of the Shell as an alternative to the CLI. Like the CLI, the Shell is a program that lets you communicate with AmigaDOS by issuing AmigaDOS commands. The Shell is found in the root directory of Workbench 1.3; you run it by double-clicking on its icon. From the Shell or the CLI, you can start another Shell by entering:

```
NEWSHELL
```

The Shell has many advantages over the CLI, including command-line editing, command history, and, most importantly, the ability to load and use memory-resident commands.

If you have used the CLI for a time, you have at some point (probably at many points) made a mistake when entering a command. You then either used the destructive backspace to erase the line to a point before the error and retyped the line again, or you hit Return, let AmigaDOS tell you the error (which you knew about already), and retyped the

line. Because the Shell uses the NEWCON: console device, you have an alternative. You can use the Right- and Left-arrow keys to move along the command line (when shifted, these bring you to the end and beginning of the line, respectively), delete characters with Backspace and Delete, and insert characters by simply typing them in. In effect, you can edit the command line, then use Return to send the line to the Shell, regardless of whether you are at the end of the line or not.

Command history makes the Shell even easier to use. The NEWCON: device has a 2K buffer that it uses to store commands entered at the keyboard. Pressing the Up-arrow key once moves you backwards in the buffer and makes the last line you entered appear under your input cursor. Pressing this key again retrieves the next previous line. To move in the other direction through the buffer, use the Down-arrow key. Shift-Up arrow moves you to the top of the buffer, while Shift-Down arrow moves you to the bottom. When you retrieve a line from the command-history buffer, you can edit that command, or simply hit Return to execute it.

The greatest advantage of the Shell, however, is its ability to use memory-resident commands. As I have noted, AmigaDOS commands are not built into the operating system. They are separate programs that must be loaded from disk before they can be run. This makes AmigaDOS inherently slower than other operating systems that have built-in commands. Through AmigaDOS 1.2, people have used RAM: disks to hold AmigaDOS commands and speed up command execution; it takes far less time to load a command from RAM: than it does to load one from a floppy or hard drive.

The RAM: solution, though good, is inefficient in its use of memory. You first load the commands into RAM:, which takes memory away from the system; then you must load the commands from RAM: into memory in order to execute them. If you have two CLIs open, and you issue the same command in both, each will load a separate copy of the command from RAM: into memory. You now have three copies

of the command occupying system memory: one on the RAM: disk and two in main memory. This is a waste of a precious resource.

Commodore has provided the RESIDENT command with AmigaDOS 1.3 to solve the problem of loading commands from disk. RESIDENT loads a command from disk and binds it into the operating system. The command is now always in memory and can be used by as many AmigaDOS processes as you have open. If you have made DIR resident, and have issued DIR commands from three different Shells at the same time, each will use the memory-resident copy of DIR. There is no slowdown in having to search through the current AmigaDOS path and load the command from disk, and no wasted memory space. This is the ideal solution.

The template for the RESIDENT command is:

```
Name,File,Remove/S,Add/S,Replace/S,Pure/S,SYSTEM/S:
```

When run without parameters, RESIDENT returns a list of all the commands that are currently resident. It also returns a usecount. As Commodore has not yet finalized how usecount will appear, you may discover that my explanation is incorrect. As of this writing, you cannot remove a resident command whose usecount is not zero.

To make a command resident, you simply move into the C: directory with CD and enter the command name. For example:

```
CD C:
RESIDENT COPY
```

makes the COPY command resident. If you want to give COPY another name when you make it resident, enter:

```
RESIDENT MYNAME COPY
```

or, using the keywords from the template, enter:

```
RESIDENT NAME MYNAME FILE COPY
```

These last two examples both make COPY resident, although you now must refer to it as MYNAME.

The REMOVE parameter lets you remove a command from the resident list and return the memory it occupied to the system. The

ADD parameter is the default. REPLACE will replace a name on the resident list. For example, if after making COPY resident and calling it MYFILE, you enter:

```
RESIDENT MYFILE DIR REPLACE
```

the name MYFILE will now refer to DIR and not to COPY.

One restriction on commands you make resident is that they must be pure. A pure command is one that is reentrant and reexecutable. Reentrant means that the execution of the command can be interrupted at any point and restarted from the same point later. Reexecutable means that once it has finished executing, a command can clean itself up and be ready to execute again. Both properties are vital to commands that are going to be shared by multiple processes.

If a command is pure, the P flag is set in its protection bits. You can discover which commands are pure by listing the contents of the 1.3 C: directory. If a command is not listed as pure but you want to make it resident, you use the PURE parameter. This forces RESIDENT to make the command resident. If you do this, you should then run the command twice in a row. If you don't crash your system, the command is probably pure.

The final RESIDENT parameter is SYSTEM. You use this to make resident code segments that you do not call from the Shell. In the standard 1.3 startup-sequence file, for example, you will find the command:

```
RESIDENT CLI L:SHELL-SEG SYSTEM PURE
```

This line loads the program L:SHELL-SEG and names it CLI. It then uses the SYSTEM option to ensure that the user can never call this code directly. It also uses the PURE option as a precaution; L:SHELL-SEG is pure as listed on Workbench 1.3. This code is used to create other Shells and CLIs.

At this writing, it is not known if commands made resident with the SYSTEM option will be listed in the RESIDENT usecount list.

Conclusion

Operation of the Amiga is enhanced greatly by OS 1.3. Of particular interest to AmigaDOS users are the new commands and devices, the improvements to familiar commands, and especially the new Shell interface and memory-resident commands. Better master the new commands quickly, however, because Commodore will probably release OS 1.4 sometime in mid-1989.

11 Software in the Public Domain

In this chapter, you will learn about some of the useful public-domain and shareware programs available for the Amiga and how you can acquire some of this software.

If you're on a budget, you should be aware of the thousands of free programs available for the Amiga. These programs have been written by Amiga enthusiasts and placed in the public domain. The programs include utilities, games, paint software, programming languages, and more. They are an inexpensive alternative to commercial software.

Free Software

When an author places a program in the public domain, he or she explicitly forfeits all rights to control distribution of the program. As a result, anyone is free to copy the program and use it. Some authors stipulate that you can copy and use a program as long as you do not pretend you wrote it or try to make money from it. As you can imagine, this stipulation is difficult to enforce. In effect, public-domain software comes with no strings attached. The authors give it away because they think others will use and enjoy it.

Because much public-domain software is programming utilities and examples, many authors release their source code along with the executable program. Others prefer to keep their source code to themselves and release only the compiled version of the program. If you are learning to program the Amiga, public-domain source code provides useful examples and handy routines or tools you can incorporate into

your own programs. Thus, public-domain software is useful not only for what it does, but also for the programming examples it contains.

Not-So-Free Software

Shareware is an offshoot of public-domain software. The shareware concept lets an author make money on a program that, like public-domain software, is freely redistributable. When authors release shareware products for free distribution, they include a notice in the program, stating that if the user finds the program useful, the author would appreciate a donation to be sent to the address listed in the notice. In effect, the author relies upon the conscience and good will of the user.

Different authors have varying degrees of success with shareware. The most successful usually provide added incentives, such as a printed manual and automatic notification of upgrades and bugs, when you send them your donation to register the product. Without an incentive to register, users often treat shareware as public-domain software.

If you find a shareware program useful, I encourage you to make a donation to the author. Shareware and public-domain authors (they are often one and the same) are a valuable part of the Amiga community, and the service they provide is invaluable. We should encourage them to keep it up.

Where to Get Public-Domain Software

You can find public-domain software in many places. One of the best is at an Amiga users' group meeting. For your convenience, I have included a list of Amiga users' groups in Appendix C. You can also obtain public-domain software from your Amiga dealer, from companies that sell the disks at a nominal price, and by downloading it from electronic bulletin-board systems (BBS) and commercial telecommunications networks.

The last two sources are especially important because you can communicate with a BBS or a network from anywhere in the world. These sources are also usually the first to introduce new public-domain software to the Amiga community. Thus, they have become clearing houses for freely distributable software.

Bulletin-board systems are run by many Amiga enthusiasts who

devote a computer and a modem to the BBS, as well as some considerable personal time to developing and maintaining its operation. The bulletin board itself is a software program that lets modem-equipped callers read messages, post bulletins, and download software. Because BBS numbers change frequently, I have not included a list in the book. Your dealer will normally know about BBS systems in your area.

Although most BBS systems are free, they do not provide the range of options and services available on commercial networks such as People Link, CompuServe, BIX, and GEnie. These four services in particluar have active Amiga sections. Most also have an on-line meeting area where you can communicate with dozens of other Amiga owners. Finally, these services normally have hundreds of different files available for downloading.

Most public-domain sources have the most popular software. In the following sections, I describe what I consider the most useful Amiga public-domain programs. I invite you to try these and other programs in the public domain. It is well worth the effort.

Public-Domain Terminal Programs

Because telecommunications is such a popular way to obtain public-domain software, it is only appropriate that we examine public-domain telecommunications software first.

A modem is a hardware device. To control it with your Amiga, you need special software called a telecommunications or terminal program. There are several good telecommunications packages available in the Amiga public domain. Five of the best are VT100, COMM, ACCESS!, ACO, and CoCOMM. Any one of these terminal programs will dial a number, make a connection, and upload or download files. Beyond the basics, each program has an area in which it excels.

COMM and VT100 are basic terminal programs; they have numerous features but employ a "clean screen" look that some people mistake for simplicity. COMM is a straightforward, general-purpose package, while VT100 lets your Amiga emulate the popular VT-100 terminals

from Digital Equipment Corporation. The special features of these programs are accessed with pull-down menus or Amiga-key commands.

ACCESS!, ACO, and CoCOMM are large programs that use more of the color and graphics capabilities of the Amiga. If you can spare the memory space, you might want to use one of these instead of more basic terminal programs.

ACCESS! is a very colorful extension to the COMM program, and performs many of its basic functions with a real flair. You can use it with any BBS or commercial network. Also based on COMM, ACO provides a graphics-based interface for the Amiga Zone of the People Link network. CoCOMM provides similar facilities for accessing the CompuServe system. If you're a People Link or CompuServe user, you should try one of these special-purpose terminal packages.

Archive Utilities

When you try to download an Amiga program from a BBS or a network, you will notice that nearly every file available for downloading has the extension .arc, which means the file has been compressed using the ARC (for archive) utility. To use the program after you have downloaded it, you must decompress the file, using either ARC or UNARC, a subset of the complete ARC program.

ARC not only compresses files but also combines related files into one large file. Thus, if a game you want to download consists of an executable file, a source-code file, a documentation file, a picture file, and some sound files, you can probably download the whole thing in one big archive file and use ARC or UNARC to unpack the individual files. A copy of ARC or UNARC is vital for downloading software.

Help for Console Windows

Before the advent of Workbench 1.3, the only console devices available for the Amiga were CON: and RAW:. CONMAN is a public-domain console device for the Amiga that provides additional capabilities to all console windows. For example, CONMAN provides command history and lets you edit the command line. CONMAN is quite a time saver.

Another enhancement for console windows, BLITZFONTS, speeds up text output to the console device. This utility is usually most effective when you use the standard TOPAZ system font.

Help for Hurt Disks

An alternative to DISKDOCTOR, DISKSALV can often recover files from a disk that AmigaDOS cannot validate and therefore cannot read. This may happen if you accidentally pop out a disk while the disk-activity light is still on. The advantage to DISKSALV is that it does not write on a hurt disk. It simply reads it (in df0:) and then copies all files that it finds to a formatted disk (in df1:).

As a side benefit, if you have accidentally deleted a file, but have not yet written anything to that disk, DISKSALV also recovers deleted files. Another utility, UNDELETE, also recovers a deleted file. DISK-SALV recovers all intact deleted files, however, whether you know they are there or not. If you have a two-floppy system, DISKSALV can be a real "lifesaver."

Cutting the Cord

RUNBACK is a smaller and better version of RUNBACKGROUND, a program I wrote in the early days of the Amiga. I wanted to be able to run programs (such as Clock) from my startup sequence and still allow my initial CLI window to close down at the end of the startup sequence. This would leave me with a normal-looking Workbench, but with the Clock running. When I used the RUN command,however, the CLI window would hang around until the Clock was shut down.

RUNBACK breaks the connection between a CLI and programs spawned by that CLI, allowing you to close the CLI window without having to quit all the programs you started from it. RUNBACK is particularly helpful in running the virus-detector program VIRUSX from your startup sequence.

Picture-Perfect Viewers

Although you may think you have to boot up DeluxePaint II or some other paint program every time you want to display a picture on your Amiga, DPSLIDE allows you to display a sequence of Amiga pictures.

You specify the names of the pictures you want to see in a text file. Long in the public domain, DPSLIDE has many useful features. This program is a must.

Another program that displays Amiga pictures is USHOW2. Although less than 1000 bytes, it can display almost any Amiga picture—lo-res, hi-res, interlaced, HAM—with the exception of overscanned pictures (those that extend beyond the normal 320-pixel-wide lo-res or 640-pixel-wide hi-res limits).

Another picture-display program, SUPERVIEW, recognizes "author chunks" and displays them in the originating CLI window. Unlike USHOW2, it does allow you to display overscanned images.

While the three programs above are geared to viewing static Amiga pictures, other public-domain offerings allow you to play animations created by a variety of commercial animation packages. Because animation files, unlike picture files, are not easily interchangeable, you need different programs to view the output from different Amiga animation packages. ANIPLAYER allows you to view an animation produced with Aegis Animator or Silver. With SHOWANIM you can run VideoScape 3D animations, while MOVIE plays animation sequences created by Sculpt 3D and Animate 3D.

Workbench Utilities

Several public-domain utilities provide neat additions to the Workbench interface.

DROPSHADOW causes Amiga windows to cast a transparent shadow on any windows running underneath. Although it tends to slow down window movement, DROPSHADOW is a great "show off" program.

RSLClock produces a small, movable clock in the title-bar area of the current screen. You can also indicate a program you want to run when the clock reaches a certain time. RSLClock is more than just a clock, however. If you have not hit a key or moved the mouse in a while, RSLClock's screen dimmer will blank your display screen. The program also includes a utility that displays free memory and disk space. Check this one out.

MACH integrates the functions of several smaller public-domain

programs. One of these, ClickToFront, lets you click anywhere in a window in order to move that window to the front of the screen. Another, SunMouse, causes the window on which rests the mouse pointer to become the current window, without the user having to click the left-mouse button. MACH also installs a mouse accelerator that makes your mouse pointer move more quickly.

Other utilities integrated into MACH include a money meter that tells you how much a session on a telecommunications network is costing you, a program called PopCLI that makes a CLI available when you press a certain key combination, and a program that lets you shuffle among numerous Amiga screens. There is even more that MACH can do, but this selection should give you some idea of the versatility of this utility

Startup-Sequence Shortcuts

Although you can, of course, run any program from your startup sequence, the following utilites are especially useful because they can produce some very useful effects when run automatically at startup.

ASSIGNDEV is very useful to Amiga 2000 owners. Because the Amiga 2000 can have two internal floppy dirves (df0: and df1:), the first external drive is called df2:. Because of shortages of internal drives, most Amiga 2000 systems have one internal and one external drive. The problem is that some software will not recognize df2:. ASSIGNDEV reassigns df2: as df1:, thus letting you run software that balks at df2:.

MAKEACV and LOADACV are useful utilities if you like to copy lots of files into a RAM: disk at startup. MAKEACV packs the files together into a single file that you copy to RAM:. LOADACV then unpacks them. Using these two utilities is faster than copying the files individually to RAM:.

Another utility that can save a lot of typing in your startup sequence is DEFDISK. This program changes the default disk of all system directories to one individual disk, such as RAM: or a hard disk. Running this program has the same effect as loading and executing the

AmigaDOS ASSIGN program seven times. The directories that DEFDISK reassigns are SYS:, C:, L:, S:, DEVS:, LIBS:, and FONTS:. To use this program, you must be certain that equivalent directories exist on the target disk.

VD0: (usually found in a file called ASDG.ARC) is a public-domain version of the recoverable RAM: disk found in Workbench 1.3. If you move a lot of files to RAM: at startup, you should use a recoverable RAM: disk to protect yourself from power interruptions, guru errors, and accidental warm boots.

Directory Utilities

There are dozens of public-domain directory utilities. Their purpose is to make much of the power of the CLI available to people who use the Workbench environment exclusively. As this book has given you a firm foundation in using the CLI, you should have no great need for directory utilities, but they can be useful if you are not a particularly good typist.

Directory utility programs that I have found useful include BROWSER (designed for programmers), CLIWIZ, DU, DISKMAN, M2DU300, and UTILIMASTER. Most of these programs make CLI commands available through buttons you select with your mouse. These program are handy in copying lots of files from one disk to another.

Fun and Games

Two of the most popular games in the Amiga public domain are HACK and Amoeba Invaders. In HACK, a text-based game popular on UNIX systems, you explore a dungeon, while Amoeba Invaders is a Space Invaders clone.

Perhaps the most entertaining programs in the Amiga public domain are not games at all. Many people, particularly Leo Schwab, have become notorious for writing "display hacks." These programs do unexpected things to your Amiga screen display. I won't spoil your

surprise by telling you what these programs do; you'll just have to get ROBOTROFF, MELT, and the others to find out for yourself.

Odds and Ends

CLIP-IT is a nifty program that lets you use the mouse to draw an invisible box around any rectangular region on any Amiga screen, and to then save the contents of the box as a file that can be read by an Amiga paint program. A similar program called SNIP-IT lets you grab lines of text from one console window and dump them into another. TASKX lets you view the list of tasks currently running and change their priorities.

Last But Not Least

Perhaps the most important programs in the public domain are virus-checking programs. The best of these are VIRUSX and VIRUSCHECK. I discuss viruses in detail in the next chapter.

This chapter has presented a small sampling of the programs available in the Amiga public domain. I hope you enjoy using the programs I've recommended, and that you take the time to discover your own gems.

12 Amiga Answers

This chapter answers some of the most commonly asked questions about the Amiga. From the Amiga 500 to Zorro 2, this chapter fills the gaps in your Amiga education.

While preparing this book, I was struck by the similarity of questions asked by people just starting out with their Amigas. In this chapter, I provide answers to these commonly asked questions. I hope the answers help you understand more about your fascinating computer.

Q What are the differences between the Amiga 500, 1000, and 2000?

A The A1000 was the first member of the Amiga family. The A500 and the A2000 evolved from and improved upon the original design. Here are a few of the differences:

1. Kickstart

When you first turn your Amiga on, it must be able to access the fundamental routines of the operating system, the ROM Kernel routines. With the A1000, these routines are loaded from a special disk called the Kickstart disk. After loading the ROM Kernel routines, the Amiga 1000 prompts you to enter a Workbench disk.

In place of a Kickstart disk, both the Amiga 500 and 2000 store the ROM Kernel routines on a ROM (Read-Only Memory) chip. When you turn one of these models on, the ROM Kernel routines are already available and the computer prompts you for a Workbench disk immediately.

Once the operating system is up and running, the A500, A1000, and A2000 operate identically. They all run the same software.

2. Keyboard

The A2000 and A500 have full-sized numeric keypads. That is, the arithmetic operators found on the shifted number keys are duplicated on the numeric keypad. The numeric keypad on the A1000 does not have the arithmetic operators.

3. Ports

On the Amiga 1000, power is available from the serial and parallel ports. You therefore have to use special cables to connect the A1000 to printers and modems. The Amiga 500 and 2000 have standard IBM PC-type ports. This makes it easy to connect them to peripheral devices.

4. Memory

The Amiga 1000 has a maximum internal memory capacity of 512K (256K on the motherboard and 256K on a front-panel expansion card). Additional expansion capability is provided by an expansion bus on the right side of the machine. The A1000 is designed to expand to 8.5MB (8.5 million bytes).

The Amiga 500, has a maximum internal memory capacity of 1MB (512K on the motherboard and 512K on the internal memory/clock card). Additional expansion capability is provided by an expansion bus on the left side of the machine.

The Amiga 2000 contains 1MB of memory on the motherboard. Further memory expansion is provided by slots into which you can place expansion cards. Both the A500 and the A2000 can normally be expanded to 9MB.

5. Expandability

The Amiga 2000 is an open-architecture system. It contains five slots that you can fill with expansion devices. These include hard-disk drives,

networking connectors, and memory cards. The A2000 can use a special board called a bridge card that lets you run IBM PC software on your Amiga. The A2000 also has a video slot that can be used to house advanced video-processor boards and a coprocessor slot that lets you install faster processors that either supplement or replace the onboard Motorola 68000 processor.

The A1000 and the A500 are not as easy to expand as the A2000. They rely upon expansion devices that connect to their external expansion buses. This is not as reliable an expansion solution as internal slots.

6. Custom Chips

All Amiga models rely upon three custom chips to provide their advanced graphics capabilities. These chips are Agnus, Denise, and Paula. Although functionally identical, the Agnus on the A500 and A2000 is physically different from the Agnus on the A1000. If it has not done so already, Commodore plans in the near future to release new versions of Agnus and Denise that will expand the amount of chip memory available to the system and provide a non-interlaced hi-res graphics mode. Because of differences in the packaging of the Agnus chip between the A2000/A500 and the A1000, it is not possible to upgrade the A1000 with the new chip set.

7. Video Capabilities

The A1000 puts out both RGB and color NTSC video signals. Thus, with the help of a simple RF converter, you can output the video from an A1000 to your television set. The Amiga 500 and Amiga 2000 put out RGB video and monochrome NTSC. To record the color output of these machines on videotape you will need an RGB encoder device.

If you need more information about the features of the different Amiga models, see your Amiga dealer.

Q How can I get my Panasonic, Toshiba, or other non-Preferences printer to work with my Amiga?

A Although your printer is not listed directly, it probably emulates a printer that is listed by Preferences. Check your printer manual or contact the printer manufacturer to find out if your printer emulates an Epson or some other Preferences printer. Then use the emulated driver to run your printer.

If your printer does not emulate a Preferences printer, contact a local Amiga users' group or a computer bulletin board near you that specializes in the Amiga. It is probable that someone has already created a printer driver for for your printer.

As a last resort, you can try to use a public-domain printer driver generator (such as PRTDRVGEN) to create your own printer driver.

Q What is the Amiga Virus and how do I keep from catching it?

A Viruses are nasty little programs that irresponsible programmers create and circulate. They range from the merely pesky to the truly damaging. Thus far, the five viruses discovered on the Amiga are all boot-block viruses.

When your Amiga is first powered up, or when you perform a warm boot (Ctrl-Left Amiga-Right Amiga), it goes through a special initialization sequence in which it reads the first two sectors on the outermost track of the disk in the internal drive. These sectors are called the boot blocks; they contain "hooks" that point to code that lets your Amiga load AmigaDOS.

Unfortunately, misguided individuals have modified the "hooks" and written their own boot code for the Amiga. This new boot code does more than load AmigaDOS; it may cause a message to be printed on your screen or even cause your entire system to crash. This is why

these programs are called viruses: they infect your disks, replicate themselves, and cause a lot of harm.

The most widespread Amiga virus was developed in Europe by the Swiss Crackers Association, and is thus called the SCA virus. When you boot your system with a disk that has an SCA-infected boot block, the virus loads itself into memory and protects itself against a warm boot. Thus, the only way to eradicate a virus from your computer's memory is to turn your machine off.

Once in memory, the virus lies dormant until you warm boot your machine with another disk. Whenever you warm boot, the virus copies itself onto the new boot disk, thus infecting it. This is how the virus spreads from disk to disk.

The SCA virus has an internal counter that is incremented whenever the virus replicates itself. After a set number of disks are infected, the virus prints a message to your screen such as: "Something wonderful has happened—your machine is now alive, and it has spread the news to a lot of your other disks." The message, though disconcerting, is harmless enough. The problem comes when the virus infects copy-protected disks. Commercial software manufacturers often alter the boot blocks of a disk as part of their copy-protection scheme. When a virus overwrites a custom boot block, it usually makes a copy-protected disk—and the programs on it—unusable.

Other viruses do more than simply write messages to your screen. For example, the Byte Bandit locks up your machine so it cannot accept mouse or keyboard input. Other viruses have other harmful or crude effects, and new strains are being developed all the time. The best cure for the virus is prevention.

You can eradicate a virus from a disk with the INSTALL command. INSTALL overwrites the boot blocks of a disk with the standard AmigaDOS boot blocks. INSTALL does not provide complete protection, however, because some viruses detect when you run INSTALL and reinfect the disk immediately afterwards. Your best protection

against a virus is to boot only with disks you know to be virus free. The Workbench disk (and any duplicates you make) that comes with your system should be virus free.

To see if a disk is virus free, you should use a virus-detection program. The two best are VIRUSX and VIRUSCHECK, both available in the Amiga public domain. These programs let you know whether a disk has a virus in its boot block. If you have an infected disk, you should turn your computer off, reboot with a known clean disk, and then run INSTALL on the infected disk. Never cold- or warm-boot your machine with an unknown disk in the internal drive. If that disk is infected, it will enter your computer's memory and infect other disks until you turn the power off.

Q Can I use single-sided disks with my Amiga disk drives?

A Although single-sided disks have been used successfully with the Amiga, I strongly recommend that you stick with double-sided disks. The difference in price is so slight that the savings you accrue using single-sided disks are not worth the chance that you will damage your disk drive or lose valuable data.

Q What does auto-config mean?

A Auto-config describes a scheme that Commodore developed to make expanding your system an easy task. Unlike other computers that force you to run installation programs and set DIP switches when you add a new device to the system, the Amiga makes expansion easy and automatic.

An auto-config (short for auto-configuration) device communicates with your Amiga when you power up your computer. It indicates what

type of device it is and how much memory space it needs from the system. Your Amiga, in turn, checks its memory list and gets the starting address of an unassigned memory space. It then passes the address to the expansion device. The device is now integrated into the system. It will handle all memory accesses to its address space and will ignore accesses to the memory spaces of other auto-config devices.

Q What are the differences between chip RAM and fast RAM?

A Chip RAM is the lowest 512K bytes of memory on your Amiga. (This may soon be increased to the lowest 1MB.) The custom chips that make the Amiga so fast and versatile have direct access to chip RAM, which holds the Amiga screen displays, any sprites that appear on the display (including the mouse cursor), data for generating sounds and music, and buffers for your floppy-disk drives.

Fast RAM is memory on your system that is not accessible by the custom chips. Commodore calls this memory fast RAM because the 68000 processor that drives your system never has to compete with the custom chips for access to fast RAM. Any time the processor accesses fast RAM, it can run at full speed.

There is one section of memory on the Amiga 500 that, although it is not accessible to the custom chips, will still sometimes cause the processor to slow down when accessing it. Located on the A501 internal memory/clock card, this memory is sometimes described as half-fast RAM (grin). When the enhanced chip set becomes available, this memory space on the Amiga 500 will be remapped so that it becomes a part of the chip RAM and Amiga techies will lose one of their favorite puns.

Q What are the Amiga coprocessor chips, and what do they do?

A In addition to the central processor, the 68000, the Amiga uses three custom chips named Paula, Agnus, and Denise. The "P" in Paula stands for "peripheral." It is the job of this custom circuit to handle audio output and interfacing with floppy-disk drives. The main

processor tells Paula where to find audio data or disk data in memory and the size of the data. (With audio data, the processor also indicates the sampling rate.) The audio or disk system in Paula then retrieves data directly from chip memory with no further intervention from the central processor.

The "A" in Agnus stands for "address." Agnus generates the addresses for the custom chips. The "D" in Denise stands for "data." Denise helps to handle the data that the custom chips use. Agnus and Denise together form a "bit blitter." This blitter is much more efficient at moving and modifying data than is the 68000. The blitter is a major reason for the speed with which the Amiga can animate graphics and handle the Intuition windowing interface. Agnus and Denise also see that system memory is constantly refreshed, preventing loss of data. Finally, these chips generate the various Amiga display modes (lo-res, hi-res, interlace, HAM, and so on) and handle sprites—objects that overlay the standard display. (The mouse pointer is a sprite.)

Together the custom chips are called the Amiga coprocessors. They were designed to use the data pathway to and from the chip memory whenever the main processor is not using this path. Because of the unique way the Amiga designers created the coprocessors, they rarely conflict with the operation of the 68000, which can be running at full speed even when the coprocessors are active. Thus, it is possible to run the Workbench display, play four channels of audio, access a floppy disk, have eight sprites bouncing around on the screen, and *still* have the 68000 working full speed on some math-intensive function. This demonstrates the advantage of the Amiga's coprocessors.

Q What is an interlaced display?

A Normally, the Amiga display is 200 pixels high. The Amiga outputs this display 60 times per second so that the display appears to be steady. Interlacing doubles the vertical resolution of the Amiga display to 400 pixels. The price of this increased resolution is a slower refresh rate. The Amiga refreshes an interlaced display only

30 times per second. Because your eye is not fooled by this low refresh rate into believing that the display is steady, an interlaced display usually has an annoying tendency to flicker.

Q What is Zorro 1 and Zorro 2?

A The terms Zorro 1 and Zorro 2 refer to different versions of the Amiga auto-config expansion specifications. The expansion specifications outline the size of Amiga expansion boards and, more importantly, describe the signals used to connect the boards with the Amiga. Zorro 1 boards are square and are designed for older Amiga 1000 expansion chassis. Zorro 2 boards are oblong and designed for the Amiga 2000. Because Commodore never released any Zorro 1 products, Zorro 2 is considered the current standard for Amiga expansion devices.

Q What is the difference between a digitizer and a genlock?

A A digitizer is a piece of hardware that can transform a picture from a television camera into an Amiga graphics picture that you can display on your Amiga, store as a picture file, print with a screen-dump program, and modify with a paint program.

A genlock is a device that synchronizes the Amiga display to an external video signal. Once synchronized, Amiga-generated graphics can be combined with external video, with the resulting signal sent to a television monitor, a videotape machine, or a broadcast antenna. Genlocks let you overlay Amiga graphics over a video picture, just as network

engineers for ABC, NBC, and CBS can overlay scores from other games on your screen while you're watching a particular football game.

Q What is a device driver?

A A device driver is a set of functions that usually resides in theDEVS: directory of your Workbench disk. The driver is loaded into the Amiga's memory when the MOUNT command is issued, and is specific to a certain piece of hardware. A device driver translates general instructions from a program into the exact actions that must be taken by the hardware. Thus, although the Amiga audio device and the floppy-disk drive both respond to a WRITE command, the device driver for each translates that command into the specific instructions that, in the first instance, output a specific sound to the audio channel, and, in the second, output data to a disk drive.

Q Can the Amiga run software written for other computers?

A The answer is yes and no. The Amiga 2000 certainly can, if you intall an A2088 Bridgeboard from Commodore. This board lets you run MS-DOS software on your Amiga. There is also a program called the Transformer available from Commodore that lets any Amiga run some MS-DOS programs, albeit very slowly.

Don't buy an Amiga simply to run IBM software. The Amiga performs best when running software that was designed exclusively for it.

Q How can I eliminate the flicker that is associated with the Amiga's interlaced, hi-res display?

A The cheapest way to eliminate the flicker is by adjusting the brightness and contrast settings on your monitor. The lower the contrast, generally, the less flicker you will notice. An alternative is to place a specially designed commercial filter in front of your monitor

screen. Another solution is to buy a monitor that has high-persistence phosphors.

The best—and most expensive—solution is available only to Amiga 2000 owners. The flickerFixer is a board from MicroWay that eliminates hi-res flicker on Amiga 2000s. It works only with multiscanning monitors.

Q What is the difference between a screen and a window?

A A screen defines the color palette and the resolution for a horizontal slice of your output display (or perhaps for the entire display). Amiga software produces screens of 2, 4, 8, 16, or 32 independent colors. Screens that can hold 64 or even 4096 colors at once are also possible, although the colors of any pixel in such a display are not necessarily independent of other pixels in the display. Concerning resolution, an Amiga screen can be lo-res (320 pixels across, 200 pixels high), medium-res (640 × 200), interlaced (320 × 400), or hi-res (640 × 400).

Windows are opened on screens and inherit the colors and resolution defined by the screen. Screens, therefore, define the environment that windows occupy. Windows are how programs display their output.

Q Why can I get only four colors in my Workbench?

A Workbench is built on a four-color, medium-resolution screen. This is the maximum resolution that the Amiga can display without slowing the 68000 processor. At this resolution, the processor is able to effectively share chip memory with the custom chips and access the display data during alternate memory access cycles.

Because it is important that the display not jump or blank out, the custom chips have priority over the central processor when it comes to refreshing the display. Therefore, because more colorful and higher-

resolution displays require more work by the custom chips, they take time away from the processor. With a four-color Workbench you get the best of both worlds—reasonably good resolution (640 pixels across) and no interference with the operation of the central processor.

Q What is EXEC, and where is it located in my machine?

A EXEC is the core of the Amiga's operating system. It is a set of software routines located in the system read-only memory of an Amiga 500 or Amiga 2000. On the Amiga 1000, EXEC is loaded from the Kickstart disk, along with the other parts of the operating system, and stored in the Writable Control Store.

EXEC is the part of the Amiga operating system that handles the details of running multiple programs at the same time. With other personal computers, multitasking is either limited or non-existent. The Amiga has had this capability since it was introduced in July of 1985.

Q Can the Amiga use Apple-type proportional joysticks?

A The Amiga supports proportional joysticks but, as of this writing, no software manufacturer has released a program that uses proportional joysticks. Without software that uses it, you have no reason to buy a proportional joystick for your Amiga.

Q What are the advantages of having a hard-disk drive?

A You can store many more programs and data files on a hard disk than on a floppy disk. Each of your floppies holds 880,000 characters of information. A typical hard disk holds 20 million characters of information. Having a hard-disk means you no longer have to search

many disks to find the program you want. In addition, loading and saving your programs is much faster with a hard-disk drive than with floppy drives.

Q What is Intuition?

A Intuition is the name given to the parts of the operating system that make up the visual interface of the Amiga. Intuition creates and manages the screens, windows, gadgets (control buttons, front/back controls, window drag bar, window expansion tool), and other items that comprise the Amiga mouse-driven interface.

Q What is Hold and Modify?

A Hold and Modify (HAM) is a special display mode unique to the Amiga. When using a 320-pixel wide display, the Amiga can display 32 independent colors. Each color has three components, a red value, a green value, and a blue value. The color of any pixel on the display is a combination of red, green, and blue color values.

In HAM mode, the color value of any pixel is dependent upon the color of the pixel to its left. The pixel to the left supplies two of the color values for the current pixel; these values are used to determine the third value.

HAM pictures are more difficult for programmers to create and manipulate, but they are far more realistic than standard pictures. They have many more colors than standard pictures, up to 4096, and the lack of abrupt color transitions lends a nice, anti-aliasing effect.

Q What is the importance of slots and how do they differ from ports?

A A port provides a connection to such peripheral devices as printers or modems. Slots, such as those built into the Amiga

2000, provide a way to expand your system. The difference is more than semantic; peripherals are outside your system, expansion devices are incorporated into your system and have access to the data and address buses.

Slots represent an investment in the future. They can hold cards that add memory to your system, increase the resolution of the output display, and even add a faster, more powerful central processor. Slots, and the expansion devices they hold, can help you keep your machine up with the state of the art: You don't have to buy a new computer, just a new expansion card.

Q Can I speed up the performance of my Amiga?

A Several hardware vendors have created speedup boards for the Amiga. These usually replace the standard 68000 processor with a faster processor and often include a math coprocessor. As of this writing, Commodore is developing its own 68020-based speedup board for the Amiga 2000. It should be available by early 1989.

A Appendix: AmigaDOS Command Reference

This appendix lists the commands contained in the C directory of Workbench 1.3. Each entry consists of the command name, command template or usage (if any), and a quick reference to the meaning of each of the template items. Again, remember that though the command template shows parameters separated by commas, when you enter key words and parameters, you separate them with spaces.

ADDBUFFERS

DRIVE/A,BUFFERS/A:

ADDBUFFERS provides more working space for a floppy-disk drive, thus speeding up AmigaDOS disk activity. DRIVE specifies the drive (df0:, df1:, and so on). BUFFERS specifies how many 512-byte buffers to add (a typical number is 50).

ASK

PROMPT/A:

ASK is used within EXECUTE scripts to output a question to the user. The response must be either a Y or N. If the user presses RETURN, N is assumed. If Y is pressed, an error code of 5 (WARN) is returned. Otherwise, 0 is returned.

ASSIGN

NAME,DIR,LIST/S,EXISTS/S,REMOVE/S:

ASSIGN establishes a connection between a logical name and a path to a directory, file, or device. NAME is the logical name to be used; DIR is the directory to which this name is equated. LIST, if specified, lists all of the items that are assigned. EXISTS searches the assign list for a particular name. If found, a 0 is returned; otherwise, a 5 (WARN) is returned. REMOVE deletes a name from the assign list. Without parameters, ASSIGN returns a list of all assigned names and mounted disks and devices.

AVAIL

Usage: avail [CHIP|FAST|TOTAL]

AVAIL lists the amount of CHIP and FAST memory available in the system, the amount in use, the total amount, and the largest free segment. Note that AVAIL reports its usage as

a reply to the question mark—it expects no parameters and executes immediately on being loaded.

BINDDRIVERS

(no template or usage)

BINDDRIVERS causes AmigaDOS to load the device drivers from the Expansion drawer.

BREAK

PROCESS/A,ALL/S,C/S,D/S,E/S,F/S:

BREAK sends a break to the specified process number. This is normally a programmer function. The effect is as if process receives a CTRL-C, CTRL-D, CTRL-E, or CTRL-F from the keyboard.

CD

DIR:

CD without parameters reports the pathname of the current directory. CD with the DIR parameter changes the current directory to the indicated path.

CHANGETASKPRI

Pri/a,Process/k:

CHANGETASKPRI changes the priority of one of the processes running on the system. PRI is the priority level (– 128 to + 127); PROCESS is the number of the process whose priority is to be changed.

COPY

FROM,TO/A,ALL/S,QUIET/S,BUF = BUFFER/K,CLONE/S,DATE/S,
NOPRO/S,COM/S:

COPY is used to duplicate one or more files from one directory to another. FROM indicates the source directory and file name, perhaps using a wild-card specification for the name. TO indicates the destination directory and, optionally, the file name if only one file is being copied.

ALL copies all files and directories from a source directory to a target directory. If ALL is not specified, but the FROM and TO specifiers are directories, then only the files are copied. QUIET suppresses progress reports. BUFFER tells how much space (in 512-byte chunks) the system can use during copying. Using more buffer space sometimes speeds the copy operation. DATE tells the system to keep the file date intact so the copy will have the same date as the source file. NOPRO tells the system not to copy the protection bits. COM indicates that the filenote is to be copied. CLONE says make the COPY of the file exactly like the original and is equivalent to specifying both DATE and COM on the command line.

DATE

TIME,DATE,TO = VER/K:

DATE with no parameters displays the system date and time. The TIME parameter

(hh:mm:ss) sets the current time. The DATE parameter (DD-MTH-YY) sets the date. If TO or VER is specified with a file name, the system date (or that just set) is written to the file.

DELETE

,,,,,,,,,,ALL/S,Q¹UIET/S:

DELETE removes up to 10 file names or wild-card specifications from disk. ALL removes all files in any directory whose name is specified in the command. DELETE reports its progress unless QUIET is specified on the command line.

DIR

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

DIR outputs a sorted list of the contents of either the current directory or the specified directory. The OPT keyword with option A lists all directories and files below the current directory. Option D lists only directories within the current directory. Option I performs an interactive directory listing. ALL can be used in place of OPT A. DIRS is the equivalent of OPT D. INTER is the equivalent of OPT I. The FILES keyword lists only the file names in a named directory. A COMMAND = option in DIR OPT I or INTER lets you execute almost any AmigaDOS command.

DISKCHANGE

dev/a:

DISKCHANGE signals the operating system that the disk has been changed in a specific drive. Normally this is used to signal a change in a 5¼-inch drive.

DISKDOCTOR

DRIVE/A:

DISKDOCTOR examines an unreadable disk in the specified drive and tries to recover all files on the disk.

ECHO

,NOLINE/S,FIRST/K,LEN/K:

ECHO is normally used from within script files to write descriptive information about the script. NOLINE causes ECHO to prevent output of a line feed at the end of the line. FIRST indicates the first character in the ECHO string that should be printed. LEN indicates the length of the printed string.

ED

FROM/A,SIZE:

ED is the AmigaDOS full-screen editor. FROM specifies the text file to be edited. The SIZE parameter specifies the maximum size the file can become. The default is 40,000 characters.

EDIT

FROM/A,TO,WITH/K,VER/K,OPT/K:

EDIT is the AmigaDOS line editor. FROM specifies the file to be edited. TO specifies the name of the destination file (which will be the same as the source if TO is not specified). WITH specifies a file of EDIT commands that should be used on a line-by-line basis to change the source into the destination. VER indicates the file to which error messages and

"verification copies" of lines that the EDIT commands have affected should be output. OPT specifies the working space that EDIT can use during its execution.

ELSE

ELSE must be in a command file

ELSE indicates those commands to be executed when an IF condition in a command script is false.

ENDCLI ?

(no template or usage)

ENDCLI ends a CLI or Shell task. It executes immediately and expects no parameters.

ENDIF

(no template or usage message)

ENDIF marks the end of IF statement block in a command file.

ENDSKIP

(no template or usage message)

ENDSKIP stops the search of a SKIP command if a label has not been found.

EXECUTE

(no template or usage, expects a file name)

EXECUTE loads and runs AmigaDOS commands stored in the indicated script file. You can pass parameters to the script file on the command line.

FAILAT

RClim:

FAILAT is used within a command script to specify the minimum return code (RClim) at which the script should fail and exit. The default is 10.

FAULT

,,,,,,,,,:

FAULT accepts up to 10 number values (such as FAULT 1 3 22) and, for each, prints a line such as "FAULT 1: ERROR 1". Normally used to output error messages from script files.

FF

(no template or usage)

FF improves the speed of text output. FF –0 turns on FastText; FF –n turns it off.

FILENOTE

FILE/A,COMMENT/A:

FILENOTE attaches a comment to a file. The FILE parameter specifies the name of the file to which the comment is to be attached. The COMMENT parameter is the comment.

You must enclose the entire comment in double quotes or else AmigaDOS will assume only the first word is the comment.

GETENV

NAME/A:

GETENV retrieves the string stored in the environment variable specified by the NAME parameter. This feature was not implemented in prerelease versions of AmigaDOS 1.3.

IF

NOT/S,WARN/S,ERROR/S,FAIL/S,,EQ/K,GT/K,GE/K,VAL/S,EXISTS/K:

IF is used in command files to test a condition. NOT reverses the result of a test. EQ tests for equality between strings. EXISTS tests whether a file exists. WARN, ERROR, and FAIL are used to test the current return code. WARN equals 5, ERROR equals 10, and FAIL equals 20. As of this writing, GT and GE have not been documented by Commodore. They probably test "greater than" and "greater than or equal to" conditions, respectively.

If the tested condition is true, the command script will execute all statements that follow until an ELSE or ENDIF is encountered. If the tested condition is false, the script skips forward to an ELSE or ENDIF statement.

INFO

DEVICE:

INFO provides information about the DEVICE specified on the command line. If no DEVICE is specified, INFO returns information about every device that has been configured into the system.

INSTALL

DRIVE/A,NOBOOT/S,CHECK/S:

INSTALL causes initialization code to be written to the boot block of the specified DRIVE, making the disk bootable. NOBOOT is used to make a disk a DOS disk, though still not bootable. CHECK checks whether a disk has a standard Commodore-Amiga boot block. If it does not, a WARN is returned.

JOIN

,,,,,,,,,,,,,,,AS = TO/K:

JOIN copies up to 15 files into one file that you specify with AS or TO.

LAB

(no template or usage)

LAB is used in command files to specify a label used as the target of a SKIP command.

LIST

DIR,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:

LIST without parameters lists the contents of the current directory. DIR lets you specify another directory to be listed. The KEYS switch turns on display of block numbers. The DATE switch turns on the date display in the listing (DATE is the default). NODATES dis-

ables the display of dates. TO specifies a filename to which you can redirect LIST output. The SINCE keyword, with a date, lists all files created or modified since that date. The UPTO keyword, with a date, lists all files created or modified on or prior to the specified date. The QUICK switch displays only filenames and directory names. The BLOCK switch makes LIST display file sizes in disk blocks used instead of in bytes used. The FILES switch makes LIST show only files, and not directories. The DIRS switch makes LIST show only directories and not files. NOHEAD suppresses the listing of header information.

LFORMAT creates script files. It takes filenames (%S) from the current directory and combines them with a string that can specify and AmigaDOS function. Redirecting the output of a listing that uses LFORMAT creates a script file. LFORMAT automatically issues QUICK and NOHEAD.

LOADWB

(no template or usage)

LOADWB loads the Workbench environment.

LOCK

Drive/A,On/S,Off/S,Passkey:

LOCK sets or unsets write protection for any drive or partition that uses the FastFileSystem. If a four-character PASSKEY is specified when the partition is locked, it must be specified to unlock the partition.

MAKEDIR

/A:

MAKEDIR creates a new directory that uses the name you specify within the current directory.

MOUNT

DEVICE/A,FROM/K:

MOUNT adds a new device to the system. DEVICE is the name of the device, as it is listed in DEVS:mountlist. FROM lets you specify another file that contains the mount specifications.

NEWCLI

Window,From:

NEWCLI starts a CLI process. WINDOW lets you specify the size, position, type, and title for the CLI. FROM specifies the name of a script file that the CLI executes when it is started. If no file is specified, the default S:CLI-Startup is used.

NEWSHELL

Window,From:

NEWSHELL starts a Shell process. Its parameters are the same as NEWCLI. Its default file is S:Shell-Startup

PATH

,,,,,,,,,ADD/S,SHOW/S,RESET/S,QUIET/S:

PATH without parameters shows the search path for the current CLI or Shell. ADD indicates you want to add the listed directory or directories to the search path. SHOW lists the current search path. RESET resets the search path to the default, the current directory, and

C:. QUIET keeps AmigaDOS from requesting that unmounted volumes that are part of the search path be mounted.

PROMPT

PROMPT:

PROMPT changes the CLI or Shell prompt string. The PROMPT parameter is a string. %N prints the current process number in the prompt. %S prints the current directory.

PROTECT

FILE/A,FLAGS,ADD/S,SUB/S:

PROTECT sets or clears file-protection flags. FILE is the file whose protection bits you wish to change. The FLAGS parameter may include any (or all) of the following flags: HSPARWED. H stands for hidden; if set, the file will not appear in DIR or LIST outputs. S indicates the file is a command script and can be executed by entering its name alone on the command line (Shell only). P is the pure flag and is used by programmers to tell AmigaDOS that an executable program is capable of being made resident. A is the archive flag, used by disk-archive programs to indicate whether a file has been backed up. R indicates that a file can be read into memory. W indicates a file can be overwritten. E indicates a file can be edited. D indicates a file can be deleted. ADD and SUB indicate whether you are adding or subtracting the indicated FLAGS. + and – serve the same purpose.

QUIT

RC:

QUIT is used to exit command scripts and to send the specified return value (RC).

RELABEL

DRIVE/A,NAME/A:

RELABEL changes the name of the disk in a specific drive. The DRIVE may be specified by the current name of the disk or by the drive designation (such as df0:). The new NAME is specified without using a colon at the end of the name.

REMRAD

(no template or usage)

REMRAD deactivates and removes the Commodore-Amiga recoverable-RAM disk from the system.

RENAME

FROM/A,TO = AS/A:

RENAME changes the name of a file. The FROM parameter specifies the old name. The TO or AS parameter specifies the new name. RENAME can also be used to move a file from one path location to another, though both the source and destination directory must be on the same device or partition.

RESIDENT

Name,File,Remove/S,Add/S,Replace/S,Pure/S,SYSTEM/S:

RESIDENT loads pure commands into memory and binds them into the system. With-

out parameters, RESIDENT lists the commands that are resident and their usecounts. A command made resident will be found only by the Shell and not by the CLI. NAME specifies the name of the command to be acted upon. REMOVE causes RESIDENT to remove the file from the resident list. ADD causes RESIDENT to try to make the named command resident. The REPLACE switch causes RESIDENT to replace a current copy of this resident command name with a different command that has the same name.

Normally a programmer sets the P (pure) flag in a file's protection bits to indicate that the file may be made resident. To try to make resident a file that does not have the P flag set, use the PURE switch and run the command twice. If the command causes a system crash the second time it is invoked, the command is not pure and cannot be made resident. SYSTEM indicates the command is not available from the Shell.

RUN

(no template or usage)

RUN executes the indicated command as a background task, freeing the current Shell or CLI. If the command requires input or output and does not create its own console or use redirection, it will use the CLI or Shell window.

SEARCH

FROM,SEARCH/A,ALL/S,NONUM/S,QUIET/S,QUICK/S,FILE/S:

SEARCH looks through one or more text files to find a user-specified pattern of characters and reports the line number in each file where the pattern was located. The FROM parameter specifies either a directory or a file. If a directory is specified, SEARCH looks at all files in that directory. When the ALL switch is used, along with a directory name in the FROM parameter, SEARCH looks into all files in the current directory and all files in directories below the current directory. The NONUM switch suppresses the output of the line numbers. The QUIET switch suppresses all output and supplies a return code of 0, if the pattern was found, or 5 if it was not found. If the QUICK switch is specified, a more compact output form is used. The FILE switch forces SEARCH to look only at files within the current directory.

SETCLOCK

Usage: setclock load|save|reset

SETLOCK lets you set (SAVE) the real-time clock in an Amiga 500 or 2000 from the system clock, or LOAD the information from the real-time clock into the system clock. RESET is not defined.

SETDATE

FILE/A,DATE,TIME,:

SETDATE changes the time and date stamp on a file. It prompts the user for a date and time if no parameters are specified. FILE is the name of the file to be stamped.

With no other parameter, the file is stamped with the current time and date. DATE specifies the date stamp, and TIME the time stamp.

SETENV

NAME/A,STRING:

SETENV sets the name of an environment variable. This feature was not implemented when this book went to press.

SKIP

LABEL:

Used in command files, SKIP jumps forward in the script to the indicated label.

SORT

FROM/A,TO/A,COLSTART/K:

SORT orders the lines of a file. The FROM parameter specifies the name of the source file. The TO parameter specifies the name of the sorted file. COLSTART specifies the column number that is the key to the sort. The default is column 1. SORT requires a large stack.

STACK

size:

Without parameters, STACK reports the current size of the stack. When used with a SIZE parameter, it reserves a new stack of the specified size for the current process. The stack is a program's working space. Some programs need more working space than others.

STATUS

PROCESS,FULL/S,TCB/S,CLI = ALL/S,COM = COMMAND/K:

STATUS returns information about AmigaDOS processes. Without parameters, it reports what processes are open and what commands, if any, are currently loaded. PROCESS indicates the process number you want information about. FULL adds stack size, global vector size, and priority to the report. TCB reports on stack size, global vector size, and priority only. CLI and ALL give you the default report. COM and COMMAND let you see if a certain command is being run by a process. They return the number of the process.

TYPE

FROM/A,TO/S,TO,OPT/K,HEX/S,NUMBER/S:

TYPE outputs the contents of the specified file. The FROM parameter specifies the file to be typed. The TO parameter allows the output of the command to be redirected to a file or device. Both OPT H and HEX output the file in hexadecimal as well as ASCII. Both OPT N and NUMBER type the file with line numbers added.

VERSION

Usage: version [<library name>] [version] [revision]

Without parameters, VERSION returns the version numbers of the Kickstart and

Workbench currently in use on the system. When a library or device is specified, VERSION returns the version number of the library or device.

WAIT

,SEC = SECS/S,MIN – MINS/S,UNTIL/K:

WAIT puts the current CLI to sleep for the specified number of seconds (SEC or SECS) or minutes (MIN or MINS), or until (UNTIL) the specified time.

WHICH

FILE/A:

WHICH searches the current AmigaDOS search path for the indicated FILE. It returns the complete path name for the file.

WHY

(no template or usage)

WHY tests the return code for the previous command and, if the code indicates an AmigaDOS error, tries to decipher the error.

B Appendix: AmigaDOS Error Codes

This appendix contains a description of the error messages that AmigaDOS generates and suggests how you may be able to avoid or rectify error conditions. Although AmigaDOS normally reports error messages only, I have included the error numbers also.

103—Insufficient free store

AmigaDOS did not have enough memory to run the most recent command you gave it. Close one or more windows, or stop one or more applications, thus freeing up some memory, and try it again. This error will also occur if the command requires a large chunk of contiguous memory, and the memory available in your system is in small chunks.

104—Task table full

AmigaDOS has a fixed limit of 20 CLI and Shell processes. If you try to exceed the limit, you get this error.

120—Argument line invalid or too long

This error indicates you have probably entered the command or its parameters incorrectly. Also, if the command line is over 255 characters, any characters beyond that point will have been ignored and may have resulted in an incorrect command being issued to AmigaDOS.

121—File is not an object module

You have tried to run a file that is not an Amiga program. You may have typed the name of another file by accident. This error was common in the early days of Amiga telecommunications when XMODEM padding changed the length of executable files and rendered them unusable. Amiga telecommunications packages and archive files have eliminated this problem.

122—Invalid resident library during load

This message indicates either a problem with one of the files in your LIBS: directory or that some program has managed to corrupt a resident library. This is a rare error, and you will probably have to reboot to fix it. If this does not work, recopy the LIBS: directory from your original Workbench disk to your working copy of Workbench.

202—Object in use

You usually see this error message when you try to delete or write to a file that is already in use by another program. This message also occurs when you try to delete an assigned file or directory.

203—Object already exists

If you have a file or directory with the same name as the one you have given to MAKEDIR, this error is reported. The fix is to give your new directory a different name.

204—Directory not found

If you specify the name of a directory as the destination for a command, and that directory does not exist, you will get either this message or simply a "not found" message.

205—Object not found

You get this error if there is no file or directory that has the name that you specified as the source or destination of a command. Perhaps you have not spelled the name correctly. This error message is printed also if you try to create a file in a directory that does not exist, or on a device that is not mounted.

206—Invalid window description

You have issued a command that uses an incorrect or incomplete CON: specification. Make sure the window you want to create is within legal limits.

207—Invalid stream component name

The file name you have tried to use is either too long or contains a control character. Watch your typing!

212—Object not of the required type

This error occurs most often if you try to run a file that is not executable. For example, a directory is not a program and cannot be run.

213—Disk not validated

This error can occur because you have turned your machine off while a file was open or you have popped a disk out of its drive while the disk-activity light was still on. The error is often detected during startup, and will often be fixed after about a minute of disk activity. If the error persists, you may have to run DISKDOCTOR on the disk to recover whatever files are still intact.

214—Disk is write-protected

You have tried to write to a disk whose write-protect notch has been set to the protected mode. A disk is protected if you can see through the hole in the corner of the disk.

215—Rename across devices attempted

You can use the RENAME command to either change the name of a file within a directory, or to move the file on the same device from one directory to another directory. You cannot, however, use RENAME in place of the COPY command to move a file from one device to another, or between two partitions of the same device.

218—Device not mounted

You will see this error if you have tried to access a file on a disk that has not been inserted into the drive. Put the proper disk in the drive and try again.

220—Comment too big

This error indicates that you have tried to set a filenote that exceeds the maximum allowable size of 80 characters. The solution, of course, is to limit your filenotes to 80 characters or less.

221—Disk full

There is not enough room on the disk you are using to perform the operation you have requested. Use another disk, or delete some files or directories that you no longer need from this disk and try again.

222—File is protected from deletion

AmigaDOS will not allow you to write over a file or to delete a file that has been protected from deletion. You protect a file either by using the Workbench INFO Menu item to make sure the DELETABLE gadget is turned off, or by using the PROTECT command from the CLI. You unprotect a file the same way.

223—File is protected from writing

This error is like error 222, but refers to the write flag (W) instead of the delete flag (D). In versions of AmigaDOS up to and including 1.3 (Gamma 7 preliminary version), this error cannot occur because AmigaDOS does not pay attention to the W flag. In later versions of AmigaDOS, the flag may be utilized.

224—File is protected from reading

This error is similar to error 223, but refers to the read flag (R) instead of the write flag (W).

225—Not a DOS disk

You have inserted a disk into a drive that has not been formatted by AmigaDOS. This error can also occur when a disk has been damaged and requires DISKDOCTOR, or when the disk is copy-protected.

C Appendix: Amiga Users' Groups

This list, supplied through the courtesy of Jim Meyers, contains the names and addresses of many Amiga-specific users' groups. The list is sorted by state and includes, where the information is available, the name of a contact person. Some entries also contain a telephone number and/or an electronic-mail address.

To add your group to the list of Amiga groups, contact Jim Meyer at AMICUS/Hudson Valley.

Amiga Users Group ARK
Howard Couch
512 Stonewall Drive Suite 120E
Jacksonville, AR 72076

Amare Amiga
211 W. Roger #29
Tucson, AZ 85705

ABACUS
Vitas Safronicikas
5001 Hunter Ave. #8
Bakersfield, CA 93309

AmiNews Users' Group
5601 Kitty Hawk Lane
Redding, CA 96003

AUGment
Jim Thomas
PO Box 1863
Fremont, CA 94538-0186

Lockhead Amiga Users' Group
PO Box 201
Los Altos, CA 94023-0201

Los Angeles Amiga Users' Group—LAAUG
William Simpson
1711 Altivo Way
Los Angeles, CA 90026
213-661-7959

Sacramento Amiga Computer Club
George Leone
PO Box 19784
Sacramento, CA 95819-0784
916-944-7400

San Diego Amiga Users Group—SDAUG
Bill Elder
PO Box 80186
San Diego, CA 92138-0186
481-7967

San Fernando Valley AUG
Albert Di Paolo
PO Box 8183
Van Nuys, CA 91409
818-786-8624

San Gabriel Valley & Pomona Amiga Tech Users Group—ATUG
Mark Randell
2227 Canyon Rd.
Arcadia, CA 91006

SFCUG
278 27th Ave., Suite 103
San Francisco, CA 94121

South County Amiga Users' Group
Roy Wagner
13712 Clarmont St.
Westminister, CA 92683

Southern California Amiga Club
PO Box 727
Walnut, CA 91789

Stockton-Modesto Amiga Users Group
Rainer Kummerle
3581 Benjamin Holt Drive #168
Stockton, CA 95209

Valley Computer Club
Amberse M. Banks
505 Ryan Avenue
Modesto, CA 95350

Mile High Amiga Users' Group
1236 Leyden
Denver, CO 80220

FCAUG—Fairfield County Amiga User's Group
Paul Gerhardt
20 Moody Lane
Danbury, CT 068111
914-855-5966
Barry Newman CIS 72426,1566

National Capital Area Users' Group
PO Box 18088
Washington, DC 20036-8088

Main Line Users Group Amiga SIG
Robert E. Taylor
210 Durso Dr.
Newark, DE 19711

Amiga Info Xchange
Charles R. Goodman
225 S. Clair Dr.
Panama City, FL 32401

Central Florida Amiga Club
T. Lee Kidwell
1056 Neely St.
Oviedo, FL 32765

Space Coast Amiga Users' Group
Michael D. Dalton
PO Box 2098
Merritt Island, FL 32952

Tampa Bay Amiga Group
Billy W. Combs
920 S. Rome Ave.
Tampa, FL 33605

The Suncoast Amiga Group (TSUNAMI)
2391 South Pines Dr. D36
Largo, FL 33541

Amiga Atlanta#
Andre French
Box 7724
Atlanta, GA 30357

Amiga User Group Of Statesboro
Dr. Russell A. Dewey
223 North Edgewood Dr.
Statesboro, GA 30906

Eagle Rock Commodore Computer Club
James R. White
2100 Belmont Ave.
Idaho Falls, ID

AAUG—Algonquin Amiga Users
Tim Atwood
802 W. Surrey Lane
Algonquin, IL 60102

Champaign County Amiga Users' Group
Wayne Hamilton
907 W. Nevada
Urbana, IL 61801

Commodore Hardware Users' Group
Greg Chaney
1322 Fairview Dr.
Greenfield, IN 46140

Fort Wayne Amiga Users' Group
Tony Vassiliadis
183 Oak Park Dr.
Roanoke, IN 46783

Indiana Amiga Advisors
Dennis Graham
912 S. Brown Ave.
Terre Haute, IN 47803

Greater Lafayette Amiga User's Group—GLAUG
Gary Rante
PO Box 246
Lafayette, IN 47902
Gary Rante 317-474-1332

Wichita Amiga User Group
Doug Hammond
1001 Capri St.
Wichita, KS 67207

Shreveport Amiga Users Group—SAUG
3311 Colquitt Rd.
Shreveport, LA 71118
Les Johnson PLINK LES*J

New Orleans Commodore Klub—NOCK
Robert H. Fergeson
6370 & ½ Catina St.
New Orleans, LA 70124
504-482-8551

M.I.T. AUG
Jim Haleblian—Liason
58 Manchester Rd.Brookline, MA 01246
617-734-0625
Note—this is an official MIT student organization; the above information is the liason's address and phone number.)

Amiga Group of The Boston Computer Society
Bill Walde
PO Box 839
Melrose, MA 02176
617-263-8070

Lowell Massachusetts Computer Club
Phillip E. Sweet
Lowell, MA
617-250-1904

Baltimore Amiga Users/Developers
Ed Hopper
243 W. 31st St.
Baltimore, MD 21211

Baltimore Area Computer Club—CUMBACC
William J. Kolodner
PO Box 479
Reisterstown, MD 21136-9998

. . . AAmiga
Eoin Cain
PO Box 4272
Ann Arbor, MI 48106

Amiga User's Group Of Pontiac
2548 Orchard Lake Rd.
Pontiac, MI 48053-2435

West Michigan Amiga Users Association
BBS—616-459-7261

Annie MUG
1500 E. Medical Center Dr.
Ann Arbor, MI 48109

Amiga MN Interest Group Alliance
PO Box 32374
Fridley, MN 55432

Gateway Amiga Club
14850 Phelps
Bridgeton, MO 63044

Kansas City Amiga User Group
Joe Simunac
22 W. 59th St.
Kansas City, MO 64113

Amiga Users Of The Raleigh Area—AURA
Ray Cook
1114 Wyldewood Rd.
Durham, NC 27704

Greater Fayetteville Amiga Users' Group
PO Box 544
Spring Lake, NC 28390

The Amigans
Dick Barnes
PO Box 411
Hatteras, NC 27943

Triad Commodore Users' Group—TCUG
Gil Hutcheson
PO Box 10833
Greensboro, NC 27404

Grand Forks Amiga Users Group
407 Demers Ave.
Grand Forks, ND 58201

Greater Omaha Commodore Users' Group
Fred Layberger
PO Box 241155
Omaha, NE 68124

Heartland Amiga Users
Fred Layberger
14524 N. St.
Omaha, NE 68137

Nebraska Amiga Users' Group
PO Box 2414
Lincoln, NE 68502

Amiga Users Group—Computer Sciences Corporation
Tony Preston
304 W. 38th St.
Moorestown, NJ 08057

Amiga Users' Group South Jersey
Jay Forman
PO Box 3761
Cherry Hill, NJ 08034

JAUG—Jersey Amiga User's Group
Eric Lavitsky
PO Box 1986
New Brunswick, NJ 08901
201-745-2839
E. Lavitsky CIS 70556,1642

New Mexico Commodore Users' Group
Walter Stanley
PO Box 37127
Albuquerque, NM 87176

AMICUS/Hudson Valley
Jim Meyer
PO Box 1168
Wappingers Falls, NY 12590
914-297-8759
Jim Meyer CIS 75475,456, PLINK NY*JIM, BIX JMEYER, Delphi JMEYER

Amiga Mouse Users' Group—AMUG
Joseph Rothman
PO Box 148
Central Islip, NY 11722

AMuse—New York Amiga Users
Joe Lowery
151 1st Ave., Suite 182
New York NY 10003
212-460-8067
Joe Lowery CIS 72437,2354

AMuse Westchester
Rick Finn
189 Pinewood Rd.
Hartsdale, NY 10530

Central NY Amiga Aggregate
8018 Evesborough Dr.
Clay, NY 13041

Computer Users' Group Of Rochester
Steve Collins
PO Box 23463
Rochester, NY 14692

Lica-Amiga
Aaron Schildkraut
PO Box 158
Mill Neck, NY 11765
516-676-8956
Rocco Passarette CIS 70435,417

National Amiga Users' Group
PO Box 151
Oakland Gardens, NY 11364

SAUG—Sperry Amiga Users Group
Michael Colucci
Sperry Corp. Lakeville Rd.
Great Neck, NY
516-574-1988

Buffalo Amiga Users and Developers—BAUD
84 Roosevelt Ave.
West Seneca, NY 14224-3158

Cleveland Area Amiga Users' Group—CA-AUG
Bill Hogsett
3715 Townly Road
Shaker Heights, OH 44122
BBS—216-341-4452

Ohio Valley Amiga Users' Group
Eric Hanson
3115 Bellewood Ave.
Cincinnati, OH 45213-1603

Amiga Computer Enthusiasts
Mark Fulton
1111 N. St. Charles #16
Oklahoma City, OK 73127

Northwest Amiga Users Group
PO Box 1140
Oregon City, OR 97045

East Hills Amiga Group
928 Presque Isle Dr.
Pittsburgh, PA

North American Amiga Users Group
Richard Shoemaker
Box 376
Lemont, PA 16851
814-237-5511
BBS—814-339-6042

The Amiga Group
719 E. Lancaster Ave.
Downingtown, PA 19335

Charleston Amiga Users Group
1030 Ft. Sumter Dr.
Charleston, SC 29412
BBS—803-571-6030

Carolina Amiga Users—CAUSERS
Ray Marsh
2224 Airport Rd.
West Columbia, SC 29169

Amiga Users Group Of Rapid City
Roxann Pappas
PO Box 1377
Rapid City, SD 57709

Memphis Amiga Group
PO Box 381462
Germantown, TN 38138-1462

Permian Basin Amiga Users Group
PO Box 12272
Odessa, TX 79768

Amiga Users' Club Of Houston
Joseph Mamby
6525 S. Gessneer #3100
Houston, TX 77036

Dallas MIDI Users Group—DMUG
Ray Reach
4306 Pineridge Dr.
Garland, TX 75042

El Paso Amiga User Group—EPAUG
Michael Cox
637 Bristol Dr.
El Paso, TX 79912

Amiga User's Society Members—AUSM
Blaine Gardner (Secretary)
160 S. 800 East
Bountiful, UT 84010
801-295-3677
BBS—801-250-7438

Richmond Amiga Group
2815 New Kent Ave.
Richmond, VA 23225

Amiga User Group
Jim Naiden
17050 2nd Ave. NW
Seattle, WA 98177

Olympia Amiga Users Group
Noah Sherman
4104 Amber Ct. SE
Olympia, WA 98501

Washington Amiga User Group—WAG
William Upton
5302 93rd Place NE
Marysville, WA 98270

The Amiga BitMappers
Clinton Kurek
PO Box 1641
Milwaukee, WI 53201-1641

N.E. Wisconsin Amiga Users Group—NEW AMUG
W.J. Raynor
3615 Sunnyview Rd.
Appleton, WI 54914

Pacific Northwest Amiga Association
Dave Allen
10851 Shellbridge Way
Richmond, BC V6X 2W8
Canada

Victoria Amiga User Group—VAUG
Ewan Edwards
2101 Government St.
Victoria, BC V8T 4P2
Canada

Amiga SIG London Amiga Club
Bob Rutter
Apt. 10006 767 Second St.
London, ON
Canada
451-0228

North-Bay Amiga Users' Group—NorAUG
Ken Strange
970 Copeland St.
North Bay, ON P1B 3E4
Canada

TPUG Magazine
Tim Grantham
5300 Yonge St.
Willowdale, ON M2N 5R2
Canada

Montral Amiga Group—MAG
Jean Papin
1160 St. Mathieu Penthouse 3
Montreal, QU H3H 2P4
Canada

Amiga Users Group (U.K.)
Tony Lacy
66 London Rd.
Leicester, LE2 0QD
England
0533-550993

The Amiga User Group
14 Parkstone Ave.
Horfield, Bristol, Avon
England

D Appendix: Using ED

Although not the most powerful editor available for the Amiga, ED is useful for creating and editing script files, especially startup-sequence files. It is also free. The following is a description of the basic ED commands, followed by a listing of all the ED commands.

When you invoke ED from the command line, you must include the name of a text file—ED can edit text files only. ED will load the file you named if the file exists, or create a file by that name if it does not.

ED is a full-screen editor: you can use the arrow keys to move to any point in the file. To enter text at any point, simply start typing—ED inserts what you type at the current cursor location. To delete text, use Delete or the backspace key.

Once you have created or modified a file, you must save it. Pressing Escape X saves the file and exits ED. Escape SA saves the file and returns to ED. Escape Q quits without saving the file.

This is all you have to know to create and edit text files with ED. All the other commands simply make the above tasks easier. ED has two types of commands: immediate and extended. You access the immediate commands by pressing the Ctrl key at the same time you press a command character. You access the extended commands by first pressing Escape, and then pressing a command character or character combination. The immediate and extended ED commands are listed below.

Immediate Command	Function
Ctrl-A	Insert a line
Ctrl-B	Delete a line
Ctrl-D	Move cursor up
Ctrl-E	Move cursor to top/bottom of display
Ctrl-F	Change case of character
Ctrl-G	Repeat last extended command
Ctrl-H	Destructive backspace
Ctrl-I	Move cursor to next tab position
Ctrl-M	Carriage return
Ctrl-O	Delete word or spaces
Ctrl-R	Move cursor to end of previous word

Ctrl-T	Move cursor to start of next word
Ctrl-U	Move cursor down
Ctrl-V	Refresh display
Ctrl-Y	Delete from cursor to end of line

Extended Command	**Function**
Escape A	Insert line
Escape B	Move to bottom of file
Escape BE	Mark end of block
Escape BF/s/	Search backwards in file for s
Escape BS	Mark start of block
Escape CE	Move cursor to end of line
Escape CL	Move cursor one position to the left
Escape CR	Move cursor one position to the right
Escape CS	Move cursor to start of line
Escape D	Delete current line
Escape DB	Delete marked block
Escape DC	Delete character
Escape E/s/t/	Change s to t
Escape EQ/s/t/	Change s to t with confirmation
Escape EX	Extend the right margin
Escape F/s/	Find s
Escape I/s/	Insert s as line before current line
Escape IB	Insert marked block
Escape IF/s/	Insert file named s
Escape J	Join current line with the following one
Escape LC	Make searches case-sensitive
Escape M n	Move cursor to line n
Escape N	Move to beginning of next line
Escape P	Move to beginning of previous line
Escape Q	Quit without saving the current file
Escape RP	Repeat following command until error
Escape S	Split line
Escape SA	Save text to file
Escape SB	Show marked block
Escape SH	Show file information
Escape SL n	Set left margin at n
Escape SR n	Set right margin at n
Escape ST n	Set distance for tab
Escape T	Move cursor to top of file
Escape U	Undo changes on current line
Escape UC	Make searches non case-sensitive
Escape WB/s/	Write block to file s
Escape X	Save file and exit

Index

Notes

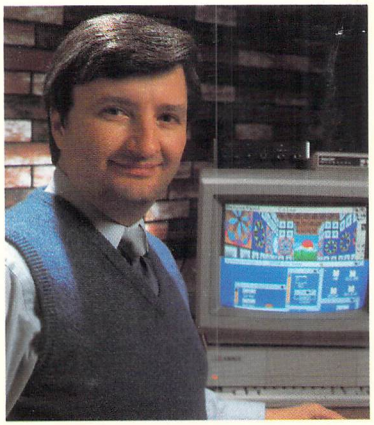
Notes

Notes

The Amiga™ Companion

A complete, up-to-date guide to using AmigaDOS and the CLI.

With its fantastic graphics and sophisticated multitasking operating system, the Amiga from Commodore is one of the most powerful and versatile personal computers ever made. Harnessing this power can be difficult, however, because much of it can only be accessed through AmigaDOS and its Command Line Interface. Now AmigaWorld magazine presents the Amiga Companion, a book containing all the information you need to master AmigaDOS and become an Amiga power user.



Written by Rob Peck, formerly of Commodore-Amiga and author of the Amiga ROM Kernel Manual and the Programmers Guide to the Amiga, the Amiga Companion is more than a reference guide— it is a set of solutions designed to let you get the most out of your Amiga.

Want to know how to use AmigaDOS to get your Amiga to run faster, smoother and smarter? The Amiga Companion has the answer. Want to know how to best utilize a RAM disk or get the most out of a one-drive system? The Amiga Companion shows you, in detail. Not satisfied with reproducing the list of AmigaDOS commands, the Amiga Companion shows you how to use AmigaDOS to improve your productivity and the productivity of your Amiga. Its no-nonsense "cookbook" approach makes AmigaDOS useful and understandable to all Amiga owners. And, unlike standard reference guides, the Amiga Companion is written to be understood by novice users as well as long-time Amiga owners.

Here are some of the informative chapters you'll find in The Amiga Companion:

- Introduction to the CLI
- AmigaDOS Shortcuts
- Software in the Public Domain
- Tools on the Workbench
- Information Commands
- The Heart of the CLI
- AmigaDOS Scripts
- Console Control Characters
- Configuring AmigaDOS

The Amiga Companion is a clear, concise and complete (it covers AmigaDOS through version 1.3) guide to AmigaDOS. Combining the talents and resources of AmigaWorld, the number one magazine for Amiga users, and Rob Peck, one of the best known "Amiga Gurus," the Amiga Companion is an essential guide to AmigaDOS and the CLI. It's the last word on using AmigaDOS.

$19.95

ISBN 0-928579-00-X