CMU-CS-85-151

The Design and Evaluation of On-line Help Systems

Nathaniel S. Borenstein

April 27, 1985

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at Carnegie-Mellon University

This research was sponsored by a Graduate Fellowship from the National Science Foundation, and by a Forgivable Loan from the General Electric Foundation. The views and conclusions contained in this document are those of the author and should not be interpreted as being the views and conclusions of the National Science Foundation or of General Electric.

To Debbie and Stan Borenstein

I would repay the bounty they have given me,
but it is as the sky: it can never be approached.

-- Huston Smith [110]

List of Tables

Abstract

On-line help is a vital part of nearly every computer system of any significant size, yet it is poorly understood and generally poorly implemented. The primary goal of this research was to discover as much as possible about how on-line help systems should be built. Several results are presented in the thesis.

First, the feasibility of *easily* building more general and powerful help systems than are commonly available is demonstrated. A prototype system, including nearly all the features included in any real-world help system, and integrating them in a manner not found in any such system, was built in just a few months without any spectacular tricks of implementation. The prototype system runs well and quickly, even with a very large database of help information, so that there is simply no reason not to expect practical help systems to live up to its standards.

Second, controlled experiments comparing various alternative help systems were conducted. These experiments suggest that the quality of help texts is far more important than the mechanisms by which those texts are accessed. The experiments also suggest that, despite the well-documented fact that people read and comprehend better from printed texts than from computer screens, it is at least possible to compensate for this fact through sophisticated help access mechanisms. Thus, on-line help with *no* printed manual may be at least as useful as a more traditional manual-based system, and there are reasons to suspect that it can be even better. Additional experiments, using a simulated natural language interface, cast doubt on the usefulness of natural language in a help system.

In addition to the concrete results summarized above, the thesis makes several other contributions. A general taxonomy of on-line help systems is developed, and a survey of the literature on help systems relates existing help systems to that taxonomy. Further, the design and evaluation of help systems is considered as an example of the more general problem of designing and evaluating user interfaces. The methodology facilitates the evolution of such interfaces with a minimum of attention to the details of implementation and experimentation. Finally, a potpourri of practical, sometimes anecdotal information likely to be of interest to future help system designers is collected in a practitioner's summary, and related topics ripe for further research are described in a researcher's agenda.

Acknowledgements

I owe so many people a debt of gratitude for their help in my studies that I had begun to fear that the acknowledgements would be the longest section of my thesis. This fear notwithstanding, I would like to thank the following people:

My advisor, Jim Morris, has been everything one could want in an advisor; without his encouragement, I would be in a gutter drinking cheap wine today.

The other members of my committee, Dick Hayes, Phil Hayes, and Frank Wimberly, have been invaluable to me. Their sharp insights and criticisms have been balanced with well-timed encouragement, and I am forever in their debt as well.

Kamila Robertson has also contributed more to me than I can repay. It was with her help and encouragement that I took the fateful step away from the orthodoxy of Computer Science and into the uncharted world of Human Factors.

I owe a special debt of gratitude to my friends in the Statistics Department, Yves Thibaudeau and Rob Kass, who saved me when I got over my head in regression analysis and other esoterica of their noble science.

I would like to give special thanks to Mark Sobell and his publisher for permission to use large portions of their excellent introductory text on UNIX[1]. Being able to use these texts saved me months, if not years, of work, and helped guarantee the objectivity of my experiments. Those experiments provided confirmation of one thing I had believed from the start, namely that Sobell's text is truly a top-quality introduction to UNIX. I recommend it highly.

Additionally, I would like to thank the many people who helped me in small ways and large during the course of this research: Mike Accetta, David Axler, Diana Bajzek, Rene Banares, Keith Barrett, Michael Bergman, Bob Bramwell, Benjamin Britt, Peter Brown, Jean Brule, Jaime Carbonell, Stuart Card, Tim Carlin, Jack Carroll, Davida Charney, Margaret Christensen, Don Cohen, Tim Curry, John Daleske, Mark Day, Walt Doherty, Marc Donner, Tom Duffy, Ivor Durham, Rex Dwyer, Jennifer Dykeman, Carl Ebeling, Jeff Eppinger, David Evans, Craig Everhart, Gary Feldman, Kathy Ferraro, Tim Finin, Alan Fisher, Ed Frank, Bob Frederking, Lionel Galway, David Garlan, Dick Glenn, James Gosling, John Gould, Walt Haas, Leonard Hamey, Fred Hansen, Peter Hibbard, Steve Hill, Susan Horner, Adele Howe, Guy Jacobson, Ron Jarrell, Robin Jeffries, Mike Kazar, Jeff Koechling, Larry Kraines, John Kunze, Diane Langston, Jill Larkin, Robert Elton Maas,

[1] *A Practical Guide to the UNIX System* by Mark G. Sobell. Copyright (c) 1984 by Mark G. Sobell, published by The Benjamin/Cummings Publishing Company. UNIX is a trademark of AT&T Bell Laboratories.

Jon McCombie, Bud Mishra, Allen Newell, Eric Osborne, Tom Peters, Frank Pfenning, Jon Rosenberg, Mike Rychener, Rex Sanders, Mahadev Satyanarayan, John Schlag, Ben Shneiderman, Jeff Shrager, Mel Siegel, Roy Taylor, Aaron Temin, Ed Thompson, Mabry Tyson, Jeff Tytus, Hank Walker, Janet Walker, Jon Webb, and Bob Whiteside. Additionally, I would especially like to thank all the people that should have been listed here, but weren't due to the frailty of my memory.

I also want to thank the National Science Foundation for a Graduate Fellowship which supported me during my first three years as a graduate student, and the General Electric Foundation for a Forgivable Loan which supported me during my last three years. Their generous help was invaluable.

My colleagues at Advanced Programming Resources of Columbus, Ohio, several of whom are already listed in the acknowledgements above, have provided me with a wonderful opportunity to use some of the knowledge I've acquired from this thesis before my thesis was even completed. I want to thank Barry Heagren, Ken Sherman, and everyone else at APR for their support and encouragement in the continuing development of ACRONYM's successors.

Finally, with their rank in these acknowledgement reflecting not their importance but rather the priority with which they all too frequently have been given my time, I want to thank my family and friends for their love and support. My wife, Trina, my children, Shana and Rachel, and most of my relatives and friends have continued to believe in me even when any rational being could tell that I would never in a million years complete my doctorate. If it weren't for them, they could never have been right.

Part One

The Problem

Chapter 1 Introduction

This thesis investigates the problems inherent in the design and evaluation of on-line help systems. The general problem of making computers easier to use is a vast enterprise of obvious importantce. One common technique used toward this end is *on-line help*. In this thesis, I will address the questions of how useful on-line help really is, what design alternatives may affect its relative usefulness, and how such systems can be evaluated.

By "on-line help systems" or "interactive help systems" I refer simply to any computer software that has as its primary function the task of providing the user with information that will assist him in the use of some other software system. This includes help systems that stand alone, as independent utility programs in a a larger operating system, and subsystems embedded within larger systems, generally help procedures within specialized utilities.

This thesis focuses entirely on the user interface to interactive help systems. Related topics that will not be treated in detail include the implementation of help systems, the design of the task domain for which help is being provided, and the importance of on-line help relative to other factors affecting the ease with which a system is used. However, each of these areas is closely related to the subject of this thesis, and some relevant material will of necessity be presented.

In the remainder of this chapter, I will describe in more detail the problems this thesis is concerned with, and will outline the way the remainder of the thesis describes the solution.

1.1. How Should We Design User Interfaces?

Much has been written about the design and testing of user interfaces.[2] Several authors have proposed design methodologies in an attempt to facilitate and standardize the process by which good

[2]For example, [1, 4, 6, 14, 29, 32, 53, 54, 60, 61, 62, 68, 75, 78, 88, 92, 98, 102, 107, 121], to name a few.

interfaces come into the world. Unfortunately, these are rarely used in the real world for a very good reason: they typically mandate a prohibitive amount of time spent in iterative design and formal testing.

Although this thesis is concerned primarily with the specific question of designing and evaluating on-line help systems, it also serves as a case study in the use of such methodologies. The methodology used here is original, but derived in some part from those that have gone before. In outline, it consists of eight steps:

1. State the problem. In order to better understand the problem, closely observe real world users of the kind of system being studied. For this thesis, the problem can be stated as "What would a good on-line help system look like?" This question is expanded upon in Section 1.2.

2. Describe what already exists. Try to develop a framework that describes all the known existing systems by their differences along some small number of simple dimensions. This is done here in Chapters 2 and 3.

3. Search the literature for previous experimental results contrasting competing system designs. This is done in Chapter 4.

4. Construct hypotheses regarding the most important design decisions: What are the most important decisions? What are the likely answers? These are made explicit in Section 1.3.

5. Implement a prototype system that supports each of the major design alternatives. Use all available tools to build the system quickly, sacrificing portability, efficiency, and modularity where necessary to get the job done fast. The implementation used in this thesis is described in Chapter 5.

6. Conduct controlled experiments to test the hypotheses about the correct design decisions. Carefully observe the details of the users' interaction with the system. The experiments used here are described in Chapter 6, with the results reported in Chapter 7.

7. Where the experiments do not bear out the hypotheses, consider reiterating from step 4.

8. Build the real system, doing right all of the things that were done wrong in step 5. This was not done as part of this thesis, but the conclusions point out a clear path in this direction. These conclusions are summarized in Chapter 8.

As stated above, this methodology is not terribly new. However, it is somewhat unusual in its use of jury-rigged prototype systems, its emphasis on thorough initial searching of the literature, and the use of protocols from previous systems. All of these features are geared to an often-mentioned aspect of software development in recent years, the fact that one of the largest costs is programmers' time. This methodology is designed to delay the costly coding efforts until the last possible moment, with the hope that this will reduce the later costs associated with software maintenance and redesign.

Methodologies such as this one are often proposed, and their evaluation is difficult and subjective. In Section 8.2, I will discuss the results of using the methodology in the domain of on-line help, as well as its applicability to other investigations of user interface design.

1.2. How Should We Build Interactive Help Systems?

Help systems have traditionally been one of the most neglected aspects of interactive systems. Many otherwise excellent pieces of software include no help system at all, or help systems so ill-conceived as to be nearly useless. Although some better examples are available [84, 100], the help systems in general use are so uniformly unappealing that designers who do make an effort to construct worthwhile help systems tend to assume that they are starting with a clean slate, working on a problem that has never been seriously confronted before. Indeed, such was my assumption when I began the work on help systems reported in this thesis. However, I have discovered that quite a few interesting examples of useful help systems do exist, from which much can be learned.

First, however, it helps to have a clear idea of the underlying questions to be answered. I begin with the assumption that the single most important criterion by which a help system should be judged is the degree to which it facilitates the accomplishment of a particular task by a user who did not previously know how to accomplish this task.

This is a very practical criterion, but not the only one possible. For example, it is not necessarily the case that the help system which is most *effective* in the sense described above will also be the one which the users most enjoy using. Additionally it is probably not the case that the help system which is most effective will also be the one which is the greatest aid to long-term learning. Indeed, precisely the opposite may be true: it may be that the harder users have to work to learn to accomplish a particular task, the more likely they are to remember the solution. Even if that were true, however, it would not be reasonable to conclude that help systems should therefore be as unhelpful as possible, in order to promote long-term learning. Therefore in the absence of an objective and reasonable alternative, the effectiveness of a help system in facilitating learning has been selected as the primary criterion by which help systems will be evaluated. In addition, subjective user preferences and retention will also be considered as secondary measures of interest.

The design of on-line help systems is hampered somewhat by a constraint that is universal in interactive systems, but uniquely important in help systems. This is the need for *simplicity* -- for an interface that requires minimal knowledge and effort for its use. Certainly simplicity is a virtue in the user interface of any program, but it is vital for a help system. Typically, a help system is never used for its own sake but only as a tool to aid in the use of another system. It is therefore used by people in a hurry, often total novices, often already very frustrated, who have absolutely no desire to learn any more than the bare minimum about the help system itself. In this sense, many fancy features may be self-defeating in a help system, if they slow the user's progress toward the explanation he seeks by making him first learn the complexities of the help system.

What, then, should a help system actually look like? Indeed, is it even possible for a help system to be of more assistance than a paper manual, which certainly presents the simplest mechanism imaginable for most users? The remainder of this thesis provides the beginning of an answer to these questions.

1.3. The Experimental Hypotheses

In this section, I will briefly describe the fundamental hypotheses that motivated the design of the ACRONYM help system (Chapter 5) and the evaluative experiments (Chapter 6). These hypotheses were arrived at after the literature surveys, user surveys, and user protocols that are summarized in Chapters 3 and 4. The most striking fact about these hypotheses is that, after all that surveying, so many of them were wrong. They are nonetheless presented here in their original form, to allow the reader to make his own guesses before the results are detailed.

Hypothesis I: While quality of help texts is crucial to good on-line help, the methods by which those texts are accessed are of equal or greater importance in their effect on learning time.

Hypothesis II: New users of a system will learn more quickly using menu and tutorial help systems than with key word or user-initiated context-dependent help systems.

Hypothesis III: Experienced users will learn new tasks more quickly with key word and user-initiated context-dependent help systems than with menu and tutorial help.

Hypothesis IV: Both new and experienced users will fare better when both context-dependent and non-context-dependent help are available than when only one or the other is available.

Hypothesis V: Even a very sophisticated help system will not be nearly as helpful as a human tutor.

Hypothesis VI: Allowing users to type help requests in English will not significantly improve their rate of learning.

The results of the experiments tend to indicate that one of these hypotheses[3] was correct, three[4] were simply wrong, and two[5] were concerned with differences too small to be detected by the experiments. Nonetheless, these basic hypotheses (most of which were specified explicitly when the experiments were designed) underly the design of the help system and experiments described in the Chapters 5 and 6.

1.4. The Structure of this Thesis

This thesis is divided into five major parts. Part One presents the problem and its background. After this introductory chapter, it consists of a general taxonomy of help systems (Chapter 2), a survey of existing help systems in the context of that taxonomy (Chapter 3), and a survey of prior experimental results relevant to help systems (Chapter 4).

Part Two describes the method used to attack the problem. Chapter 5 describes the design of a prototype help system, known as ACRONYM. This system offers all of those features isolated as most important by the taxonomy and survey, using an integrated database. The factors leading to ACRONYM's design and its limitations are discussed, along with the actual workings of the system. Chapter 6 describes the experimental method used to evaluate ACRONYM and the hypotheses associated with its design. This chapter also explains why the experiments were designed as they were, and what other help systems were evaluated.

Part Three analyzes the results of the experiments. Chapter 7 presents in detail the results comparing the help systems, as well as issues of user expertise, task and subject variation, subjective evaluations, and user retention. Chapter 8 summarizes the results and contributions of the thesis, both in regard to help systems and in the more general area of user interface design. This last chapter includes a Practitioner's Summary, which could be subtitled "Advice to Builders of Future Help Systems", and a Researcher's Agenda, summarizing the vast body of relevant and interesting questions *not* answered by this research.

---

[3]Hypothesis VI

[4]Hypotheses I, IV, and V

[5]Hypotheses II and III

1.5. How to Read this Thesis

There are several different approaches to reading this thesis which might be useful, depending on what you want to get out of it.

If you want to build a real help system, and are looking for practical advice, you should probably read all of the background material (Chapters 1-4). You should then read the description of ACRONYM in Chapter 5, and the conclusions in Chapter 8. You will certainly want to look at Appendix B, which demonstrates the system in use, and may want to peruse Appendix C as well.

If you're more interested in this thesis for its experimental methodology, especially if you're planning similar experiments in the future, you may want to skip Part One altogether; in fact, you can probably begin with Chapter 6 and read the rest of the thesis from there. You may wish to look over Appendix A to see how the experiments looked to the subjects. If you're not comfortable with regression analysis, you might want to read Appendix E as well.

If you have a special interest in issues of text readability and technical writing, you should test your skills on the examples in Appendix D. This appendix presents samples of the two versions of the UNIX manual studied in the experiment, one of which was dramatically more useful to the subjects than the other. The details and implications of this particular results are presented in Section 7.3.

Chapter 2
A General Taxonomy of Help Systems

People have been building interactive help systems almost as long as they have been building software systems of any kind. In general, however, the help system has been an afterthought, quickly and hastily constructed and only marginally integrated into the larger system. Programmers like to program; They are not so fond of documenting their code, and even less fond of writing documentation for the final users of their systems. On-line help, it would appear, is yet still lower in the hierarchy of the programmer's favorite activities.

Nonetheless, over the years quite a few interesting approaches have been tried, generally in isolated settings and in an *ad hoc* manner. A few attempts have been made to survey the field [55, 113, 108], but these have been less than successful. Houghton [55], for example, purports to survey the area of on-line help, but includes in this category such diverse topics as error messages and prompting. As far as the central topic of this thesis, the mechanisms of on-line help, is concerned, he does describe many of the types of help system to be discussed here, but he makes no attempt to fit them into any framework or taxonomy to describe such systems.

Sondeheimer and Relles [113] actually do construct a simple taxonomy of on-line help systems. They classify help systems according to four dimensions. The following descriptions are from the article cited:

1. *access method -- the way users can construct or enter requests for assistance.*
2. *data structure -- the manner in which different portions of assistance information are related to each other.*
3. *software architecture -- how assistance requests and their responses are communicated among a user, an operating system, application programs, and the assistance database.*
4. *contextual knowledge -- how much information is retained about the assistance environment, including the user, the application, and the tasks being performed.*

(From Sondheimer & Relles, [113])

These categories, however, view on-line help from a lower level than is desired in this thesis. Of the four categories, the second and third are primarily questions of implementation, while the fourth is also an implementation consideration in the sense that the implementor must decide how much context to preserve. For the purposes of this taxonomy, I would like to presume an arbitrarily fast computer with otherwise ideal hardware and sufficient memory to easily retain all relevant contextual information. (Recall that the topic is the design and evaluation of on-line help, not its implementation.) From this idealized perspective, it is easier to develop a user-oriented taxonomy of help systems. After that taxonomy has been developed, it will be scrutinized in Section 2.4 for implementation considerations.

In the sections that follow, I will describe seven dimensions along which the help system *as the user sees it* may vary. Three of these are issues of *help access*, among which is Sondheimer and Relles' first category, access mechanisms. Another three dimensions are issues of *help presentation*. The seventh, and possibly most important dimension of help system variation is *integration* -- the degree to which the various help features are uniformly available in all potentially relevant contexts.

Among the areas not included in this taxonomy are error messages, prompting messages, command languages and basic interface paradigms. Such topics are certainly important components in the usability and learnability of interactive systems, but are beyond the scope of this thesis. Eventually a taxonomy of on-line help systems should be subsumed by a larger taxonomy of user interfaces, which would include such topics.

Also not included in this taxonomy are any issues related to implementation. It seems premature to dwell on the appropriate underlying data structures for on-line help systems before it is clear how the systems themselves should work. However, it is worth noting that the data structures used in the prototype system described in Chapter 5 are sufficient to support nearly all of the functionality described here in an efficient manner.

2.1. Access Issues

There are at least three major issues involved in the user's access to a help system. These issues involve whose initiative first stimulates the help system's activity, how the user may request further help, and how complex the help request language is.

2.1.1. Access Initiative

The initiative in a help system may come from only two sources: the human user and the computer. In most systems, the initiative is strictly the user's. No help is presented until the user activates some explicit mechanism to request help -- for example, by typing the word "help" or pressing a key labeled "HELP". In such a system, the help component may be viewed as simply a utility program that is activated by specific commands.

In other systems, however, initiative may reside wholly or partially with the computer. In several tutoring systems (such as the WIZARD system [37, 109]) the program will intervene and suggest new approaches or provide new information when the user seems to need it. In more conventional systems, software carefully engineered for novice users (such as Lotus 1-2-3 [84]) may provide automatic help in the form of two-level menus that indicate some consequences of possible menu choices.[6] (Here, the border between on-line help as I have defined it and prompting messages becomes somewhat fuzzy.) Other possibilities for mixed-initiative systems include systems that automatically offer help if the user is idle for a certain period of time, systems that automatically attempt to interpret invalid commands as requests for help of some sort, and systems that maintain a constantly-updated display of help that seems appropriate for the current context, such as Lotus 1-2-3.

In classifying help systems according to access initiative, such systems are simply placed on a continuum between systems in which human users have the sole initiative and systems in which the computer has the sole initiative. In practice, it is very rare to find a system in which the computer more often takes the initiative than the human, but systems with somewhat mixed initiative are common.

2.1.2. Access Mechanisms

Despite the large number of on-line help systems that have been built, the number of access mechanisms implemented or even proposed is actually very small. I have been able to isolate only six such mechanisms. Of course, there is a wide room for variation in the implementation of these mechanisms; the most important aspect of such variation is access complexity, which is discussed in Section 2.1.3.

[6]See Section 3.1.8 for a more complete description of how this works.

The mechanisms discussed below are all discussed in terms of human-initiated help. However, the same mechanisms can be available for communicating with a system in which the computer takes the initiative; in such a case, they are simply mechanisms for requesting further help or clarification, rather than for requesting initial help. (The exception to this is contextually dependent help. Context is generally most useful for an initial help request, rather than in requests for clarification. However, context is an essential component of *any* help that is initiated by the computer.)

2.1.2.1. Key Word Help

Probably the most common help mechanism is the *key word* help request. Here the user simply specifies a key word which the system uses as an index to its help database. The sophistication of the key word mechanisms can vary greatly, however.

Most often, the only key words acceptable to a system are the actual names of commands. Although this is extremely common, due to the obvious ease of implementation, this method suffers from the equally obvious disadvantage that users can not get any useful help until they know the name of the command they need to learn about. (This effect is partially compensated for, in some systems, by allowing the system to print a list of the key words it knows about. By skimming this list, the user can try to guess which key word is the one he wants. But this quickly becomes unwieldy on large systems such as TOPS-20 [22], in which the list of key words may number in the hundreds, or in other systems in which the key words are simply inscrutable.)

A somewhat more sophisticated key word mechanism might search through a sectioned database for any key word that might be specified, printing out for the user those sections of the database that contain the key word. This approach is also quite easy to implement, but in a large database it can be very time-consuming. Some systems, such as the "key" command on CMU UNIX [118], modify this approach by only searching through a "header" area of each section of the database, rather than through every line of the database. This improves the system's performance at the cost of a substantial reduction in the number of key words recognized and the proportion of possible relevant entries found.

One problem with any method that involves searching through a textual database for an arbitrary word or phrase is that such unintelligent searching inevitably finds texts that are totally inappropriate. A request for information about the key word "see" might elicit information about a random number generating subroutine that requires a "seed" value.[7] Applying the computer's raw power to the task of searching a database can thus have the undesirable result of overwhelming the user with irrelevant information. (Such an information flood becomes somewhat more manageable in a system that facilitates scrolling, but the sheer volume can still exact a significant cost in the user's time, as the experiments in this thesis suggest.)

Unfortunately, what is probably the "right" technique for constructing key word help mechanisms is also the most difficult and expensive. Each section of the help database can be explicitly indexed by the relevant key words. This places a heavy burden on the documentation designer, in that it is crucial that he provide all such keywords, but it is the surest method for guaranteeing that all relevant key words will find the text, while also insuring that key word requests for common or short words will not overwhelm the user with useless information. This form of key word request is rare in on-line help systems, though it is more common in large information retrieval systems. This is the approach used by the prototype help system described in Chapter 5.[8]

2.1.2.2. Menu Help

The second most common type of help access mechanism is the *menu*. In a menu system, the user is given a list of help topics from which to select the topic or topics of interest to him. Details of menu implementation can vary greatly, generally along a simple continuum of ease of use. Menu selection can be as simple as pointing with a mouse, or can depend on a complex syntax for requesting the next menu item. Menu systems also differ in the mechanisms by which the first menu is created. Menus can be created in response to erroneous commands, in response to explicit requests for help, or as a follow-up to the previous menu selection.

Systems such as ZOG [100] have had some success with the idea of a menu system as the *only* mode of human-computer interaction; other systems that have not embraced the menu as the entire command language have nonetheless used it as the entire help system structure [66]. Nonetheless, the strict discipline of a menu may slow down the performance of experts[9] and may make it difficult for

[7]In this example, it might seem that an appropriate mechanism would simply require that an *entire word* match the key word being searched for, but this is inadequate in other ways: for example, real users will often try to search for a key word such as "del" instead of a sequence of key words such as "delete" and "deleting". Thus partial word matching is often desirable.

[8]The prototype system hand-codes all of the key words, a very expensive implementation in terms of documentation design time. Development of such systems in the future could be greatly streamlined with automated aids for the documentation designer, including, most important, an on-line thesaurus.

[9]Although this is a common opinion among computer experts, I have found no empirical evidence to support it.

naive users to find help on particular topics. Thus integration with other modes of help requests may be particularly valuable for menu-based help systems.

2.1.2.3. Contextually Invoked Help

Several help systems have had great success with providing information based on the user's current context. In such a system, information ranging from a partial command line to a user's entire history of use of the system can be used to determine the help most appropriate at the moment.

Obviously, an ambitious context-dependent help system could be a major programming undertaking. Yet the basic mechanisms that have proved useful in systems such as TOPS-20 [22] can be implemented fairly easily. The most basic such mechanism is a parser that analyzes the current partial command line and uses it as a help request, as is done in TOPS-20. Another simple mechanism would just keep usage statistics to show which commands and options a user had often executed in the past, and would use these to choose among possible interpretations of help requests. (Note that these statistics could be used in responding to any type of help request, not merely one based on the context of the current command line.)

A common failure of context-based help systems such as TOPS-20 is a failure to integrate the context-dependent help with other components of the help systems. That is, the system makes it impossible for the user to pursue more information from the help system without abandoning his command state. This issue will be discussed more fully in Section 2.3, below.

2.1.2.4. Graphically Invoked Help

With the advent of integrated workstation environments, utilizing bitmap display technology and pointing devices such as mice, new kinds of help access mechanisms have become possible. While these technologies facilitate improvements in the other help mechanisms -- menu selection, for example, is likely to be much easier with a mouse than without one -- they also allow the possibility of simply pointing at images on the screen and asking for help about them.

Since the technology is still quite new, on-line help's traditional position in the pantheon[10] of things important to computer programmers has predictably insured that not much has been done with these technologies in the area of on-line help. In particular, I have found no general-purpose help system in which the mouse could be used to point at any object, including pictures, on the

[10] **pan-the'-on**, *n.* 3. The aggregate gods of a people. (*Webster's Collegiate Dictionary*, third edition)

screen in order to request help about it. However, an early vision of such a system may be found in the GROK program at the Information Technology Center at Carnegie-Mellon University [83]. This utility program allows the user to point at any *text* on the screen to request help about it. It is perhaps best thought of as a graphically-aided key word help request system. An unfinished help system for the SPICE project at CMU uses bitmap technology to produce innovative help displays, with "push buttons" in the help windows allowing the mouse to be used to get further help [21]. This is essentially a variant of menu selection, in a more command-oriented display. It remains to be seen whether graphically based help systems such as these will eventually provide entirely new paradigms for help, or whether they will just facilitate improvements on other help access mechanisms such as menu and key word help.

2.1.2.5. Natural Language Help Requests

An obvious possible access mechanism for on-line help is natural language. After all, if the nature of the task domain (help requests) presupposes that the user is having some kind of difficulty using the computer, it seems logical to suppose that communication in the user's native tongue would be an invaluable assistance. Indeed, this kind of communication seems so natural that its desirability is accepted without question in the sparse literature on help systems. Sondheimer and Relles, for example, state flatly that "Ideally, we would like to allow users to enter assistance requests as questions in natural language." [113]

Such uncritical assessments are made possible only by the difficulty of implementing such systems. Panaceas generally only last until they are tried. And indeed, the difficulty of constructing natural language interfaces has so far prevented any realistic testing of them in the context of on-line help. For example, Wilensky [122] has developed a natural language consultant for UNIX, but has been unable to provide any real data about its usefulness for two reasons: its database is too small to be really useful, and it takes over a minute to respond to even the simplest of requests. In the absence of the technology to make such systems perform well enough for real users, it is certainly tempting to imagine that simple performance enhancements will one day allow such systems to solve all of the problems of on-line help.

Unfortunately, there are several reasons to suppose that this is not the case. Natural language is extremely verbose; thus it seems possible that the time necessary to actually type in natural language help requests will at least partially compensate for the ease with which such requests can be formulated in the mind. Moreover, expert users seem to rebel against any systems that force them into what might be considered "needless verbosity". Draper [27] has suggested that experts may in fact be the most important users of certain kinds of help systems; if this is true, it does not bode well for natural language help.

As part of this thesis, an experiment was conducted testing a simulated natural language help system, with results that will be disappointing to those who believe that natural language is the answer. These results are reported in Section 7.1.

2.1.2.6. Spoken Help Requests

One mechanism that has apparently never been used in a help system is speech recognition. Speech recognition has obvious potential usefulness in help systems; for example, a natural language system that understood spoken input would not be vulnerable to most of the criticisms of natural language help systems that were listed in the previous section. However, since speech recognition in a complex domain remains a topic of intense research activity with little practical success, this will not be possible in the near future.

Speech recognition has had more success in limited domains; systems such as HEARSAY-II [87] and HARPY [71] have successfully recognized vocabularies of hundreds of words. It is conceivable that limited-domain speech recognition could open up new possibilities in on-line help systems. No such work has yet been done, and time and hardware have prevented the investigation of such possibilities in this thesis.

2.1.3. Access Complexity

The third access dimension along which on-line help systems may vary is in the complexity of the mechanisms by which help is requested. Virtually all of the mechanisms described in the previous section may be implemented well or poorly from the user's perspective. The difference can be overwhelming in terms of its effect on the system's effectiveness.

For example, SHEPHERD [105] is a system designed to managed source files for SAIL programs and to organize their documentation. The system includes an on-line help system that organizes the information about the SAIL library into a tree structure that is, in essence, a menu system. However, the access mechanisms were sufficiently cumbersome that the system was rarely used by the community for which it was designed. In contrast, a well-designed menu system such as ZOG [100] has had enormous success with a wide variety of user groups, using the same basic access mechanism. The difference is that in ZOG, a menu selection can be made with a single keystroke, generally with a fair amount of semantic content. In poorer menu systems, such as the CMU LISP help system described in Section 3.1.6, a menu selection may necessitate typing the name of a path through a help network, such as "HELP PRINT-LPT-EXAMPLES". In general, a little extra programming can go a long way in improving such systems for the end user; for example, if the user had just viewed help by typing "HELP PRINT-LPT", it is not a major programming effort to have a follow-up request of the form "HELP EXAMPLES" translate to "HELP PRINT-LPT-EXAMPLES". Using a mouse to point at the word "EXAMPLES" is probably better still.

Of course, syntax issues are relevant to other types of help access mechanisms besides menus; in general, the simpler the syntax the better. This is true of user interfaces in general, but it is especially true in help systems because the systems must be useful when the user is in a state of confusion or ignorance. The last thing such a user needs is a help system with unnecessarily baroque syntax.

In addition to syntax, there are several other factors affecting the complexity of help access mechanisms. In menu systems, the branching factor is crucial: a system with too many choices at each level can overwhelm the user, while a system with too few choices can force the user to choose too often, making the process of actually finding the right information an arduous one. In graphics-based systems, issues of what icons should look like and how selection with a mouse should be achieved (number of clicks, meanings of buttons, etc.) are still in general unresolved; their resolution is obviously important for on-line help systems that use these mechanism. Finally, context-dependent help systems may react in a number of ways when help requests are ambiguous in context. Dynamically-generated menus that allow the user to choose between the ambiguous interpretations are clearly less complex and less frustrating to the user than a message which simply tells the user that his request is ambiguous.

Many of the help systems I have studied have suffered from these kinds of "minor" flaws. That is, their basic help access mechanisms have been sound, but the details of the interactions with the users have been unnecessarily complex or otherwise difficult. It seems likely that in the domain of on-line help, where the user's frustration level is very high to begin with, such small problems can have a high cost. Ultimately, they can cause such systems to end up like SHEPHERD: virtually unused despite the wealth of information they contain and the reasonable paradigm with which they are designed, simply because they are too frustrating to use.

2.2. Presentation Issues

On-line help varies greatly not only in how it is accessed, but in how it is presented as well. In this section, I will explore the three principal dimension along which the presentation of help information may vary.

2.2.1. Presentation Methods

The simplest and most common method of presenting help information is to simply throw it onto the user's screen, with no regard to what was there first. This is the teletype model of interaction, which views the user's terminal as a one-way device that can do nothing more than accept sequential lines of text. Although this model is hopelessly out of date, it is simple for the programmer to deal with and therefore underlies the majority of on-line help systems in the world today. Its greatest flaw is that it almost always causes the user's previous context to scroll of the screen, so that he can no longer see the result of the interaction with the computer that caused him to request help in the first place.

Another presentation method that is almost as easy for programmers is to simply rely heavily on a printed manual. The on-line component of a help system can simply tell the user which part of the manual to look at, thus serving as no more than an "electronic index". This drastically reduces the volume of output from the on-line system, thus at least partially preserving the user's context on the screen. The great drawback of this system, however, is its reliance on paper. A user without a paper manual is doomed, and a user with an outdated manual can be in even worse shape: instead of no information, he may have erroneous information.

Video screen technology has allowed more modern help systems to use multiple windows. In such a system, the user can preserve his context at the bottom of the screen, for example, while scrolling through help texts on the top of his screen. The penalty here, of course, is that the available screen size for each of these activities is only half of the screen size to which the user is otherwise accustomed. Large displays such as those found in the new generation of workstations will alleviate this problem to a large extent. If help systems that make use of multiple windows become commonly available in tandem with the larger screens, even experienced users are less likely to resent the screen territory that is allotted to the help system.

It should be stressed, however, that despite the penalty that multiple windows can impose on screen "real estate," the overall effect of windowing technology is extremely beneficial, and facilitates a host of new techniques. Multiple windows and multiple processes (which facilitate help processes running independently of the application program) have fundamentally changed the landscape in the world of help systems.

Finally, the future holds the promise of new technologies that might be useful for the presentation of help information. Synthesized speech might provide help without sacrificing any of the screen territory, though no experiments have been conducted on such a system. (Synthesized speech has often been of a rather low quality, which could make it less comprehensible and thus less useful in a help system.) New displays also make it plausible to include pictures, animations, or even videotapes as part of a help message. Thorough investigation of these possibilities will require substantial effort in both hardware and software development, but a first glimpse of these may be seen from some of the tutorial programs that come with the Apple Macintosh computer.

2.2.2. Presentation Source

Besides the question of how the help information is to be presented, there is the question of where it is to come from. This may seem to be primarily an implementation question: text may be clipped out of a monolithic on-line manual, it may be retrieved from a network of help texts, or accessed by key word from a relational database. However, there is a more fundamental issue here as well. The text can either be retrieved verbatim from some data structure that contains it, or it may be generated "on-the-fly" by some natural language composition mechanisms acting on an underlying knowledge representation.

Dynamically generating text from an underlying knowledge representation is the kind of project that computer scientists love, and indeed many current researchers are taking this approach to on-line help [40, 91, 117]. However, it seems reasonable to question the motivations of such an effort. Certainly getting computers to generate reasonable natural language from a knowledge representation base is a fascinating and useful research topic. But in the domain of on-line help, presumably, the primary goal is not to build an on-line help system that is theoretically interesting but rather to build an on-line help system that helps the user. In this light, there is no evidence to suggest that dynamic generation of text is likely to produce higher quality texts than human documenters would produce if they prepared all the system's texts in advance. (Indeed, the output of language generation programs suggests that the opposite is true.) However, it may be useful to use language generation facilities to dynamically generate examples, or to customize explanations to a specific context in which the user is having difficulties. Given the current state of language generation, this does not yet sound like a very even trade. Still, a state of the art help system might want to provide standard texts for standard situations, dynamically generated examples, and dynamically generated or dynamically modified texts for unusual situations, most notably situations involving ambiguous help requests where none of the "canned" texts unambiguously apply.

2.2.3. Text Quality

The final dimension along which help presentation varies is that of *text quality*. Help texts are English prose, and as such vary in quality as widely as the readings and compositions vary in a high school English class. The literature on text readability is enormous, and nearly all of it applies to help texts.

What this suggests primarily is that such texts should be designed by a specialist. Nonetheless, it is possible for the non-specialist to evaluate his own texts in the light of certain very general criteria. It should be remembered, however, that most of these have not been experimentally studied for their importance in the particular domain of on-line help systems.

- ***Readability***: Several standard measures exist by which text readability can be measured. National magazines such as Time and Newsweek have determined that their texts are best accepted when written at a seventh or eighth grade reading level. Documentation writers can use automated tools such as the UNIX style utility [15, 16] to determine the reading levels of their texts. The current study does shed some light on the usefulness of some of these tools, as explained in Section 7.3.

- ***Organization***: Familiarity with the task domain and user population of a help system can help the documentation designer to organize his texts intelligently. In addition to standard considerations of reasonable structure, knowledge about the tasks and users can help to put the most commonly-needed texts up front. Little details such as these can save enormous amounts of user time and frustration cumulatively over the life of a system.

- ***Formatting and headings***: Designers of paper documents have long been aware of the importants of headings, font size, and similar considerations of layout [33]. With the advent of bitmap display technology, these same issues are becoming relevant to on-line help designers.

- ***Chunk Size***: Complementing meaningful, highlighted headings should be texts divided into small pieces. If the headings are meaningful and easy to read and notice, then they allow the user to only read the relevant sections of text. In order to make this work, the text should be divided into sections that are as small as is reasonably possible, thus minimizing what the user actually has to read. (How small is small enough? It seems likely that this depends on several factors, including task complexity and display characteristics. Certainly it is best if the texts fit in a window without any need to scroll, but even this is not always possible.)

- ***Voice and Orientation***: It seems likely that use of the passive voice, anthropomorphic orientation, and similar considerations of style may heavily affect the usefulness of text. While passive voice seems to be universally condemned, other issues such as anthropomorphism are more disputed. Houghton [55] declares it to be absolutely wrong, but at least some professional documentation writers seem to think that texts in the "You User, Me Computer" style put users at ease and make the texts generally clearer. Commodore Business Machines has graced the literature with several extreme examples of this style [20] without anything remotely resembling negative effects in the marketplace.

2.3. Integration

Finally, perhaps the most important aspect of an on-line help system as a whole is its level of integration. Many computer systems provide several on-line help mechanisms, each with its own database, operating completely independently. The experimental system designed for this thesis offers several methods of accessing the same information, making it easy to switch between methods when appropriate.[11] Similarly, some systems make on-line help an independent utility, accessible only when you're not doing anything else, while others try to make the help available within the context of most other, larger tasks.

The virtues of integration are obvious: by providing uniform access to help, you eliminate confusion for the user and make it easier for him to stay in context. By making various mechanisms access a single help database, you make it easier for the user to try all of the mechanisms you provide in his attempt to learn what he needs to know. Of course, integrated help is more challenging than non-integrated help from an implementation perspective, but the prototype help system described in Chapter 5 demonstrates that the problem is neither impossible nor, indeed, particularly difficult.

An excellent example of the need for integrated help systems was found in the observation of a TOPS-20 user attempting to create a subdirectory for the first time. This user first used the list of help topics that TOPS-20 provides as an index to its key word help system, and managed to determine that the "BUILD" program was for building and modifying subdirectories. After typing "build", the user typed a question mark. This activated TOPS-20's context-sensitive help facility, which told her that the next thing she should type should be a directory name. Unfortunately, this user did not realize that in TOPS-20, directory names are almost always surrounded by angle brackets. After trying unsuccessfully for several minutes to get the program to accept her version of a

[11] I know of no other system that has done this, although Fenchel's thesis [34]described a system designed to work this way but never fully implemented in this regard.

directory name, the user had to get out of the BUILD program entirely and use the key word help system to search for an explanation of what directory names look like. Had the context-sensitive help mechanism and the key word help mechanisms been integrated using a common database, she could have gotten this information from within the BUILD utility, without sacrificing her entire context.

Moreover, integrated help systems would be helpful to the documentation designers as well as to the users. In non-integrated systems, several different databases of help information must be created and maintained, which is inevitably more work than maintaining a single database.

It should be noted that an "integrated" help system, in which all help systems access the same database and are available together in all contexts, is not at all the same as an "integral" help system. An "integral" help system is one in which the help is provided by the same program that executes the command. The help is viewed as an essential part of the program's functionality, hence the term "integral". The argument in favor of integral help is that the help system can provide better help because it has access to the program's underlying data structures. While this may be true in the case of an extremely sophisticated help system, it is unlikely in most cases to compensate from the loss of a uniform help system available for use in all programs. Essentially, this is the same trade-off involved in the choice between designing sophisticated user interfaces for each program, or designing a single user interface management system which will communicate with all of the underlying programs. You may sometimes have to sacrifice a little functionality if you really want a well-designed, uniform interface.

It is also worth noting that it is the availability of multiprocessing capabilities that makes it reasonable to consider an elaborate non-integral help system of the kind built for this thesis. In an environment without multiple processes, it would be much harder to design an integrated help system that preserved any arbitrary command context.

2.4. Implementation Considerations

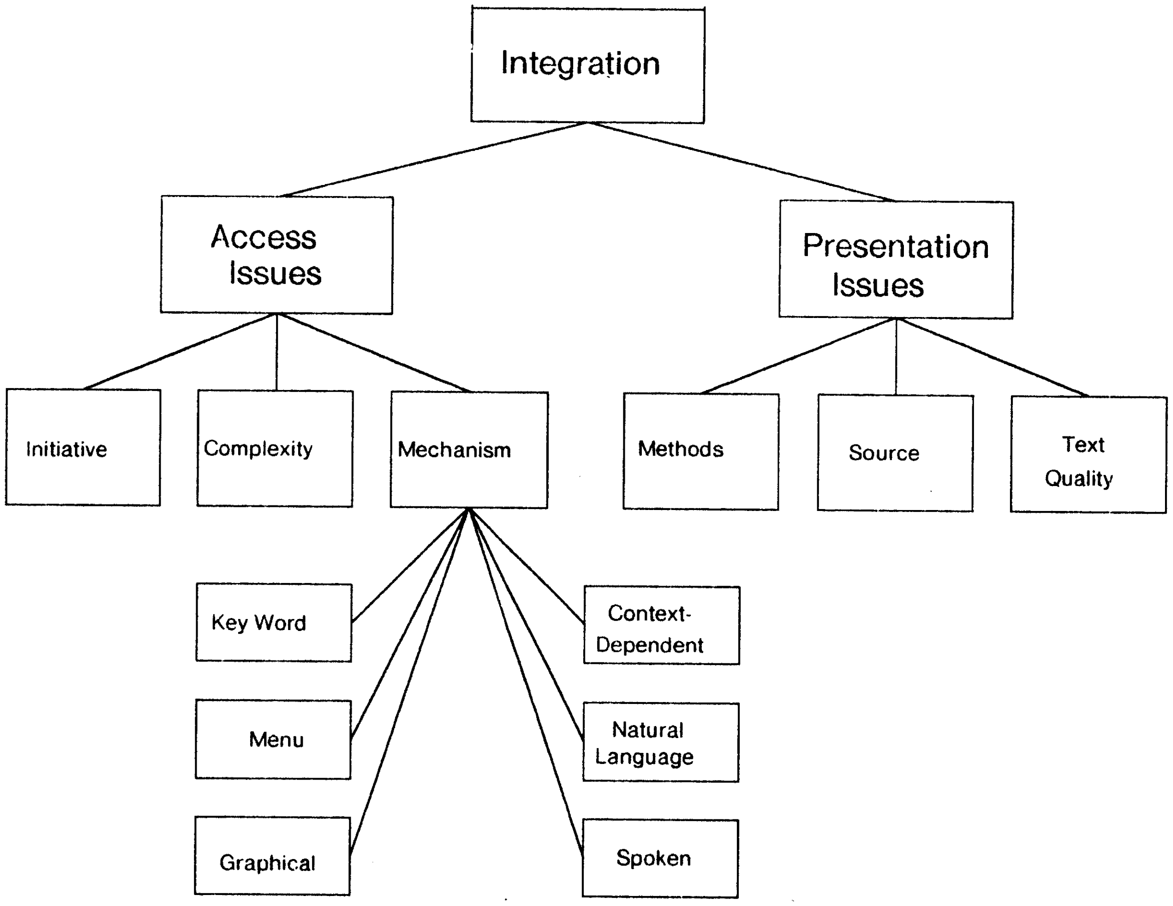
In the preceding sections, I have discussed several dimensions along which on-line help may vary. In most cases, the variation can be clearly supposed to be for the better in one particular direction. What has not been considered is the cost of implementing some of the help mechanisms described. Such a cost assessment and implementation analysis is beyond the scope of this thesis. However, it is worth noting which things are particularly hard.

In particular, sophisticated graphics are hard, natural language is hard, speech recognition is hard, and context-dependent help can be arbitrarily easy or hard, depending on how far the system tries to go. Aside from these things, however, all of the help mechanisms described in this chapter can be implemented reasonably inexpensively at quickly by any good programmer. Why this hasn't generally been done is a very good question.

2.5. Summary

The dimensions among which help systems vary, as described in this chapter, are pictured in Figure 2-1. This chart may be helpful in considering the design of help systems as yet unimplemented. In the next chapter, I will use the taxonomy to describe a number of help systems actually used in the real world, and to explain the successes and failures of these help systems, as reported by their users and observed in user protocol experiments.

Figure 2-1: The Dimensions of Help System Variability



Chapter 3
A Survey of Existing Help Systems

3.1. On-Line Help Systems for User-Oriented Software

In this chapter, I will discuss a number of real world help systems in the perspective of the taxonomy of help systems presented in the previous chapter. The list of systems discussed in this chapter is far from exhaustive; it is intended instead to be representative of the diversity of help systems that have been previously implemented. Each of the systems will be described briefly, and its place in the taxonomy of the previous chapter will be explained. That taxonomy will also be used to describe simple changes that might significantly improve the help system.

In what follows, reference will be made at several points to "users' comments". These are simply the comments users made in response to a very open-ended survey about help systems. Users were asked, among other things, to provide examples of the best and worst examples of help systems they had used, along with their ideas about why these systems were good or bad. The responses displayed an interesting tolerance on the part of the users; many more positive comments were made than negative ones, even in the case of extremely simple-minded or even obviously poorly-designed help systems. It seems that users are so grateful for any help at all that they tend to mention the positive more often. Or, perhaps their experience with getting help is so generally negative that anything at all helpful really stands out in their minds. Typical comments began "I hardly ever find help systems useful, but..." The observations below are based on the "buts" these users provided, and (in many cases) on observations of experienced users of the systems trying to get the help systems to help them perform new tasks.

3.1.1. TOPS-20

TOPS-20 [22] is an operating system that runs on large DECsystem-20 computers. Its help system can be briefly characterized as coming in two poorly integrated parts. The first part is a very simple key word help mechanism, with a simple syntax ("HELP <command name>"), a rudimentary presentation method (simply printing text, no scrolling or windowing), and pre-written text of erratic quality. The second part is a context-dependent help mechanism, invoked by the simple method of typing either ESCAPE or a question mark at any point in a command line. This provides an extremely short help message, without the possibility of obtaining any further information in context. The help system is completely human-initiated, and although the context-dependent help is well integrated into the system as a whole, the help system overall suffers from a very low level of integration.

TOPS-20 had the help system most commonly cited as "best" in my user surveys.[12] The biggest advantage of TOPS-20 appears to be the nearly uniform availability of context-dependent help. Users can get very short syntactic help at any time by typing a question mark. However, there is only one layer of context-dependent help available. If the user doesn't learn what he needs to know from the short help provided, he can't find out any more in context. In such a case, the user must resort to TOPS-20's *second* help system, a very primitive key word system. To use the key word system, the user must abort any partial command context, and issue the "HELP" command to the operating system. "HELP" must be followed by the name of a TOPS-20 command; such a request will cause the *entire* documentation on that command to be printed on the user's terminal, without even pausing after each screenfull of information.

Obviously, TOPS-20 suffers from a very low degree of integration. (In fact, it was in the observation of a subject using the TOPS-20 help system that the need for help system integration first became apparent, as explained in Section 2.3.) Merely being able to access both help systems from an embedded context -- that is, to issue key word help requests without forfeiting one's command context -- would be a significant improvement.

However, the improvement would be more significant if the key word help that was being integrated was of a higher quality. TOPS-20 offers no help at all to the user who does not already know the name of the command he is trying to use; there is no provision for lookup of synonyms.

[12]Of course, TOPS-20 is also one of the most widely-used of the systems listed here. Nonetheless, the mentions of TOPS-20 were almost uniformly positive.

There is also no conceptual help: help is provided only for commands, not for such vital concepts as "file" and "directory". All of these could easily be provided without significantly changing the TOPS-20 help access methods.

Additionally, the presentation of TOPS-20's key word help leaves obvious room for improvement. At the very least, it could easily be modified to pause after each screen full of information had been printed. (Actually, a TOPS-20 terminal setting command can achieve this effect, but it is not a part of the help system, is not documented with the help system, and is hence effectively unknowable for novices.) With only a little programming effort, such texts could be made to scroll through the top of the screen while preserving context in the bottom.

Finally, the quality of the TOPS-20 help texts is erratic, a common circumstance in a system where the drafting of help texts has been left to the implementors of the relevant programs. Some of the texts are of quite high quality, but this is largely a matter of chance and the temperment of the individual programmers.

With all of the criticisms listed here, it is worth noting again that TOPS-20 is commonly cited as having one of the world's great help systems. This reputation is probably in large part attributable to its competition: few other operating systems offer any kind of context-dependent help at all. The feature does seem to be enormously popular with users.

3.1.2. Emacs

Emacs in both its UNIX [44] and TOPS-20 [115] versions provides an interesting assortment of help features. Each provides two kinds of help: a straightforward menu system, and a context-dependent help system which is activated by typing a help character at various points in the middle of commands. These two help components are entirely non-integrated. As with so many other systems, the text's varied origins are reflected in its erratic quality. The initiative is primarily human, although some versions of Emacs automatically invoke the context-dependent help when certain command errors are made. (This is a customization option; users can choose whether or not to let the computer take the initiative in such situations.) The presentation methods are fairly sophisticated, preserving context or restoring it when finished, which is not surprising in a text editor which already implements a great deal of screen management software. Help texts are primarily pre-written, but are generated dynamically in certain cases where the information is highly context dependent (e.g. a list of valid commands, which depends on what extension packages have been loaded).

Probably the most common complaint about these systems is the lack of uniform availability of the context-dependent help. This is because packages written in Emacs' extension languages do not have easy access to the context-dependent help-mechanisms. This fact and the fact that the two help system components are entirely independent lead to the characterization of Emacs as having an extremely non-integrated help system.

Especially interesting is that, despite the structural similarities of the two systems, the TOPS-20 version of the help system generally seems to meet with much more approval from its users than the UNIX Emacs help system. This is not the case with Emacs in general, which seems to be better liked on UNIX. Possibly the difference can be attributed to the fact that the menu system in UNIX Emacs is very loosely structured, forming a menu tree that is too bushy and not deep enough. This underscores the need for careful structuring of a menu system, and may be seen as a failure of UNIX Emacs to meet the general requirements of menu help systems. Of course, other factors may also be responsible, but I have not yet isolated any. In general, the help systems are sufficiently similar in mechanisms that it seems hard to avoid the conclusion that any differences are due to content -- help text and menu structure -- rather than to the help mechanisms themselves. (The different perceptions of the Emacs help systems may also be due in part to differnces in their user communities, but the perception seems to hold even among those who use both. Many UNIX Emacs users have moved from TOPS-20 because they prefer the UNIX version, but remark that the only thing they miss is the help system on the TOPS-20 version.)

3.1.3. UNIX

UNIX [118], the well known operating system which runs on a wide variety of computers, was widely cited in my user surveys as an example of both a good and a bad help system. Its help system is based entirely on the key word approach. There are two key word help commands. The ***man*** command can be used to print out the entire documentation for a UNIX system command. The ***key*** (or, on some systems, ***apropos***) command will take a single key word as its argument and will suggest a number of commands that may be related to the key word; such commands can then be looked up with the man command. The system is entirely human-initiated. Access complexity is slightly higher than it should be, in that there is really no reason why key and man need to be separate commands; a single help command could act like man when its argument is a command name and like key when its argument is not. System integration is also fairly poor; the help is only available from the top command level, with a very few exceptions, and there is no special provision for the common situation of choosing one of the key command's suggestions as the argument to the man command.

Presentation methods are erratic: The man command's output is paginated, so that it won't write more than a screenfull of information at a time. The key command is not so careful, but its output is not often longer than a single page anyway. All of the texts are prewritten, although the key command does select very small pieces of the provided texts. As in most systems where programmers write the documentation, the text quality is erratic, but not as much so in a system such as TOPS-20, where some of the text is quite good. On UNIX, the text is almost uniformly bad.

The popularity of the UNIX help system seems to stem from the fact that the key command constitutes a reasonable index to the on-line manual. (That is, key words often suffice to find the command you're looking for even if you don't know its name.) A few individual complaints about UNIX's help centered on particular failings of this indexing scheme, which is useful but certainly not optimal. (The key command only recognizes as relevant key words those words which occur in the first line of a command's documentation. This is better than only recognizing a command name, but not nearly as useful as a true thesaurus-based or explicitly-specified key word system.)

Other problems with the UNIX help system include the lack of any context-dependent help and the incredibly slow performance of the man command. (The man utility reformats its help texts with a text processor every time you ask for help, a staggeringly poor design decision.)

The fact that many users seem to like UNIX's help system despite its manifold defects is indicative of the high value of key word help systems that can aid users who do *not* know the name of the command they are trying to use. Aside from the mechanism which reformats help texts every time they are printed, the UNIX help system is a fair example of a well-designed, nearly minimal key word help system -- that is, a system which can help its users with only a modicum of actual help mechanism. Experiments in this thesis, reported in Section 7.1, suggest that such a system is indeed highly useful when the texts it provides are of high enough quality.

3.1.4. PLOT

PLOT [30] is an interactive program for producing graphs from sets of data. PLOT provides key word style help, but like TOPS-20 requires that the user already know the appropriate command name in order to access this help. Some context-dependent help is available in the form of a list of syntactic options at some (but not all) points in the command line. A very good tutorial for novices is also available. In addition, a special provision is made for editing the entire manual with a text editor. The system thus provides a wide variety of access methods, as an integral part of the application but with the help access methods not at all integrated together. The presentation methods are very simple, with text simply printed on the screen as if it were a teletype, but the text is generally presented in small enough pieces that the user's context is not entirely lost; he can still see what he was doing at the top of the screen. With the exception of the tutorial, the initiative is completely the user's. Quality of the texts, all pre-written, is unusually high, reflecting the fact that they were written by a single author (Ivor Durham) who was highly concerned with making the system easy for novices to understand.

PLOT's help appears popular primarily for its completeness, both in terms of content and the number of ways of accessing the help information. Its failings usually appear as failures of completeness, which, given the enormous amount of effort involved in its on-line help, points to the desirability of a uniform help database. Typically, information available via one of PLOT's help mechanisms is not available from the others, so that users have to keep trying them until one is found satisfactory. The lack of uniform availability of context-dependent help is also a deficiency. Thus PLOT appears an excellent example of how a help system can be very good on most of the details and still suffer from a lack of integration.

3.1.5. Shepherd

Shepherd [105] is a system designed to manage source files for SAIL programs and to organize their documentation. The system includes a component that serves as an on-line help system, organizing the information about the SAIL library into a menu-like tree structure. The system is noteworthy for the attempt to provide uniform documentation facilities for a variety of programs written by different programmers in different contexts. Unfortunately, the program is extremely slow and the interface cumbersome, so that the system was apparently never heavily used nor well-liked. This example illustrates the importance of access complexity. Even with a fairly large database of useful information, the program was poorly utilized, largely because the interaction syntax was frightening. As remarked before, this is even more likely to be prohibitive in a help system than in other interfaces, because users begin using help systems with an already-high level of frustration and in a state of acute awareness of their own ignorance.

3.1.6. CMU LISP

Before the reader concludes from the previous example that a complex access mechanism is the seal of doom for a help system design, he should consider the case of CMU LISP. CMU LISP [19] has a large and apparently thorough static help system using a menu approach. As with Shepherd, the mechanism for accessing the help is cumbersome. Each node in the tree of help menus has a unique name, and must be accessed by it. Typically, each node also lists some other nodes that are related, but provides no easy way for the user to get to it. Therefore he must generally type "help" followed by "help foobar," "help foobar.options," "help foobar.options.details," and so on. In addition to the cumbersome syntax for help requests, the system suffers from primitive display methods and erratic text quality.

Despite its flaws, which also include the total absence of context-dependent help, CMU LISP was mentioned only positively in the user survey, possibly because so few programming language environments offer any help at all. The system was praised for the completeness of its database, which is of course a factor of overriding importance. Apparently users were generally able to get the information they needed from CMU LISP's help system; the information was there and was not impossible to obtain. In this case, the users were willing to forgive the system its other flaws. Future designers of programming language environments, including syntax-based editors, would do well to consider the popularity of CMU LISP's help system, which had no context-dependent features at all, before they devote too much time to the mechanisms of their environments and not enough time to the contents of the messages provided.

3.1.7. TOPS-10

TOPS-10 is an operating system that runs on DECsystem-10 computers. The Carnegie-Mellon University TOPS-10 help system [18] is composed of just two commands used to access help in a key word system. "Help <topic>" gets a short message explaining the topic, while "Doc <topic>" elicits more details. Most topics are simply program names, but a few more general topics also exist. For example, "help dialup" yields a list of phone numbers for dialup lines. There is no indexing and no context-dependent help. The two commands are totally unintegrated, and the presentation method is the most basic possible: the texts are printed on the screen without even a pause as the page fills up. Texts are of predictably erratic quality, given that they are written in general by the implementors of the programs they describe.

In short, TOPS-10 help is about as simple and primitive as possible, with the exception of the unusual fact that there are two levels of detail available. Nonetheless, this simple fact was enough for several users to single it out for praise. This appears to indicate a great appreciation on the part of users for some structure in the help database. Those who commented favorably about this aspect of TOPS-10 help appeared to be mostly people who had otherwise used only systems that provided all the help in a single undifferentiated mass. Again, users are so accustomed to absolutely minimal help systems that they are grateful for even the smallest improvements.

3.1.8. Lotus 1-2-3

Lotus 1-2-3 [84] is an interactive spreadsheet/database program on the IBM PC and other personal computers. It incorporates (as one of its major selling points) a very impressive help system. The basic system is menu-driven for commands, with one menu item always highlighted on the screen as "selected." A second-level menu is continuously updated to display what the menu choices would be if the currently selected menu item were chosen. This provides a form of context-dependent help without human initiative. In addition there is an elaborate static help menu, which is often entered in a context-dependent manner -- the starting help menu is chosen according to the user's context. The texts are apparently all pre-written and are of extremely high quality, reflecting an obvious emphasis on documentation by the Lotus Development Corporation. As one would expect in an integrated system such as Lotus, presentation methods are sophisticated, with the screen carefully managed to preserve context when possible and to restore it later when it was necessary to overwrite important state information with help texts. The complexity of help access is extremely low; it is an excellent example of what a menu-based help system should look like.

The menu itself is generally very well-structured, making it very easy for novices to learn about the system. For experts, the story appears slightly different. The one experienced Lotus user I was able to interview did not think very highly of the on-line help, because she was unable to easily get answers to specific questions. The menu system apparently thwarted her in her quest to quickly answer such questions. This suggests that even the most elaborate menu systems may be insufficient as complete help systems, and that a key word component will be extremely valuable to experts. (Limiting a help system to step-by-step traversal of a menu network seems a bit like taking away a driver's mode map and forcing him to find his way over a long distance using only road signs.) However, since the user reported that she often simply could not find what she wanted with the help system and resorted instead to the printed manual, another explanation is possible: it may be that Lotus' help database is simply not sufficiently comprehensive. Obviously if the information isn't there, the user can't find it, no matter how clever the help access mechanisms might be.

3.1.9. RdMail

RdMail [67, 31], a TOPS-10 mail management program, was singled out for a large number of comments, both positive and negative, in the user surveys. Its help is entirely key word based, but with sufficient indexing to make it quite often useful even when the user does not know the name of the command he needs to use. The system uses key words as indices into the printed manual, and prints out the sections that seem to match the key words. No attempt at screen management is made.

The text quality is quite good, reflecting the effort that went into the writing of the RdMail manual. The help system is simple to use, and since only one help mechanism and one command level are provided, integration is not really an issue. There is no notion of levels of explanation, and no context-dependent help except in the correction of spelling errors.

A common complaint about RdMail's help system was the volume of material help requests can elicit; typically a help request will cause RdMail to select a half dozen or so paragraphs of help text, and will show you the heading of each and ask if you want to see the entire section. This illustrates a problem that comes as a natural consequence of a thorough key word index: too many commands may match a given key word. In such a system, it is important to provide a simple mechanism -- some form of menu selection is the obvious choice -- by which the user can choose among many possibly relevant topics. RdMail asks a yes or no question for each such topic, which is time-consuming and often frustrating for the user, since the desired information is just as likely to be asked about last as first.

Another common complaint about RdMail is that the key words are simply poorly chosen; it is difficult at first glance to reconcile this with the complaint that key words yield too much information, but they may in fact be the same problem. Users who begin iterating through the many choices key word produces in RdMail may simply conclude that they have chosen the wrong key word and give up, never realizing that they had the right key word but simply had to sit through a large number of wrong choices before RdMail would give them the right one.

3.1.10. Ci

Ci [106] is a command-interpreting front end for UNIX application programs. It provides key word static help, using short descriptions for its indexing much like the UNIX key command described above. It also provides a special character ('*') for requesting context-dependent help; placing this character in a command line turns the command line into a request for syntactic help in many contexts. Texts are supplied by the application programs.

Ci is interesting primarily in contrast to TOPS-20. TOPS-20 offers a less complex help interface -- typing a question mark in the middle of a command line, and maintaining context, is certainly easier than typing an entire line with an asterisk marking the missing information -- but ci integrates the context-dependent and key word help systems to a greater degree than TOPS-20. In TOPS-20, the application programmer may design his subcommand interface to utilize the built-in help mechanisms only for context-dependent help, whereas ci-based UNIX applications can take advantage of both types of help. Unfortunately, the number of ci-based applications is not very large, so it is not possible to tell from experienced users whether this does in fact make the ci help facility significantly better than TOPS-20's.

3.1.11. VM/CMS

VM/CMS [58] is a widely-used IBM conversational operating system. It incorporates a menu-based help facility that seems very complete with respect to the programs documented, but less satisfactory with regard to fundamental concepts of the system. A modicum of context-dependent help is provided as context-dependent entry to the menu system. One bad feature of the system as a whole is the complexity of its user interface; most programs can take commands either via a conventional command line or through a sophisticated graphical interaction program, but the two methods of giving commands are not equally well documented in most cases, for no apparent reason.

This is a difficult problem to avoid: the difficulty of providing a coherent integrated help interface seems to increase dramatically as the fundamental underlying complexity of the system increases. Those who regard the fundamental VM/CMS interface as too complex would probably claim, with some justification, that trying to fit a good help system onto such a complex underlying domain is a task doomed from the start.

3.1.12. SARA

SARA [34, 35] incorporates an experimental help system most noteworthy for its representation of help information rather than for its presentation of it. Help is available through queries of a static database in a rigid command language, although what is actually happening is quite similar to the CMU LISP method (Section 3.1.6) of accessing a menu network by giving the full name of each help frame. The ability of the system to provide context-dependent help is restricted by the inability to process input one character at a time, which virtually rules out meaningful context-dependent help.

SARA was probably the first system to approach on-line help as an integrated database to be probed with multiple access methods. As such, it was a significant landmark, inasmuch as the integration of multiple help functions is considered to be an important component of good help systems. However, most of the work on the SARA system focused strictly on that database representation of help information; after it was constructed, only a single help access method was actually implemented to use it! The data representation used in the prototype help system described in Section 5 is derived in some measure from the representation used in SARA.

3.1.16. ANLHS

ANLHS [117] is an ambitious attempt to design a natural language question-answering help facility. The author is principally interested in the knowledge representation aspects of the problem, and is apparently largely ignoring the interface with the end user. (This system will receive help requests in a formal language, and is not designed as the front end of the final system.) Several efforts are currently under way to apply natural language to the domain of help systems, but the experimental results reported in Section 7.1 tend to cast some doubt on the general usefulness of this approach.

3.1.17. ZOG

ZOG [97, 99, 101] is the menu system *par excellence*. The ZOG system was developed as an integrated user interface based entirely on menus. The system includes a well-fleshed-out help system, which is naturally entirely menu-based. (However, the fact that help is always invoked from some known previous menu loation allows the help to be somewhat context-dependent with no special effort at all.) The system places a premium on integration, and the help facility is entirely integrated with the system as a whole. The system evolved in an atmosphere of extensive testing and refinement of all of the details of the system, with the predictable result of a smooth interface, minimal interaction complexity, and high quality texts and presentation methods.

Because the help system is embedded in an interactive system that is entirely menu-based, the menu approach appears to completely eliminate the need for syntactic help. This is in keeping with the general ZOG philosophy of an extremely straightforward and consistent user interface. With such an interface, help needed is nearly always conceptual rather than syntactic, and for this menu help seems to suffice. However, this sufficiency of menu help probably does not generalize to the non-ZOG environment, where command syntax can be extremely complex, and where large branching factors may make a menu system unwieldy.

3.1.18. COUSIN

The COUSIN project [42, 50, 51, 52, 53] has been an evolving series of systems investigating cooperative user interfaces. In its early versions, COUSIN used ZOG's menu system to automatically generate help of various kinds for non-ZOG interfaces. This has manifested some of the fragility of the success of ZOG's help facility. In the non-ZOG context, help was not always sufficiently cross-referenced, with menu frames taking on inappropriate sizes and inadequate links. Nonetheless the

3.1.13. How?

HOW? [56] is an associative network-based help facility for LISP, most interesting for its knowledge-based approach. The help database is a network suitable for normal menu traversal, but may be accessed via key word requests which are interpreted in a sophisticated and flexible manner, using a complex LISP program. It provides no context-dependent help, and the presentation and text quality have not been emphasized, since the research has focused on knowledge-based interpretation of help requests.

Such interpretation focuses primarily on questions of customization and user state and history. The primary advantage that this knowledge seems to offer over more naive key word approaches is in a pruning of the search space for highly ambiguous requests. The idea of using knowledge about user state, preferences, and history to disambiguate help requests has obvious appeal, but it will probably be a difficult one ever to test conclusively. Such tests would of necessity monitor users over a rather long term, as they developed a history of using the system. Thus it is unlikely to be seriously tested until an implementation is carried to the point of being practically useful in a well-used environment.

3.1.14. The IA Tutor

The IA Tutor [103] is an on-line intelligent tutor for a specific application, namely a military message service. The tutor uses a fairly simplistic knowledge representation to guess about the type of help a user needs, utilizing in the process a fairly extensive user profile for customization and individual tailoring. As with HOW?, there is no evaluative data available, and the "intelligent" component can not easily be studied outside its unique task domain.

3.1.15. WIZARD

WIZARD [37, 109] is a knowledge engineering effort that produced a small help system able to volunteer advice based on preconceived "plans" and "bad plans" it hypothesized might be in the user's mind. As such, it is an unusual example of a help system in which the initiative rests almost entirely with the computer rather than the user. It apparently does a good job helping novice users of VMS, but would require a great deal of effort (both initial and ongoing) to become a practical help system for VMS in general. Such systems will need to be much more thoroughly understood before they can be integrated with the kinds of help discussed in most of this thesis, and before they can be studied in sufficiently complex domains to be rigorously tested against more conventional systems.

system demonstrated that much can be done by way of generating both static and context-dependent help from a static database.

Later COUSIN systems have offered more sophisticated context-dependent help in the context of its unusual interaction methods. Such help methods correspond closely to the kinds advocated in this proposal. In general, though, COUSIN has concentrated on minimizing the need for help rather than on giving elaborate help, so the help systems that have been implemented have not been fleshed out with sufficient texts to test them thoroughly. A fascinating topic for future research would be to develop a version of COUSIN that incorporated some of the help techniques recommended here, and to test various subsets of that COUSIN to see which features are most important for enabling novice and expert users to execute unfamiliar tasks. Without such studies, it is virtually impossible to weigh the relative importance of graceful interaction paradigms and clever help systems.

3.1.19. STAR

The STAR system [111] is a workstation that uses icons and graphics to provide its fundamental interaction mechanisms. STAR provides key word static help, with help entries pointing to other entries in a menu-style network, without menu access mechanisms. A limited amount of context-dependent help is available in the form of context-dependent behavior when help is invoked. Syntactic help is apparently not provided because it is expected that the unusual interaction paradigm of the STAR system (graphics and icon-based) will eliminate most of the need for such help. The validity of this expectation remains untested.

3.1.20. The Berkeley UNIX Help System

A new help system [66] in use at the University of California at Berkeley utilizes the UNIX directory structure as an easy way of implementing a tree-structured menu help system. The system allows for progressive deepening and can suggest related topics using this menu mechanism. However, as with some of the other menu-based systems reported above, this help system requires an extremely baroque syntax to specify movement through the help menus. As such, its practical usefulness is questionable. Since the system has been released for general use at Berkeley, it seems likely that these questions will be answered, and this system as it evolves should provide some interesting data on the usefulness of menu-based help systems with awkward access mechanisms. However, no such data is yet available, as the system is still brand new.

3.1.21. UC: The UNIX Consultant

Wilensky [122] reports on an experimental help system that answers questions about UNIX in natural language. Unfortunately, the system is both too slow and insufficiently broad in its database to be useful to real users, but it demonstrates that natural language may in fact eventually be practical, from an implementation standpoint, as a help system interface. Whether this would be *desirable* is another question, and the results reported in Section 7.1 of this thesis cast doubt on the entire enterprise.

3.1.22. The MACSYMA Consultant

Genesereth [40] described a help system for MACSYMA that would utilize a knowledge base to analyze user help requests in terms of faulty plans, goals, and so on. As such it is a clear precursor of the WIZARD system reported above. However, it is unclear whether the MACSYMA consultant was ever completed, as I have been unable to locate any later articles; the article cited described a nearly-completed, untested system.

3.1.23. BROWSE

BROWSE [11, 12] is an on-line system for providing help for UNIX. It is an extremely powerful static help system, integrating excellent menu and key word help. However, it is implemented as a stand-alone utility, which means that it has no context-dependent help and cannot preserve the user's screen context when it is invoked. The system has also been used as the front end for a system for viewing structured text [124]. In general, it looks like ZOG with keyword help attached, but designed to run on lower-cost display hardware than is required by ZOG.

The BROWSE system described above should not be confused with the identically named sytem of Palay and Fox [82], a system designed to facilitate "browsing through databases." The latter system integrates menu-based "browsing" with more traditional parameterized search of a database. In the database system, of course, parameterized search can reasonably be much more complex than a simple key word, but such complex search expressions are less clearly desirable in the narrower context of on-line help.

3.1.24. INTERLISP DWIM

Interlisp's DWIM facility [116] is not a help system in the sense studied in this thesis, but is worth noting as an example of the extreme toward which some help systems seem to aspire. DWIM stands for "Do What I Mean", and can be used to correct a large number of syntactic errors in Interlisp programs, either automatically or with user confirmation. DWIM is able to be successful in this effort largely because of the extremely regular and straightforward syntax of LISP. Applying its techniques in less regular domains will inevitably yield less spectacular results, but is certainly worth studying.

3.1.25. SPF

The IBM System Productivity Facility [59] incorporates a sophisticated help and tutorial system. The system is essentially menu-based, with a strong bias toward walking a "standard" path through the menu network, but it also provides limited facilities for keyword search through the network, and allows context-sensitive invocation of help (that is, the root menu frame varies with the context in which help is requested). The SPF help facility appears to be entirely integral; documentation and application program are inextricably bound together at the programming level.

3.1.26. Symbolics Sage and Document Examiner

Janet Walker's work on on-line documentation [119, 120] has produced interesting results in two areas relevant to this thesis. Her Sage system [119] provides a highly structured mechanism for constructing help databases, rather like Fenchel's SARA or the ACRONYM system developed for this thesis, but with a more rigorously constrained document structure. Whether such structure is good, of course, depends on whether the basic parts chosen for that structure are the right ones, which is a difficult assessment to make. However, Sage clearly provides more power to the documentation writer than the other systems, by virtue of its highly structured and integrated environment, possibly at the cost of making the documentation occasionally strained to fit into the prescribed format.

A more recent development is the Symbolics Document Examiner [120], a system utilizing the large high-resolution display of a Lisp Machine to provide help via key word and menu selection, with an unusual facility for using "bookmarks" to remember the user's previous history of especially successful help requests. The system can be accessed in a context-dependent manner, at least when looking at a LISP program. It appears to correspond closely to the state of the art in help systems, something like the ACRONYM system developed here, but designed to take advantage of modern workstation display technology. As of this writing, no description of the Document Examiner has been published.

3.1.27. CRAS

The on-line help system for CRAS (Cable Repair Administrative System) [43] at AT&T reflects a deliberate effort to integrate on-line help with on-line documentation design and more traditional user's manuals. Special facilities are provided both to reduce the need for paper documentation and to provide tools to help users keep their paper manuals up-to-date. The help system is highly integral to the CRAS application system, even utilizing CRAS file structures and software as part of the help database.

Despite the care given to complex information handling, CRAS's help system is extemely limited from the user's end. There is no context-dependent help, no menu system, and no keyword help worthy of the name. Instead, a few simple commands are used to extract help from the database, and the relevant application programs tell how to get that help as part of their error messages, e.g. "For more detail type: prtdoc cmd.rpt02". Thus, despite the complexity and effort involved in the database, CRAS's help offers very little in the way of help functionality to the user. If, however, this is the price which the designers paid in order to make the actual help texts extremely readable and useful, then the results of this thesis tend to validate that tradeoff. Unfortunately, the published description of the system gives no indication that text readability was emphasized, and no samples of the actual texts.

3.1.28. DOCUMENT

DOCUMENT [41] is a help system primarily geared to producing on- and off-line help from key word requests, where all key words were supplied explicitly by the document designers. The system is especially notable for its support of multiple interaction styles; there are three levels of expertise, and the verbosity of prompting and error messages varies with the expertise level. Expertise level may be specified explicitly, but the system will sometimes revise its estimation of a user's expertise based on that user's actual performance. DOCUMENT is also noteworthy for the care given to integrating the delivery of paper and on-line help in an extremely complex user environment.

3.1.29. Thumb

Thumb [85] is a system that weds on-line help to the kind of search mechanisms generally reserved for infrormation retrieval systems. Documentation is carefully encoded in such a way as to finely describe its structure, and is then accessed with complex search commands. The resulting language is extremely powerful but baroque; the paper on Thumb [85] cites an example in which "about rain *interacting* with night" and "about rain *occurring* with night" match *different* sets of texts for which "rain" and "night" are keywords. Although the primary mechanism is this kind of complex keyword search, menus are provided at points of ambiguity. Context-dependent help is apparently not provided. The complexity of the mechanisms suggests that such methods may remain more appropriate for general information retrieval than for the specific task of on-line help.

3.1.30. Transition Diagram-based Help

Feyock [36] describes a planned help system which would utilize a transition diagram representation of the user's state to generate help messages. In his discussion of the representation of the user's state, Feyock anticipates much of the command grammars that have been used subsequently, including in this thesis. He gives little attention to user input, which he correctly sees as being easily implemented in a number of ways with the single database. More surprisingly, he completely disregards the role of help texts, assuming instead that the representation of a state in the command grammar will easily yield readable help messages: "*Once such a* [regular] *expression has been generated, a simple recursive routine suffices to output the regular expression in a quite readable format.*" The results of this thesis, however, suggest that the construction of readable texts is the most important single part of help system design. At a minimum, it must be said that there is no evidence at all to support Feyock's claim that automatic generation of texts can, in our current state of knowledge, produce high-quality help messages.

3.1.31. ACTIVIST and PASSIVIST

Fischer, et al. [38] describe a knowledge-based help system that can act both actively (making helpful suggestions, in a manner similar to the WIZARD system described above) and passively, interpreting help requests given in natural language. They concentrate on the underlying knowledge representation supporting the help systems, rather than on the details of the user interface. As with WIZARD, there is no evidence presented to prove that the knowledge-based systems actually gain anything over the more traditional approaches, but instead the authors accept this as being simply obvious to anyone who considers it.

3.2. Perspectives from Other Domains

In thinking about help systems, it is useful to think not only about what has been done in on-line help for software, but also about analogous methods that have developed in other domains in response to the problem of informing the user about the tool at his disposal. A comprehensive survey of such areas is beyond the scope of this thesis, and indeed is probably impossible. However, I will discuss here a few of the related problems that have helped me in thinking about help systems. Heckel's enjoyable book [54] goes into much greater detail with a large number of analogies relevant to interface design.

3.2.1. Highway Navigation

In navigating an automobile, there are two primary help systems. The road map is a static help system, which provides a stable and (it is hoped) consistent picture of the task domain. This is most useful for general orientation and for high-level planning, as well as for problem-solving when errors occur. Road signs, the second help system, are by contrast highly context-sensitive, and are useful primarily for making low-level decisions in a hurry (e.g. "Do I turn here? Right or left?")

The driver of a car on the U.S. highways today faces a system as vast as nearly any software system yet devised, but despite well-known and oft-discussed exceptions, generally manages to find his way with little trouble. Why is this so?

To begin with, the driver usually benefits by well-defined context. It is exceedingly rare to see a road sign in California that points to Manhattan, which is as it should be given the percentage of drivers in the area whose ultimate destination is on the East coast. Indeed, it is unlikely that more than a small fraction of California drivers even have a map of New York in their cars. This is in sharp contrast to the typical computer user, whose "road maps" are the myriad paper manuals that line his walls, even if he has no intention of ever using most of the features they describe, and whose "road signs" are most often on-line help systems that point, with equal cheerfulness, to every feature on the system. (A road sign comparable to such help systems might read "Exit 22: Ho Chi Minh City, New Delhi, Spokane, Chicago, Forbes Ave.")

Of course, most computer systems do not begin with the clear usage patterns and locality of reference characteristic of a highway system, but this does not mean they cannot evolve in such a direction. Certainly it is reasonable to imagine a help system smart enough to say, "Hmm, this user doesn't do anything but word processing, so I really don't need to tell him about the debugger's symbol table even if he did type 'help symbol'." Moreover, it is also reasonable to design future systems to promote a clearer division of task categories, which will facilitate context-sensitive help.

Consideration of the imaginative generations of work that have designed the modern road map are also worthwhile. A road map is a static picture of a very large object. Nearly every such map, nowadays, comes in a very unusual form: an enormous page of paper cleverly folded to facilitate looking at smaller parts of it at a time. Perhaps computer manuals can be imaginatively reformatted to facilitate a large number of help situations. Some of the conventions used in maps -- most notably, insets showing details at different scales -- may also be relevant.

There is a large amount of literature on traffic signs and marking, some of which I surveyed in my research on help systems. While surveying it, I dismissed it as generally irrelevant, much to my disappointment, because it concentrated on such "mundane" factors as the proper height of lettering on the signs, the proper colors of signs, and so on. (See, for example, [70].) In retrospect, given the results of the experiments reported in this thesis, it seems that I might have properly regarded this emphasis as a warning that my preconceived notions about what is most important in a help system might have been entirely misguided.

It is also worth considering, with regard to the current vogue for iconic user interfaces, the system of international symbols that has come, in recent years, to decorate the world's roads, airports, and other public places. Nearly everyone can tell a story of coming face to face with an international symbol, designed to be meaningful even to savages from the most obscure corners of the globe, and being utterly baffled by its meaning. Many iconic interfaces coming onto the post-Macintosh market seem equally poorly chosen.

3.2.2. Driving a Car

For better or for worse, most people seem to learn to drive both their first car and any subsequent car without once opening the owner's manual. That they prefer to do so is a motivation for designing "self-evident" software. That they are able to do so is a motivation for studying the layout of an automobile's controls.

The basic controls of an automobile are, of course, relatively uninteresting and the same for nearly ever car. Steering wheels, brakes, and gas pedals are the apparently inevitable results of decades of tinkering with real people operating real machines. The few points of variation are relatively minor, enough so to be easily indicated by a small picture, as in the map that tells where the gear positions are on a manual transmission. Desirable though such a situation evidently is, it seems unlikely to be matched by software systems for a long time.

More interesting is the design of the peripheral controls -- such things as the lights, radio, windshield wipers, and so on. With less intrinsic reason to standardize, these features vary wildly from one car to the next. In most American cars, the controls are labeled in English, but increasingly cars are built for worldwide distribution, and are labeled with cryptic and often meaningless icons. How do most people deal with these peripheral controls? Trial and error. Each dial is turned, each switch flipped, each knob pulled, until the desired effect is achieved. Crucial to this process are two assumptions on the part of the user: none of the controls can produce a harmful effect, and the set of things he'll need to try is very small. These assumptions are virtually never true for software systems. Help systems can only sometimes help to make them true, and hence to facilitate explanations; more often, it is in the underlying application interface design that these assumptions are violated. That's one reason people use software manuals more than the owners' manuals for their cars.

3.2.3. Small Appliances

The average American today uses a wide variety of small mechanical devices that, seemingly at least, make his life a bit easier. While the quality of the instructions that accompany these devices varies substantially, some of the techniques used in the better ones seem applicable to help systems. Most notably, some food processors and similar appliances have a few pictures on them, carefully designed and strategically placed, that seem to entirely obviate the manual for most situations. This technique of *designing for the most common need* is virtually never applied in help systems, but would be obviously useful there. Most help systems require the same amount of effort to answer the simplest questions as the hardest and least common ones. (In fact, the ACRONYM system described later in this thesis shared that deficiency, and even made some of the simplest tasks *harder* to get help for than some of the genuinely difficult tasks. See Figure 7-5, on page 99 and the accompanying text.)

3.2.4. Human Experts

Of course, the best help systems for almost any applications seem to be human experts. Such experts offer at least five things other systems generally lack: spoken input, spoken output, natural language, immediate recognition and processing of feedback when explanations are inadequate, and empathy. The importance of any of these should should not be underrated.

Chapter 4
Relevant Previous Research

The literature describing human factors research on computer software is small but growing. The subset of that literature that is relevant to on-line help systems is extremely small. On-line help systems have been as neglected by human factors experimenters as they have been by programmers. For example, Mudge [77] wrote a dissertation entitled "Human Factors in the Design of a Computer-Aided Instruction System" and, despite being generally quite thorough, managed to avoid making even a single mention of on-line help.

Thus the amount that is actually known is not great, but at least a comprehensive survey is still possible. In this chapter, I will discuss the major experiments with results relevant to on-line help systems. Since details about actual help systems were collected in the previous chapter, this chapter will focus on more general work, reporting relevant experimental results, critical surveys, technical notes, and more general psychological and text-related perspectives.

4.1. Surveys

Only a few papers have surveyed the state of the art in on-line help systems. Sondheimer and Relles [113] present an excellent survey of the various methods by which help systems can be studied. However, they then concern themselves largely with issues of implementation, and make some unwarranted assumptions about what help systems *ought* to look like. Houghton [55] surveys various techniques available in help systems and related aspects of user interfaces, but does not go into any significant depth about any of them. Shneiderman [108] surveys primarily the relevant experimental results. Christensen [17] surveys actual existing help systems.

4.2. Experimental Results

The few experiments that have been reported with regard to on-line help present a mixed bag of results, generally discouraging. From the published literature, it would seem that on-line help holds little promise for actually helping real users. Most of this phenomenon, however, can probably be accounted for by the fact that the experiments were conducted using rather primitive help systems. This may be inevitable, given the fact that psychologists don't build complex software systems, while programmers who do build them don't generally run experiments testing them. The current thesis is designed in large part to address this phenomenon.

Magers [73] describes an experiment in which a fairly unsophisticated help system is modified in several simple and easy ways to yield a new system that is significantly more useful for novice users. This study demonstrates the importance of such non-technical "details" as text quality, command names, and non-technical orientation. Unfortunately, the study suffers from a major confounding factor: while it demonstrated that one of the two systems was better than another, it could not pinpoint which of the differences were responsible, and the systems differed in many small ways. In the absence of a complete theory behind the experimental design, the results are open to multiple interpretations.

O'Malley et al. [81] report that after extensive observations of the actual usage of the UNIX help system, they determined that it was used in three distinguishable types of situations: for quick reference, for task-specific help, and for full explanations of commands. The latter is what UNIX already provides, and the former is easily provided, as indeed it was in a related study [5]. However, the second type of help, which the authors call "task-specific help", requires help that presents an integrated perspective on several separate system functions, a kind of help not often found in working help systems.

The later Bannon and O'Malley study [5] reports on the evaluation of a quick-reference help facility designed to supplement the standard UNIX help facility. Their paper focuses on the difficulties of evaluating such systems, and discusses several possible approaches to such evaluation. Apparently without having actually run any controlled experiments on the system, they conclude that controlled experiments are *not* useful in evaluating the high-level design and utility of the system. In their own words, "A more controlled study would be appropriate for the purposes of debugging the specific display design, after we had determined the usefulness of this type of facility." [5, page 68] This conclusion seems to be contradicted by the results of this thesis, which demonstrate that even the most basic high-level decisions that are made in the design and informal testing of a user interface can be completely wrong when reached by simple subjective evaluations. (Bannon and O'Malley would not be likely to deny that such decisions are often wrong, but merely that controlled experiments can help get them right. This thesis is an argument that they can indeed help, but only with the enormous amount of effort that went into the experiments reported here.) Of course, Bannon and O'Malley relied on intermediate methods -- not wholly subjective, but not rigorously controlled experiments. In the context of this thesis, their methods -- primarily long-term monitoring of real usage patterns -- would have required as much effort as did the controlled studies, and the results at best could not have been any more reliable.

Mantei and Haskell [74] analyze in detail the experience of a first-time microcomputer user and find that 54% of that novice's problems were related to the system documentation rather than to the interface itself. The authors make some suggestions about the causes of this phenomenon in terms of the conceptual orientation of the texts, but the study is equally valid as an argument for much more sophisticated help systems. Primarily, their study demonstrates that much of novice users' problems with computers may be related to inadequate help and documentation.

Draper [27] extensively analyzes experts' command usage on UNIX, and concludes that the notion of "expertise" on a complex system such as UNIX is highly misleading. He finds that so-called "experts" may know virtually non-overlapping subsets of the entire system, and act much like novices outside of their domains of expertise. He suggests, in fact, that the only real measure of expertise may be a user's facility with the help system -- that is, his ability to use the available help to learn what he needs to know. This conclusion, if correct, might help explain the surprising results of the experiments on experts using help systems reported in Section 7.4 of this thesis.

Mack, Lewis, and Carroll [72] report on studies of novices learning to use word processors. They conclude that the help systems were totally ineffective for their users because the help was oriented to answering specific questions, whereas discerning the right question was often the crux of the subjects' problems. This reinforces the importances of examples, tutorials, and concept-based help, which were apparently not provided by the help system on the word processor in the study. (The authors described the help system as "state-of-the-art" without elaboration.)

Bott [10] reports extensively on how naive users learn a simple task on a computer. Perhaps the most interesting and relevant conclusion he reaches is that analogies in help information can be highly misleading, producing "blind spots" when the analogy is not quite exact. This concern is echoed in a later article by Halasz and Moran [48].

Lang, Auld, and Lang [69] study the goals and methods of users trying to accomplish various simple tasks, and conclude that documentation is most useful when it is extremely short and to the point. This suggests that on-line help access mechanisms can be of great value if they can quickly direct the user to a short piece of text without making him look through a large body of irrelevant information.

For the computer scientist contemplating experimental research, examples and guidelines for experimental design are invaluable. Aside from general texts on experimental design [13, 57, 123, 125], there are several texts that demonstrate successful experimental methodologies in user studies of software other than help systems. Card, Moran, and Newell's book [14], Shneiderman's book [107], and Reisner's survey [88] summarize the state of the art to a large degree. Enlightening examples of well-constructed specific studies include the Roberts and Moran experiments [8, 93, 94, 95], Robertson, et al. [96, 97, 98, 99], Dzida, et al. [32], Black and Moran [7], and Loo [70].

4.3. Relevant Results from Psychology and Document Design Research

Compared to the paucity of actual help system research, an enormous amount of work has been done on the psychology of human-computer interaction and on the design of technical documentation. Summarizing all of this work is beyond the scope of this thesis; indeed, it might require a thesis-sized work in itself. Here, however, I will mention several highly relevant work which either summarize results from those fields in a manner relevant to help system design, or otherwise provide valuable insights for the design of help systems. This section should by no means be considered to be exhaustive.

The problems of naive users learning to use on-line systems have been discussed in many articles. Kennedy [62, 63] has advocated a number of specific remedies for these problems, including some unspecific recommendations for on-line help ("A HELP key or command ... is essential to give confidence to a casual or naive user." [62, p. 319]) Other worthwhile work discussing the needs and motivations of individuals learning to use computer systems has been done by Anderson [2, 3], Bott [10], Moran [75, 76], Nicholson [78], Norcio [79], and Rosenberg [102].

The problem of document design has received an enormous amount of attention. Much of this work is summarized in a set of document design guidelines by Felker, et al. [33], which draws on a wide range of research in this area.

Several studies [45, 46, 47] have demonstrated that reading from a standard video display screen is significantly slower than reading from paper, although the differences seem to go away with larger screens and higher resolution.

As a global motivation of studies such as this one, the study by Sproull, et al. [114] is a compelling demonstration of how far computer scientists still have to go to make computer systems less intimidating to new users. In this study, college freshmen responding to a carefully constructed survey indicate that the fear, frustration, and sense of inadequacy generated by introductory computer courses dwarfs any other terrors the university can offer to its undergraduates.

In addition, studies done at IBM [23, 24, 25] have dramatized the importance of response time for productive user interfaces. These studies suggest that rapid response time is an ideal worthy of more than lip service, for a small increase in response time can, at least for certain kinds of tasks, lead to a much larger increase in total task execution time. These studies should serve as a red flag to all who believe that it is reasonable to trade off speed for power.

4.4. Technical Research on Help Systems

Finally, a small amount of useful work has been done with regard to the actual implementation of help systems. In a sense, it is surprising that any work at all in this area could be useful, since in the absence of any real understanding of what a help system ought to look like it is premature to describe how it should be implemented. Nonetheless, a few authors whose work has included good ideas on the design of help systems have also contributed information on their implementation.

Fenchel [34] built the first help system to maintain all of its help in a single coherent database, thus making it possible to consider providing several kinds of help in an integrated manner from such a single database. Unfortunately, his work focused so much on the implementation that he never did build more than a single access mechanism, but the implementation work was quite interesting, and is a clear ancestor of the prototype system built for this thesis.

Sondheimer and Relles applied the same knowledge that produced their survey article [90] to produce a specification for a unified approach to on-line help systems [113]. This latter paper is clearly preliminary to a new implementation effort, and describes a comprehensive framework for the implementation of such a system. The authors may actually assume too easily that the proper design of such a help system is understood -- at least, it is obvious that they think they understand it -- but given the nature of the design they propose, their discussion of implementation techniques is very useful. An earlier paper by Relles and Price [89] goes into great detail about the architecture of a help system they built, describing primarily how the programmer interface to a highly flexible system was designed.

A few of the more experimental help systems reported in the literature actually are reported more heavily for their implementation techniques than their interaction facilities. These include especially the knowledge-based help systems such as Finin's [37, 109], Wilensky's [122], Howe's [56], and Genesereth's [40]. These papers are of interest primarily to the builder of knowledge-based help systems; it should be noted that there is as yet no demonstration that the knowledge-based approach is either practical or useful, despite the promises of its practitioners.

Of course, there is also a fair amount of literature detailing implementation considerations for user interfaces other than help systems. While some of this may well be of use to the help system designer, it will not be surveyed here.

Part Two

The Method

Chapter 5
The ACRONYM Help System

This chapter describes the prototype help system implemented for evaluation in this thesis. Sections 5.1 and 5.2 discuss the motivation of its design and the choice of UNIX as the implementation domain. Section 5.3 presents a top-level view of how the system works. This is followed, in Section 5.4, by a more technical discussion of the system's implementation, which may be safely skipped by the reader unconcerned with such details. The chapter concludes with a discussion of the limitations of the prototype system in Section 5.5.

5.1. Motivation: Factors in the Design of ACRONYM

Research on help systems can proceed in any number of directions, as the survey in Chapter 3 demonstrates. At the present time, promising fields for research include the use of graphics in help systems, natural language help systems, and "intelligent" help systems or tutoring systems which apply real knowledge about the task domain to the generation of help.

This thesis, however, is not designed to address any of the ongoing research problems in those areas. Rather, it is concerned with a more practical set of questions: how should techniques that are already well-known and well-understood be used to create the best help system that is practical and reasonably cost-effective at the present time? Obviously there are some grey areas: current workstation technology makes windowing and simple icons easy, but programming these workstations for sophisticated graphics, though possible, is still extremely complex and time-consuming, and hence not cost-effective for many applications.

The concern here is to utilize known techniques to the maximum and to study their effects. The importance of this approach is amply demonstrated by the sorry state of help systems in the real world today. It would appear that the few programmers who have seen fit to devote real effort to building help systems have generally acted in ignorance of the good facets of help systems that have gone before. The help system to be constructed for this thesis, therefore, was specified to be one that *any* good programmer should be able to completely implement in a reasonable amount of time and that would run on ordinary hardware.[13] In this way, the system should serve as a model for future implementations, or at least as a new minimum standard to which all future help system designers might be held accountable.

Given this limitation -- that the implementation of the help system should not itself address any unsolved research issues -- certain types of help features are immediately ruled out. However, the vast bulk of the help spectrum discussed in Chapter 2 remains available as techniques for the implementation of the prototype help system. In particular, all that are really ruled out are help systems which use fancy graphical techniques, beyond simple windowing, intelligent help systems and tutors, and natural language systems (although the latter was simulated with extremely interesting results).

Other factors motivated the exclusion of a few other potentially useful help features. Since the system was to be studied only in relatively short-term experiments, it seemed useless to build into it facilities for long-term monitoring of individuals and for tailoring its behavior to individual user models and profiles.

Perhaps more importantly, the contents of the database were dictated to a large extent by the necessity for objective experimentation. Since the experimental methodology (to be described in subsequent chapters) was designed before the help system was actually built, it was important to ensure that I did not bias the help system's database in favor of those tasks that I knew would be a part of the experimental task set. Therefore I obtained the contents of the database from the best and most appropriate book I could find.[14] Although the information in that book was already well-structured for inclusion in a network-based help database, it did omit some kinds of texts that might be useful in a help system. In particular, it did not, for each command, include special sections on level of expertise required, category of command (file handling, word processing, etc.), analogies or

[13] The definition of "ordinary" may be somewhat strained here: while sophisticated graphics are not presumed, the system studied in the experiments featured a 60-line terminal with a mouse. However, a second version of the system ran on an ordinary 24 line terminal without a mouse. Differences between the two versions, and observations relevant to future implementors who are constrained to a smaller screen, are recounted in Chapter 8.3.

[14] Texts for the experimental help system were derived in large part from Mark Sobell's *A Practical Guide to the UNIX System*, Copyright (c) 1984 by Mark G. Sobell, published by the Benjamin/Cummings Publishing Company. The cooperation of the author and publisher is gratefully acknowledged.

metaphors[15], or customization. It also did not provide multiple levels of deepening help for people with differing expertise. Any of these might improve the system, and all of them are perfectly feasible; indeed, they are merely the data for the help mechanisms, and could be included with no change to the mechanisms themselves.

5.2. The Choice of the Implementation Domain

As has been stated, the task domain chosen for the prototype help system and experiments was the UNIX operating system. While UNIX's widespread use and notorious opacity for novices [80] might make this choice seem the obvious one to many, a few words about its selection are in order.

The three key factors in choosing UNIX as the implementation domain were a large local user community, an interface that was initially difficult to learn, and a software environment conducive to the quick construction of the prototype system.

The requirement of a large user community, which was designed to facilitate experiments testing the help system on expert users, effectively narrowed the choices available locally to four: UNIX, TOPS-10, VMS, and TOPS-20. (Two other systems available locally that were rejected primarily because of the size of their user community were the SPICE integrated programming workstation environment and the UNIX-based Andrew system under development at the Information Technology Center at Carnegie-Mellon.)

Of these four systems, UNIX was the clear choice for both of the remaining two reasons. The cryptic nature of the UNIX command interface is legendary, as its supportiveness for software engineering tasks [65, 64]. In addition, UNIX was chosen because of the availability of UNIX Emacs [44, 9], a powerful editor which is programmable in a lisp-like language. By building the entire system within Emacs, it was possible to completely avoid having to write any screen management or process management code, which undoubtedly saved many weeks of implementation time.

[15]It should be noted, however, that several authors have argued against using analogies or metaphors in computer documentation. [10, 48]

5.3. The Design of ACRONYM

ACRONYM[16] is the prototype help system designed for use in this thesis. As stated earlier in this chapter, it was intended not to break new ground in help system design but rather to consolidate and integrate existing techniques. As it happens, this consolidation and integration itself may be regarded as ground-breaking: experienced computer users testing ACRONYM have routinely remarked on its power and clarity of design.

ACRONYM represents help in a single database which can be accessed via a number of independent mechanisms. The system was designed to make it easy to turn various component mechanisms on and off to facilitate testing. However, the version described here is the full version with all facilities enabled.

5.3.1. The User Interface

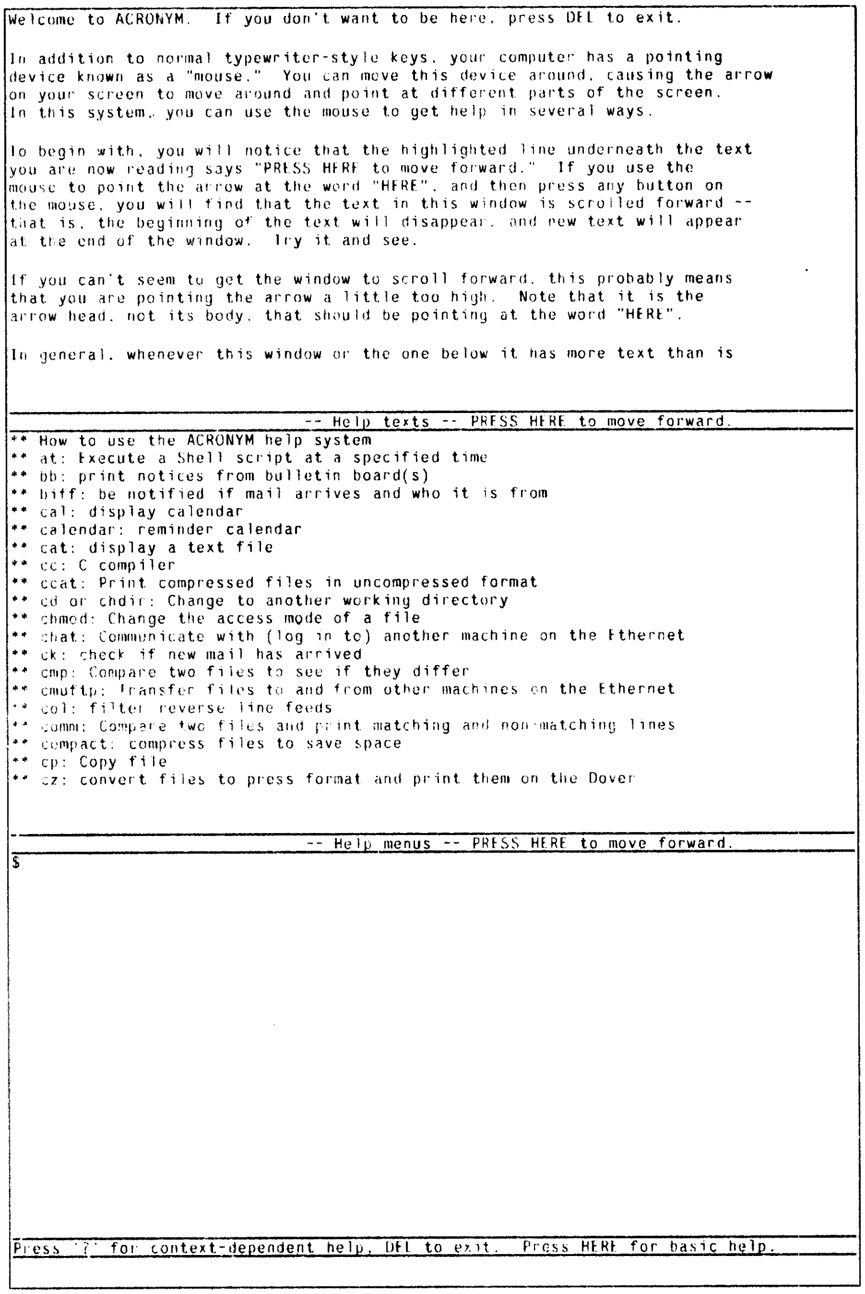
When ACRONYM runs, it divides the screen into three windows. (This is the primary motivation for using a larger-than-usual 60-line screen. Those versions of ACRONYM which use standard 24 inch screens end up with 7-line windows, a bit small for most people's taste.) The bottom window is the command window, where the user types commands and the computer prints its responses. The top window is the help text window, where the system displays the help texts that have been accessed by one of the help mechanisms. The center window is a help menu window, listing relevant topics on which further help is available. Figure 5-1 shows the ACRONYM screen when it first starts running. Appendix B shows a series of such screen pictures as it illustrates how ACRONYM looks in actual use.

All of the windows are independently scrollable, using mechanisms to be described below. However, the database has been structured so that in most situations, scrolling is not necessary; the size of the help text and the number of menu options is usually small enough to fit in the 19-line windows provided.

Help is available via four basic mechanisms. The first mechanism is *passive help*. This is the only computer-initiated help ACRONYM provides. Whenever the user types the SPACE key (the most common separator between components of a UNIX command) or the RETURN key (the terminator of UNIX commands), the system parses the partial command line and updates the help text and

[16]ACRONYM is not an acronym.

Figure 5-1: What ACRONYM's Screen Looked Like



```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is


                              -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** at: Execute a Shell script at a specified time
** bb: print notices from bulletin board(s)
** biff: be notified if mail arrives and who it is from
** cal: display calendar
** calendar: reminder calendar
** cat: display a text file
** cc: C compiler
** ccat: Print compressed files in uncompressed format
** cd or chdir: Change to another working directory
** chmod: Change the access mode of a file
** chat: Communicate with (log in to) another machine on the Ethernet
** ck: check if new mail has arrived
** cmp: Compare two files to see if they differ
** cmuftp: Transfer files to and from other machines on the Ethernet
** col: filter reverse line feeds
** comm: Compare two files and print matching and non-matching lines
** compact: compress files to save space
** cp: Copy file
** cz: convert files to press format and print them on the Dover


                              -- Help menus -- PRESS HERE to move forward.
$











Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

menu windows accordingly. Thus, if a user types "rm" and then presses SPACE, the help text window is updated with a short text describing the options and arguments for the rm command.[17] The menu window is updated to list such related help topics as "Examples of the rm command", "Options for the rm command", "What is a file?", and "rmdir: Delete a directory". Note that some of these menu items provide further details about the command in question, while some describe concepts of global importance (files, directories, path names, and so on), and still others point to the documentation on other, related commands.

The second method by which help can be obtained is also context-dependent; it is in fact just like the first except that it is human-initiated instead of computer-initiated. By typing a question mark ("?") at any time, the user can cause the system to parse the current command line and update the help in much the same manner as it does automatically when SPACE is typed. However, this mechanism is available even if the automatic updating is turned off, and can be used in the middle of a word to beneficial effect. For example, in UNIX a hyphen ("-") usually signals an option on a command line. Therefore, if an ACRONYM user types, for example, "rm -" and then types a question mark, ACRONYM will update its help window to contain a detailed description of the options for the rm command. This technique is also used for file name completion. For example, if the user types "rm mx" and then types a question mark, the menu window will be updated to include, as new menu choices, the list of all file names in the current directory starting with "mx".

The third method by which help may be obtained is key word requests. At any point, the user may type the word "help" followed by a key word. ACRONYM will then look up the key word and update the help and menu windows accordingly. For example, if the user types "help rm", the help text will be updated to the initial help for the rm command, with appropriate menu items. If the user types "help file", the text will discuss what a file is while the menu will list commands and concepts related to the notion of a "file". In the case of ambiguous requests, such as "help fil" which might reasonably refer to "file", "filter", or "profile", ACRONYM will dynamically generate a help menu that allows the user to choose between those ambiguous interpretations.

Finally, the fourth method by which help may be obtained is menu selection. As described above, the menu window at all times contains a list of topics for further help. In the mouse-based version of ACRONYM, users can simply point to one of these at any time and click any mouse key, and the help will be updated to explain the topic selected. (In the non-mouse, small screen version of ACRONYM, menu selection is done with function keys.)

---

[17] The rm command on UNIX is used to delete a file or files.

When help is updated by a mechanism that is not context-sensitive -- that is, by key word or menu help -- the new menu will include as its first item the option of returning to the previous help frame. This makes it relatively easy for users to recover after an ineffective help request. An arbitrary number of menu or key word help requests can all be backtracked, but the stack is reset every time context-sensitive help is obtained. (This seemed to make sense when it was designed, but observation of users suggests that resetting the stack causes unnecessary confusion. However, the backtracking feature was never used to backtrack long distances, so a better yet still efficient implementation might simply allow backtracking of the last 20 help frames, whatever they were.)

A few other features of the system are worth mentioning. Independently scrolled windows are implemented as follows: when the help text or help menu window contain more text than can be displayed, one or both of the phrases "Press HERE to scroll backward" and "Press HERE to scroll forward" appear on the line that divides the window from the one below it. Users can scroll the windows by simply pointing at the appropriate spot with the mouse. Scrolling is implemented with function keys on the non-mouse version (e.g. "Press f13 to scroll backward"). Help with the mechanics of the help system itself is also always available with a mouse selection; a spot at the bottom of the screen says "Press HERE for help with the ACRONYM Help System". This, too, is done with a function key in the mouseless version. An error message is printed if the user tries to select an inappropriate part of the screen with the mouse.

The use of ACRONYM is further illustrated by example in Appendix B. Reading that example will probably give the reader who has never used ACRONYM a clearer picture of how the system works. A brief videotape of ACRONYM in action is also available.

5.3.2. The ACRONYM Database

The most important thing about the ACRONYM database is that it *is* a database -- a database of knowledge about help systems, rather than a set of texts for use in specific circumstances. It was not designed to support a single help mechanism, but instead to structure the help data in such a way as to facilitate retrieval via several different mechanisms. This kind of database allows the maintainers to add new help mechanisms without having to write new texts.

ACRONYM's database is structured as a network in which the basic object is an indivisible chunk of help text. These texts are linked to each other by pointers, which are used both for the construction of menus and for the parsing of command lines. Pointers may be *syntactic*, in which case they are used for parsing, *semantic*, in which case they are used for menu construction, or both.

Purely semantic pointers can be specified forward or in reverse, to simplify fleshing out the database; duplicate pointers are automatically eliminated. Each text has a short name, used in the database implementation to specify pointers, and a long name, which is the one line of text displayed in menus.

As stated before, the ACRONYM database comes primarily from Sobell's excellent text on the UNIX system [112]. It was necessary to supplement these texts with additional texts to describe those commands which are not part of standard UNIX but are in common use at Carnegie-Mellon. (It was considered essential that the depth and breadth of the ACRONYM database should be approximately the same as the standard help system to which it was being compared.) These texts were written in a format and level that resembled Sobell's texts as much as possible. It was also necessary to supplement the Sobell texts with additional texts that explained concepts, such as "file" and "directory", which were not available in the right format in the Sobell book. Finally, of course, it was necessary to link these texts together with pointers; no such linking was provided by the Sobell book. (Such links would not make much sense in the context of a textbook.)

5.4. The Implementation of ACRONYM

This section describes some of the details of ACRONYM's implementation. It can be skipped without loss of continuity by the uninterested reader.

ACRONYM was implemented on a Digital Equipment Corporation VAX-11/750 running UNIX and Emacs. The terminal used was a Xerox Alto personal computer running a terminal emulation package (rchat) which simulated a 60 line intelligent terminal with a mouse.[18] Most of the code was written in Mock Lisp, the Emacs extension language, but the most computation-intensive portions were written as a C process communicating with Emacs via the CMU IPC [86]. This allowed the C program to parse the command line asynchronously, so that there were no user-level delays for parsing. (This also meant that occasionally the help update performed when the user pressed SPACE was not completed until the user had typed several more characters.) The system ran with no detectable response delays on the VAX when unloaded, and with a few seconds delay for context-sensitive help when the VAX was heavily loaded. However, for the experiments the system ran at high priority, and hence with very fast response.

[18] The terminal emulation package did not permit graphics or sophisticated control of the mouse. The mouse identified its own location only by its character row/column position, rather than the much finer resolution of which the hardware was capable.

The details of the implementation of ACRONYM and its data structure were dictated in large part by the decision to build the system within Emacs. Emacs offers several advantages in such a system, most importantly the built-in provision of sophisticated screen management, string handling, and process management facilities. However, its facilities for file access are rudimentary: either it reads in an entire file or nothing at all. Thus, the implementation either had to store the ACRONYM database in one or a few large files which would be read by Emacs in their entirety and then taken apart as needed, or it had to store the database in a very large number of small files, each containing the help text for a single node in the help network. This latter approach was used, as it held out the clear promise of faster performance in the Emacs environment.[19] A system operating independent of Emacs could easily achieve the same performance without such a proliferation of files simply by using pointers into files and random disk access.

Thus, ACRONYM stores almost everything as a separate file. What the documentation designer creates for a help node is two files: the first file is the help text, displayed by ACRONYM in its help text window, and stored in a file named by the node's unique name followed by the suffix ".help". The second file, suffixed ".hcom" is a program specifying the way the help node is linked to other nodes, written in a compiled language that could easily be improved on in future systems. The first line of such a program is the "tag line" used to identify the node in help menus. The remaining lines are each pairs of words specifying relations between nodes. The first word gives the *link mechanism*, while the second word is the name of the node being linked to.

The link mechanism is simply "@link" to specify a semantic (menu-only) link from the current node to the node named as the second word. The link mechanism "@key" is the opposite: it specifies a menu link from the named node to the current node. (The word "key" is intended to suggest the specification of a key word. For example, the .hcom file for "rm" includes "@key file" to indicate that "file" is a key word for rm, so that people looking at the help for "file" should see a menu item leading them to the "rm" command.) Other link mechanisms available are for syntactic links -- links that are used in the parsing of command lines for context-sensitive help. These mechanisms specify state transitions: if the parse is in a state corresponding to the current node, the syntactic links specify how the parse can advance to another node. The simplest such mechanism is a simple string, indicating that the transition can be made if the user types the string verbatim. Thus, for the top level (root) help node, the .hcom file includes the line "rm rm", indicating that if the user

[19]Emacs can't easily do random access to a file, and can hence extract pieces of a file only relatively slowly. Using independent files allowed each window's contents to be pre-written into a single file, so that a simple read-file operation was all that was ever needed. Hence the database had thousands of files.

types "rm" the parse can proceed to the help node "rm". Other syntactic link mechanisms include "@file", which is matched (and hence the parse proceeds) if the user types any existing file name, "@filewrite", which matches any potential (writable or existing) file name, "@directory", which matches any directory name, and "@user", which matches any user name. An early version of ACRONYM included a generalized regular expression parser for syntactic links, but this proved almost completely useless and was disabled in order to increase performance. It was replaced with "@opt", which matches anything starting with a hyphen ("-" starts most UNIX options), and "@any", which matches any single word, and is used as an escape mechanism in those few cases where the full regular expression parsing would have been genuinely useful. (Such a mechanism would probably be more useful for parsing help requests in a less rigid command language, but most UNIX commands are easily and quickly described using the simpler mechanisms.) All syntactic links are assumed to also be semantic links unless the name of the node being linked to is preceded with an "@" sign. In such a case, the link is purely syntactic, and no listing will appear in the relevant menu.

A documentation designer modifying the ACRONYM database therefore has to modify only the ".help" files and the ".hcom" files. Modifying the .help files is simply a matter of editing text, which is of course what documentation designers are paid for. Modifying the .hcom files is clumsier; in a system designed for real-world use, it would probably be worthwhile to invest some effort in the creation of better support tools to replace the cumbersome language used in ACRONYM. In particular, a useful tool would utilize a graphical display to create a map of the database, allowing the database designer to create links by drawing arrows and to specify their syntactic and semantic content by annotating the arrows.

When it runs, however, ACRONYM does not read the entire database in the format thus described; this takes too long. Rather, it reads in a compiled description of the database. Hence, a compiler must be run each time the database is changed. This compiler reads in all of the .hcom files, constructs an intermediate data representation, and stores this in a file that is actually read by ACRONYM. The compiler also, after reading in all of the .hcom files, creates for each node a ".hmen" file. This is the menu file that is inserted in the ACRONYM window. (ACRONYM does, however, add menu items to these files on certain occasions -- the .hmen files are simply the basic menus for each node.) Thus ACRONYM when running finds both its help texts and its menus in prewritten files, and needs only to use the compiled database to figure out which files to use. This is the primary reason why ACRONYM actually runs faster than the UNIX man command, which is a much less sophisticated help system but has to run everything through the nroff text processor before printing it on the screen. (This is obviously a rather worthless comparison; almost anything is faster than the man command. However, ACRONYM did actually run quite fast; a response time longer than a second was extremely rare.)

The hmen files can only be safely created after *all* of the .hcom files have been read in, in order to properly take account of reverse-specified links (links specified using @key). Compilation would be quicker and easier if no mechanism such as @key existed, so that each menu could be generated directly from the associated .hcom file (indeed, this would make it easy to implement separate compilation of each node, as opposed to ACRONYM's current all-or-nothing compilation.) However, this would make the documentation designer's problem harder -- he would have to edit more files more often. Since the implementor of ACRONYM was also the documentation designer, compilation speed was willingly sacrificed. Of course, with a good (graphical) interface for the documentation designer, this problem would be moot.

When ACRONYM runs, therefore, it has to read only the compiled representation of the database structure; this information includes the names of the nodes, the extended names of the nodes (single line descriptions for menus), and the pointer relationships. The format in which this information is stored is not especially efficient. No real users ever had to wait for ACRONYM to start up, as the system was already running when subjects showed up for experiments, so there was no great effort to make this part of the system work fast. With its full database (about half a megabyte of text and pointers) ACRONYM generally requires about a minute of real time to start up. Since the actual help texts are not stored in the compiled database, ACRONYM doesn't read in any such texts until they are genuinely needed.

ACRONYM was invoked from the UNIX shell by an alias that started up Emacs and executed a Mock Lisp "acronym" package. This program divided the screen into the appropriate windows, started a shell process in the command window, and started a hidden process (using the Emacs start-process mechanism) to run a C program also (confusingly) called acronym. This was the process that actually read in the ACRONYM database. When the database had been read in, the C program sent a message up to Emacs telling it to read in the root help and menu files, and the Emacs program then printed an appropriate startup message.

Menu selection was accomplished with no communication at all with the underlying C program. The menu files included (in the first 14 characters, and hidden from the user by Emacs trickery) the names of the files associated with each menu item. Thus the emacs code could look at the menu line that was selected and determine the name of the associated node in the help database. It would then read in the relevant ".help" and ".hmen" files.

Context-dependent help was executed by having a Mock Lisp program copy the current command line and send it to the C program. The C program would parse the line using its database of information, and send back to Emacs (via the IPC) the name of the new help node. Emacs would then update the help text and menu windows from the associated .help and .hmen files.

Key word help was accomplished similarly to context-dependent help; the C program would disambiguate the key word if possible, using the thesaurus-like information in ACRONYM's database, and send Emacs the name of the appropriate help node. All key words were implemented simply as individual nodes in the ACRONYM database; the system simulated a thesaurus inasmuch as a large number of synonyms for each concept were explicitly defined. A genuine thesaurus would be both more complete and less painful for the documentation designer.

In some cases of context-sensitive help, especially those involving file name completion, user name completion, or something similar, the C program had to write out a new version of the .hmen file "on the fly", to incorporate information about the current state of the world (available files, etc.). This was the only reason that the compiled database included the text lines (long names) for each help node.

When key word help or menu-selected help was provided, the mock lisp program that updated the display also inserted an appropriate menu selection item to allow the user to go back to the previous help node. The stack of such items was implemented as an Emacs buffer, so that the C program was never involved in this procedure.

In summary, the implementation of ACRONYM was quick and dirty. However, it is worth noting that these tricks sufficed to allow all of the mechanisms of ACRONYM to be built in about three weeks, from scratch. It was only the availability of the sophisticated support mechanisms of UNIX Emacs that made this possible, and hence that made it possible to include both the construction of the system and its experimental evaluation in a single thesis.

Obviously a real-world system, not embedded in Emacs, concerned with startup efficiency and with the interface to the documentation designer, and with some customization options to allow at least minimal user tailoring of the help system's behavior, would take considerably longer to build. However, it still seems reasonable to expect that such a system could be built by experienced programmers in not much more than one man-year.[20] Compared to the total cost of building a major

[20]This is the cost of building the mechanism, not of fleshing out the database. However, fleshing out the database is likely to take only slightly longer than writing conventional documentation for the same system.

software system, this seems quite reasonable, especially in the light of the frustration with which users have traditionally viewed their help systems.

Finally, for the masochistic reader, some annotated examples of the ACRONYM database are provided in Appendix C, which explain how the examples in Appendix B actually work.

5.5. Limitations of the ACRONYM Help System

Aside from the corners that were cut in its implementation, ACRONYM differs in a number of ways from the optimal system that could be built with the same basic approach.

Some of these are dictated by hardware and by the terminal emulation software used: ACRONYM would be obviously improved with support for highlighting and multiple fonts, careful use of graphics and animation, a better mouse mechanism (the rchat program's support for the mouse can only charitably be described as merely clumsy), and a higher quality display (the particular Xerox Alto used in the experiments probably had an excellent display about ten years before it was used in these experiments, but has not aged well).

Other ways in which ACRONYM fails to meet its potential are dictated by the methodological decision to use only externally-generated texts as much as possible: ACRONYM would be improved if it included tutorials structured for inclusion in its menu network, multiple levels of explanation, special information for experts, customization information, and references to external sources of help.

Finally, some of ACRONYM's other failings can only be attributed to implementation inadequacies, many of which only became obvious after the system had been in use for some time in the experiments: A future system like ACRONYM might include a provision for cycling through menu choices when successive requests for context-dependent help are made. (ACRONYM simply gave users the same thing, over and over again.) It might be very useful to treat any erroneous command as a key word help request, so that if a user types "directory" instead of "ls"[21] (a very common thing for a naive user to do), ACRONYM would supply help about "directory" that might quickly lead the user to "ls". File name completion could be more thoroughly integrated into the menu selection process, so that one could avoid typing a complete file name by selecting a menu item. The top-level menu, currently much too large, could be replaced with a series of submenus categorizing commands by major topics, as recommended by several authors [49, 81]. A real-world

[21]ls is the UNIX command for listing the files in some directory.

ACRONYM should probably include extensive opportunities for customization, and some provisions for choosing among alternative help presentations on the basis of the user's history of use of the system. All of the above are practical, fairly simple improvements that could be made to ACRONYM itself, with no undue programming effort.

Finally, an interesting research project might be to tie a natural language help system into the ACRONYM database. Although the experiments reported in Section 7.1 cast doubt on the general utility of natural language help systems, they may yet prove very useful to total novices who find even the mechanics of a help system such as ACRONYM difficult and intimidating.

Chapter 6
The Experimental Method

Evaluating the user interface of a computer system is difficult. It has been done rarely enough that no general theory exists, and only a few good examples such as Roberts' editor evaluation methodology [93, 94, 95, 8] are available to use as models. In addition, the wide variety of user interfaces that can be evaluated carries with it widely varying circumstances and problems for the evaluator, so that experimental methods do not transfer simply from one domain to another. In this chapter, I will describe not only the method used for the experiments in this thesis, but also the factors which shaped the design of the experiments, in the hope that an example of how such experiments are designed may prove useful to future designers of human factors experiments on software.

6.1. Factors Affecting the Experimental Design

A number of special problems influenced the design of the evaluation experiments. The most important of these factors were the need for a uniform implementation domain, the selection of experimental tasks, the stimulation of expert usage, variation among subjects, and the selection of interfaces to be tested.

6.1.1. Uniformity of Implementation Domain

Section 5.2 explained some of the reasons why UNIX was chosen as the implementation domain for the prototype help system. It did not mention, however, why it was considered necessary that all help systems to be tested operate in the same domain. This was necessary in order to prevent variability in the inherent difficulty of the task domain from confounding effects of the usefulness of the help system. If interface A with help system B is faster than interface C with help system D, it is simply not clear if the effect is due to a better help system or a better fundamental interface design. It is fundamental to design scientific experiments to minimize all variations except the one(s) being studied, so it is clearly preferable to study help systems against the backdrop of a single task domain.

6.1.2. Task Selection and Stimulation of Expert Usage

Help systems are, fundamentally, only used when subjects don't know how to execute the task they are trying to perform. This poses an interesting problem for experiments on experts: how can experts reliably be made to use a help system? Obviously, the selection of tasks for experts must be very carefully made in order to ensure that the tasks are sufficiently challenging or obscure that most of them will force most experts to seek some source of information other than their own memories.

For experiments on novices, of course, the problem is rather different; virtually any task will force a novice to seek help, but many tasks will prove hopeless even with the most sophisticated help. For that reason, tasks given to novices in the hope of observing their performance with a help system must be sufficiently simple to give them a reasonable chance of success even with the worst help system to be studied.

The successful selection of such tasks seems to require an iterative process. For these experiments, the selection began with a questionnaire that was given to about a dozen UNIX experts, asking two simple questions: First, what are the most basic commands that every novice learning UNIX needs to know? Second, what tasks can you imagine that you might have to perform on UNIX that would make you consult the help system?

From the answers to these questions, a first set of tasks for novices and a first set of tasks for experts were derived. These were each about thirty tasks, representing the thirty most common answers to each of the two questions. A pilot experiment was run, studying only two help conditions (the standard UNIX help system and a human tutor) using these tasks.

In the pilot experiments, a few of the tasks were obviously unsuccessful, either because they were too difficult for nearly all of the subjects or, for some of the expert tasks, because they were too simple and forced virtually no one to ask for help. The task lists were each pared down to 22 tasks for the final experiments.

By this process, an expert task list was obtained that was sufficiently challenging that the experts needed some kind of help to complete 87.5% of the tasks in the final experiments. A novice task list was obtained that contained tasks of sufficient simplicity that 95.2% of the novice tasks were completed successfuly within the time allotted.

However, this process is not without its flaws. Although the expert tasks seem fairly representative of how experts actually use the help system, they do consist in substantial part of tasks that no one would very often want to do; most of the interesting tasks are already known by too many experts. This seems a fairly insurmountable problem: the only way you can get experts to use a help system reliably is to ask them to do something they wouldn't ordinarily have much interest in doing.

Another flaw with the process by which the tasks were selected is that it could not take into account very recent research on UNIX command usage [27, 49]. The Draper study suggested that expertise in a domain such as UNIX is chimerical in nature: Draper would undoubtedly have predicted correctly that the only major difference between the experts and novices would be that the experts are already proficient with the standard help system and hence perform better with it. The influence of Draper's study might have been to focus the research more exclusively on novices at an earlier stage. In the study by Hanson, et al., real communities of users were observed and a list of the most important UNIX commands was obtained. Had these results been available when the task lists were selected, they would probably have influenced the selection of tasks for novices, as their data is surely more reliable than the survey of experts described above. However, as it turns out, given the restrictions imposed on the tasks for the current experiments (the tasks were individual commands, not involving editors or programming languages), the Hanson study only suggested three additional commands that possibly should have been included in the tasks for these experiments.

A complete summary of the expert and novice task sets is included in Table 6-1, along with the command(s) that were acceptable as solutions for each task. The exact texts of the task descriptions presented to the subjects appears as part of the experimental materials in Appendix A.

6.1.3. Subject Variation

A major focus of concern throughout these studies was variation in the previous experience and general computer aptitude of the subjects. Variation among subjects is a well-known problem in human factors experimentation, and has in fact proved very costly in many studies. In Roberts' pioneering studies of editors [93, 94, 95], this variation was sufficiently wide that the bulk of the editors were not significantly distinguishable for most of the tests. (This, remember, was in a *successful* experiment on human factors in software!) In the hope of obtaining more significant results, great attention was paid to the problem of subject variation, with modest success.

For the pilot experiments, five categories of expertise were defined and considered. Based on a preliminary questionnaire, included in the experimental materials in Appendix A, subjects were classified in one of the following categories:

Table 6-1: Summary of Tasks in the Experiments

Order	Intermediate task *(solution)*	Expert Task *(solution)*
T_1	Time of day *(date, uptime, whenis)*	Print on dover specifying font *(cz -f)*
T_2	Change password *(passwd)*	Sort in reverse order *(sort -r)*
T_3	List files *(ls)*	List files by time modified *(ls -t)*
T_4	View file *(cat, pr, more)*	Change protection as specified *(chmod o-r, chmod 640)*
T_5	Copy file *(cp)*	Delete file reversibly *(del)*
T_6	Rename file *(mv)*	List deleted files *(lsd)*
T_7	Print file on dover *(cz)*	Restore deleted file *(undel)*
T_8	Delete file reversibly *(del)*	Find i-number *(ls -i)*
T_9	List deleted files *(lsd)*	Set setuid bit *(chmod u+s, chmod 4xxx)*
T_{10}	Restore deleted file *(undel)*	Send to user logged in twice *(send -all, send user ttyxx)*
T_{11}	Direct message to another user *(send, write)*	List processes for all users *(ps a)*
T_{12}	Print calendar *(cal)*	Print file on dover with header *(cz -h "...")*
T_{13}	View file backwards *(rev)*	Sort, ignoring capitalization *(sort -f)*
T_{14}	Print working directory *(pwd)*	Cancel all pending mail requests *(mailq -retain)*
T_{15}	Make new directory *(mkdir)*	View only the printable strings in a binary file *(strings)*
T_{16}	Change directory *(cd, chdir)*	Execute remote command *(cmuftp r g - = "/date")*
T_{17}	Move file *(mv)*	Undelete old version of file *(undel -g)*
T_{18}	Delete empty directory *(rmdir, rm -r)*	Retrieve file from Onyx server *(ecp -u guest guest "[onyx]<AltoDocs>chat.tty" chat.tty)*
T_{19}	Delete full directory *(rm -r)*	Send to user on remote machine *(rsend user@host, send user@host)*
T_{20}	List current users *(u, users, finger, who, w)*	List processes on terminal ttypa *(ps tpa)*
T_{21}	Find string in file *(grep)*	List process with no terminal *(ps tp?)*
T_{22}	Send mail *(mail)*	List total space occupied by deleted files *(lsd -t)*

1. Total novices -- people who had never before used a computer.

2. UNIX novices -- people who had a certain minimum of computer experience, but had never used UNIX. For these experiments, to guarantee similarity of backgrounds in this category, all of the subjects used could perform all or most of the questionnaire tasks on the TOPS-20 operating systems, but none or nearly none of them on UNIX.

3. UNIX experts -- people who could perform all or nearly all of the questionnaire tasks on UNIX.

4. UNIX wizards -- people who were not only UNIX experts, but who also were so knowledgeable about UNIX in general that they knew virtually everything about the system. Obviously such knowledge could not be detected by the questionnaire, but became obvious during the experiments. UNIX wizards generally did not have much use for the help systems during the experiments, and were hence their data were disqualified after they were finished. Several such disqualifications were necessary in the pilot study, but none were necessary in the final experiments reported here.

5. Mixed expertise -- people who did not fit in any of the above categories. Generally, these were people who could perform some but not all of the tasks on the questionnaire, or who had never used either UNIX or TOPS-20.

In the pilot study, subjects from the first three categories were studied. However, the first category, total novices, proved intractable for the purposes of the experiments. Many subjects in this category were totally unable to perform even the simplest of tasks; one of them actually spent half an hour trying to understand the simple software for the typing test that preceded the real experimental tasks. More important, nearly all of the subjects had severe difficulty in using the baseline help system, the standard UNIX help system. Since the subjects simply could not get anything done with the standard help system, the entire methodology could not have worked well for them.

In the final experiments, which studied UNIX novices and UNIX experts, the results were discouraging in terms of these expertise classifications. In particular, the experts behaved in varied and unexpected ways, often belying their classification as "experts". These results are discussed in Section 7.4.

The problem of assessing subject expertise before the experiment is a difficult one, and one that simply was not solved by the "classification-by-questionnaire" method used in these experiments. The questionnaire was useful in that it did guarantee that the subjects had similar backgrounds, but it did not succeed in clearly differentiating between groups of users with similar performance abilities.

However, another aspect of the methodology proved very useful in reducing the effect of subject variation, given that such variation could not be avoided. Each subject in the experiments used two help systems, the standard CMU UNIX help system and one of the other help systems studied. (The exact experimental method is described in Section 6.3.) The performance of the individual on the standard system, compared to the large pool of data accumulated on that system, provided a rough measure of the user's basic competence against which to judge his performance on the non-standard system. (The actual technique used for most of the analysis was regression, as described in Section 7.1.) The net effect was that the primary measure of interest was not a subject's raw performance time with a given help system, but rather the difference between his performance on that help system and his performance on the standard "baseline" help system.

One negative consequence of the decision to use UNIX novices with TOPS-20 expertise is the possibility that these subjects were, to some extent, biased in favor of the context-sensitive component of the ACRONYM help system. However, ACRONYM is to TOPS-20's help what the modern jet is to the first airplanes, and early aviation pioneers would not like modern jets merely because of their prior prejudice in favor of being airborne. Nonetheless, the charge is serious enough to be guarded against in future studies of this kind. Fortunately, in this regard, the experimental results certainly give no indication that the deck was in any way stacked in ACRONYM's favor.

6.1.4. Selection of Help Interfaces

Another major problem in the experimental design was simply to choose the help systems that would be evaluated. This was difficult primarily because there were so many alternatives, and it was unclear which comparisons would yield the most significant differences. In the end, a two-stage approach was used.

First, the most basic unanswered questions were selected: Which is more important, help mechanisms or text quality? Can on-line help be implemented with no paper manual without adversely affecting its usefulness? How does the performance of on-line help systems compare to human tutoring? Appropriate help interfaces (as described in Section 6.3.4) were selected to try to answer these questions in particular. The results of these studies were analyzed before deciding what other help conditions to study in the last half of the experiment. This turned out to be an extremely useful approach because the results were so unexpected. Since it turned out (see Section 7.1) that text quality was so much more important than interface mechanisms, it became obvious that it was pointless to use the methodology to investigate small variations in help mechanisms. Had the complete selection of help interfaces to be studied been made without these preliminary results, it is likely that the experiments would have been twice as time-consuming without producing any more significant results.

6.2. General Evaluation Criteria for Help Systems

There are a number of measurable quantities that correspond to common intuitions about what constitutes a "good" help system. It is helpful to consider these general notions first, and then to try to translate them into the design of specific experiments.

One key question about any help system is simply, "How often does it tell you what you need to know?" That is, what fraction of the time will the system ultimately give a user the information he seeks? This will be referred to as the *hit ratio.*

Of special interest is the amount of time it takes to learn a given task using a particular help system. This time will be referred to as the *acquisition time for task t*, as measurements for different help systems are fairly comparable only when the task is the same for each.

Finally, there is also the question of *user satisfaction.* That is, it is not known whether or not the system that is most efficient is always most *preferred* by informed users. In fact, informal reports from the working world often seem to indicate that this is not the case, that workers often prefer interfaces that require slightly more time and work but are somehow more pleasant to use. Objective measurements of these preferences are possible through monitoring of the actual use of help systems when several alternatives are available, though it must be ensured that the users are fully aware of all the available alternatives. These measurements, however, would require long-term monitoring of a real-world system and its users, and are therefore beyond the scope of this thesis.

6.3. The Experimental Design

The considerations described above helped to shape the final experimental design, which will now at last be described.

Two parallel experiments were conducted for the two different expertise categories being studied, UNIX experts and UNIX novices with TOPS-20 experience. The two experiments were nearly identical in structure, differing only in the expertise of the subjects and, accordingly, the specific tasks to be presented to them. They are best conceived as two separate experiments, rather than as a single experiment in which expertise level is the second independent variable, because the differences in tasks for the different groups preclude direct comparison of performance times. However, a section of the task domain does overlap, so that between-experiment comparisons can be made for that portion of the data. (The overlapping portion consists of learning to use a set of simple UNIX utilities written by the experimenter and not commonly used by UNIX experts. This data is discussed in Section 7.4.)

6.3.1. Overall Design

Two experiments were conducted, each following the same general pattern. In each experiment, a group of subjects with similar backgrounds and experience in using computers were given a set of tasks to perform on UNIX. The tasks were selected so that each group of subjects were executing tasks they had never done before, and thus needed some source of information about how to accomplish the tasks. The independent variable in each experiment was the method by which this information was obtained. Each subject used the standard CMU UNIX help system during half the experiment, and used one of the other help methods during the other half.[22] These were balanced so

[22]The use of a *baseline* condition was designed to reduce the effects of subject variation, as described earlier in this chapter. The standard (man/key) help system was the obvious choice for the baseline: It is the only commonly available on-line help system for the task domain, and thus provides a practical base for measuring any improvements. In a practical sense, what is really interesting to know about UNIX's help is, "How can ***man*** and ***key*** be improved on?"

that an equal number of people used each help system for each half of the experiment. The primary dependent variable measured was the time it took to successfuly execute each task. The experiments were videotaped, and times were computed from the time stamp on the videotape.

In order to limit the time of the experiment and to insure that no subject got bogged down with a single task early in the experiment, a cap of ten minutes was placed on task execution time. The tasks were small enough that this was enough time for nearly all subjects on nearly all of the tasks.[23] When a subject failed to complete a task in ten minutes, the experimenter showed him the right solution (the right way to get the task done) and then allowed him to go on to the next task.

6.3.2. Experimental Setting

The experiment took place in the User Studies Laboratory of the Carnegie-Mellon University Computer Science Department. The subject sat at a Xerox Alto personal computer emulating a 60-line video terminal with a mouse, using the rchat program. With the exception of those subjects who were using a human tutor as their help system (see below), the subjects were alone in the room, with the experimenter monitoring them on a video screen in the next room. The video cameras were arranged to show a partial view of the subject, the experimental materials, and the subject's computer screen, in addition to the time stamp on the video tape. (The video tape included a time stamp in milliseconds, although the length of a video frame, about 17 milliseconds, defined the limit to the precision of the measurements.)

6.3.3. Pretest and Typing Test

Prior to the experiments, a pretest was used to determine the expertise level of the subjects. Each subject was classified in one of five ways: UNIX wizards, UNIX experts, UNIX novices who were TOPS-20 experts, total novices, and those of mixed expertise. These classifications were discussed in Section 6.1.3. Only the UNIX experts and UNIX novices who are TOPS-20 experts were studied in these experiments.

[23]In the pilot experiment, the cap was set at fifteen minutes. However, that experiment made it clear that this cap was unnecessarily high. Ten subjects were studied when the cap was fifteen minutes, and there was not a single instance where a subject completed a task in more than ten but less than fifteen minutes. In general, the final five minutes were simply a time of extreme frustration for subjects who had gotten hopelessly stuck.

If the subject fit into one of those two expertise categories, the next step[24] was an explanation of the experiment and a typing test. The typing test was included in case variation in typing speed turned out to contribute substantially to individual variation in task performance. It did not, and I would recommend to anyone who uses this methodology in the future that they simply omit the typing test to save themselves and their subjects a little time.

6.3.4. The Help Systems

As stated before, the independent variable was the help system. Each subject was presented with a written description of the use of the available help facilities, and was encouraged to practice using these facilities before the experimental tasks actually began. Each subject used the standard CMU UNIX help system as a "baseline" condition during half the experiment, and one of the other help conditions being studied during the other half. All of the help conditions studied will be described below, along with a list of other conditions that were considered for study but rejected for various reasons. The help systems actually studied are summarized in Table 6-2.

It should be noted that the assignment of subjects to help conditions was random, except that the final help condition (H_4) was studied later than the other three. Thus, the early subjects were randomly assigned to any condition except that one, and the last subjects were assigned to H_4.

Table 6-2: Help Systems Studied in the Experiments
(details in Section 6.3.4)

Help System	Description
H_0	Standard CMU UNIX help system (man/key)
H_1	Hybrid system: man/key with texts from ACRONYM
H_2	Fully implemented prototype system (ACRONYM)
H_3	Ever-present human tutor
H_4	Simulated natural language help system.

[24]Actually, the pretest was often administered prior to the date of the experiment, via campus, electronic, or US mail. Thus the "next step" described here was actually the first step in the experimental setting for most of the subjects.

6.3.4.1. The "Baseline" Help System: man and key

The "baseline" help system, which each subject used for either the first or second half of the experiment, is the standard help system used on the CMU UNIX systems. This system consists of two commands, ***man*** and ***key***. The man command is used to print the complete UNIX manual entry for a given command. The key command can be used to find out about unknown commands; users type "key file", and the system prints a single descriptive line for each manual entry that it finds for the key word "file".

This system embodies the key word help paradigm discussed in Chapter 2, but it does so less than ideally. First, the texts are of extraordinarily poor quality, by almost any standard. Second, the key word lookup is done in a very stupid manner: a key word matches a manual entry only if the word is exactly a substring of the first line of that manual entry. Third, the man command, for printing out manual entries, is very slow because it runs the entire manual entry through the nroff text processing utility before printing it out.

Subjects using the baseline system (H_0) were also supplied with a physical copy of the UNIX manual, so that they did not actually have to sit still and wait for the man command to perform. They were also supplied with a booklet called "UNIX for Beginners", which is generally supplied as part of the standard UNIX documentation for new users. Finally, they were supplied with a short instruction sheet explaining the use of the help system and manual, which is reproduced in the experimental materials in Appendix A.

6.3.4.2. The Hybrid System

The second help condition studied, condition H_1, was a hybrid system that consisted of the same mechanisms used in the standard system (H_0), but with better texts (derived from the ACRONYM help system described in Chapter 5, and hence derived in large part from Sobell's book [112]). The mechanisms were the same as the standard system at the user level, but performed better -- the man command was faster, and the key command, though somewhat slower, did a much more thorough search for key words. This hybrid system is thus basically just H_0 with better texts, but is probably better thought of as the standard system done right. Users of the hybrid system received exactly the same instruction sheet and supplementary materials that were given with the baseline system. Of course, they recieved a paper copy of the manual which contained the improved, non-standard texts.

6.3.4.3. The ACRONYM Help System

Help condition H_2 was the multi-featured prototype help system, ACRONYM, described in Chapter 5. Users of ACRONYM in these experiments were given a short set of instructions in its use, which is included in the experimental materials in Appendix A.

6.3.4.4. The Human Tutor

Help condition H_3 was a human tutor. Subjects with this help condition were allowed to ask any question of the tutor, but were not allowed to rely on the tutor's prior knowledge of what the problem was. Hence, all they had to do was to state the problem clearly in order to have the solution explained to them. Subjects using human tutors were given a short page of instructions, which are included in Appendix A.

6.3.4.5. Simulated Natural Language Help

The final help condition studied, condition H_4, was a simulated natural language help system. Subjects with this help condition were allowed to ask any question in natural language typed on their keyboard; the responses were determined by the experimenter in the next room, whose participation was not known to the subjects and came as a surprise to all of them when the deception was revealed after the experiment. Special support software allowed the experimenter to react quickly to each help request by sending the user a small portion of the ACRONYM database; thus the experimenter acted as an English-to-ACRONYM translator. The instructions given to subjects with this help condition are included in Appendix A.

6.3.5. Tasks

There were 22 tasks for novices and 22 for experts, divided evenly into two comparable sets. At the midpoint of the experiment, the subject were shown a different way of getting help, and were required to use that second method during the second half of the experiment. The task order was fixed throughout the experiment; the nature of the tasks themselves imposed at least a partial ordering, making it difficult to vary the task order in any reasonable way. A complete summary of the tasks for the experiments is given in Table 6-1, on page 74.

6.3.6. Posttest

The final part of the experiment was a posttest, in which subjects were asked to provide on paper some evidence of retention of what they had learned. A questionnaire asked each subject simply to provide the command used to execute certain tasks. This test measured the extent to which the information learned was immediately discarded. It was included to test the hypothesis that material which is learned more quickly is also more readily forgotten.

Part Three

The Results

Chapter 7
Results of the Experiments

7.1. Basic Comparison of the Help Systems

The basic purpose of the experiments was to compare the ease of learning to accomplish new tasks using a number of different help systems. Those help systems were described in Section 6.3.4, and are summarized for convenience in Table 6-2 on page 79. In this section, I will describe the results of the experiments.

The experiments yielded a complex set of multivariate data. It proved necessary to analyze the data in several different ways in order to provide a complete picture. In this section, the results will be broadly summarized, with a more detailed analysis following in subsequent sections.

In each of the sections that follow, the results of the experiments will be presented first for the novice experiments, and then for the expert experiments. The differences between these two sets of results will be discussed in Section 7.4.

7.1.1. The Basic Novice Results

Every one of the four help systems that were compared to the standard CMU UNIX help system yielded a significant improvement over that system, confirming the widespread impression that the standard UNIX help system is very poor. This is not terribly surprising; UNIX was chosen as the implementation domain in large part because its help system seemed to offer so much room for improvement.

Of greater interest is the relative degree of improvement each of the four systems offered. Also not surprisingly, the best system was the human tutor, which yielded significantly better results than the other three non-standard systems. Those systems were statistically indistinguishable. Perhaps most interesting, the difference between the standard UNIX system and the other systems was about the same as the difference between those systems and the human tutor. Thus, although none of the experimental systems performed as well as a human tutor, they did make up for about half of the difference between the standard UNIX help system and a human tutor on the tasks studied.

These highest-level results for novices are summarized in Table 7-1. This table's first column presents the average time per task for each of the five help systems. No data regarding variation due to subjects and tasks are figured into that column, which accounts for the extremely high levels of variance. The second column presents "normalized" averages for the same data. The normalization was done by the following procedure: First, a weighting factor was computed for each subject based on his performance using the baseline help system (H_0). The average total time it took all subjects who used that help system to complete those eleven tasks was divided by the time it took a particular subject to complete those tasks to obtain the weighting factor for that subject. All of the subject's scores for both help systems that he used were then multiplied by this weighting factor. That is, for each subject S_i, the weighting factor w_i was computed by the formula:

$$w_i = \frac{\sum_{j=1}^{11} \# A_j}{\sum_{j=1}^{11} \# T_{i,j}}$$

where A_j is the average time it took subjects using H_0 to execute task j, and $T_{i,j}$ is the actual time it took subject S_i to execute task j. Note also that j actually sometimes goes from 1 to 11, and sometimes goes from 12 to 22, dependening on whether the subject used H_0 in the first or second half of the experiment; the limits on the summation in the numerator follow those in the denominator in this regard.[25]

After each subject's scores were normalized by this procedure, overall averages were computed. The normalized averages thus compensate in large part for variation among subjects, and hence correspond more closely to the findings of the regression analysis to be reported in Section 7.2. However, they still have very high variances due to the variation among tasks. (Task execution times ranged from under ten seconds to ten minutes.) This variation is sufficiently large to preclude any significant results. Thus Table 7-1 does not present the data in the form in which significant results were obtained, but merely summarizes the overall trends; the regression analysis, to be explained below, yields the significance asserted to exist in the results.

[25]It would be more correct to say that the summations go from $\lambda(1,i)$ to $\lambda(12,i)$ in this regard, where $\lambda(x,y)$ is defined to be x for those S_i who used H_0 in the first half of the experiment, and x+11 for those who used it in the second half. However, this seemed overly formal and even harder to read.

Table 7-1: Summary of Novice Experiments
(Significantly different systems are separated by a blank line.
Significance is derived not from these measures but from regression.)

Code	Description	Average time per task (SD)	Normalized time per task (SD)
H_0	Standard (man/key)	167.0 (173.8)	167.0 (161.7)
H_4	Simulated Natural Language	101.4 (131.5)	123.0 (150.2)
H_1	Hybrid (man/key, ACRONYM texts)	136.9 (129.0)	115.4 (105.5)
H_2	Full ACRONYM	116.7 (122.2)	103.0 (104.8)
H_3	Human Tutor	45.6 (23.8)	60.1 (27.8)

7.1.2. The Basic Expert Results

The expert experiments are summarized in Table 7-2, in the same format as Table 7-1. By chance, the average time for experts using the standard help system was almost exactly the same as the average for novices using the standard help system, despite the fact that the task sets were almost entirely different. Aside from the fact that experts were not studied using the simulated natural language help system, there are two obvious differences between the novice and expert results. First, the ACRONYM system did not perform nearly as well for the experts. Second, human tutors were not nearly as useful for the experts. ACRONYM was not significantly distinguishable from the baseline system, while a human tutor was not significantly distinguishable from the hybrid system. The meaning of these somewhat surprising results is discussed in Section 7.4.

Table 7-2: Summary of Expert Experiments
(Significantly different systems are separated by a blank line.
Significance is derived not from these measures but from regression.)

Code	Description	Average Time per task (SD)	Normalized time per task (SD)
H_0	Standard (man/key)	168.7 (181.3)	168.7 (180.1)
H_2	Full ACRONYM	142.8 (120.1)	138.7 (124.4)
H_1	Hybrid (man/key, ACRONYM texts)	135.8 (156.5)	116.4 (134.3)
H_3	Human Tutor	79.4 (61.2)	103.2 (80.4)

7.2. Regression Analysis

The primary test for significant differences between the help systems was regression analysis, using the analysis program MINITAB [104]. The data were treated as a set of observations of four values. Each observation included the task, the subject, the help system, and the log of the time to complete the task. The first three were converted to indicator variables -- Boolean variables which indicated whether or not a given discrete value of the primarily value was observed. Thus, since there were 22 tasks, there were 22 indicator variables for the tasks. For each observation, exactly one of these 22 variables was 1 while the rest were 0. Similarly, there were 5 indicator variables for help systems, and 20 indicator variables for subjects. Regression analysis was then used to obtain a regression equation which indicated the effect of each of the indicator variables on the dependent variable, the time it took to complete the task.

The regression analysis yielded significant differences due to help system variation, as reported in the previous section. Not surprisingly, significant differences were also found due to task variation and subject variation. These results will be discussed in Sections 7.5 and 7.6. Table 7-3 summarizes the results of the regression analysis of the data from novice users. The constant in the equation, 4.29, indicates the predicted log time when all of the indicator variables in the equation are zero -- in this case, when the first subject executed the first task using the baseline (standard UNIX) help system. The remaining variables show the effects of different subjects, tasks, and help systems; the "Coefficient" column shows how much the predicted log time is changed, while the T ratio gives a measure of the significance of this change. Table 7-4 gives the results of the identical analysis of the expert experiments.

It is interesting to compare the results of the regression analysis to the cruder measures of help system variation that were summarized in Tables 7-1 and 7-2. Tables 7-5 and 7-6 demonstrate this comparison by listing, for each of the non-standard help systems, the actual decrease in average task time observed in the experiments, the decrease in normalized average task times, and the decrease in log time asserted by the regression analysis. (By "decrease", what is meant here is the average reduction in time when other factors are held constant and the help system changes from the baseline help system to some other help system.) It is reassuring to note that the regression results show basically the same thing that the normalized averages showed, only with a higher level of confidence.

It should be noted that the T ratios given in the regression tables in this chapter measure only the significance of the difference of a given condition from the "all zeroes" condition. In particular, the significance of differences between the other cases is not given explicitly in the tables. However, due

Table 7-3: Regression Analysis of Novice Data

$$y = 4.29 + \sum_{i=1}^{40} c_i x_i$$

Variable	Meaning	Coefficient	T Ratio (Coefficient/SD)
Y	Log of task execution time		
Constant	Task T_1, Subject S_1, Help H_0	4.2889	14.94
X_1	Subject S_2	0.1780	0.65
X_2	Subject S_3	0.9796	3.64
X_3	Subject S_4	-0.0556	-0.21
X_4	Subject S_5	0.1513	0.56
X_5	Subject S_6	-0.0480	-0.18
X_6	Subject S_7	0.2232	0.91
X_7	Subject S_8	0.1759	0.71
X_8	Subject S_9	0.1458	0.54
X_9	Subject S_{10}	1.2371	5.00
X_{10}	Subject S_{11}	-0.2200	-0.79
X_{11}	Subject S_{12}	0.2500	0.93
X_{12}	Subject S_{13}	-0.3570	-1.31
X_{13}	Subject S_{14}	-0.1048	-0.39
X_{14}	Subject S_{15}	0.8545	2.80
X_{15}	Subject S_{16}	0.0531	0.20
X_{16}	Help H_1 (Hybrid)	-0.2675	-1.53
X_{17}	Help H_2 (ACRONYM)	-0.3668	-2.15
X_{18}	Help H_3 (Tutor)	-0.6095	-3.52
X_{19}	Help H_4 (English)	-0.3010	-1.65
X_{20}	Task T_2	0.3025	1.01
X_{21}	Task T_3	0.1039	0.35
X_{22}	Task T_4	0.0736	0.25
X_{23}	Task T_5	-0.4312	-1.46
X_{24}	Task T_6	-0.0560	-0.19
X_{25}	Task T_7	0.4494	1.47
X_{26}	Task T_8	-0.2529	-0.85
X_{27}	Task T_9	-0.4343	-1.47
X_{28}	Task T_{10}	-1.2560	-4.24
X_{29}	Task T_{11}	0.6135	2.07
X_{30}	Task T_{12}	0.3786	1.26
X_{31}	Task T_{13}	0.5217	1.70
X_{32}	Task T_{14}	-0.0779	-0.25
X_{33}	Task T_{15}	-0.0768	-0.26
X_{34}	Task T_{16}	-0.7176	-2.39
X_{35}	Task T_{17}	0.7218	2.44
X_{36}	Task T_{18}	0.0158	0.05
X_{37}	Task T_{19}	0.3121	1.04
X_{38}	Task T_{20}	-0.6085	-1.76
X_{39}	Task T_{21}	1.0288	3.48
X_{40}	Task T_{22}	0.6411	2.17

Table 7-4: Regression Analysis of Expert Data

$$y = 4.29 + \sum_{i=1}^{35} c_i x_i$$

Variable	Meaning	Coefficient	T Ratio (Coefficient/SD)
Y	Log of task execution time		
Constant	Task T_1, Subject S_1, Help H_0	4.8348	17.21
X_1	Subject S_2	-0.3679	-1.57
X_2	Subject S_3	-0.5257	-2.55
X_3	Subject S_4	-0.1238	-0.53
X_4	Subject S_5	-0.4390	-2.16
X_5	Subject S_6	0.1475	0.61
X_6	Subject S_7	-0.2044	-1.01
X_7	Subject S_8	-0.8386	-3.48
X_8	Subject S_9	-0.5267	-2.22
X_9	Subject S_{10}	-0.4866	-2.09
X_{10}	Subject S_{11}	-0.3694	-1.49
X_{11}	Subject S_{12}	-1.0153	-4.25
X_{12}	Help H_1 (Hybrid)	-0.3030	-1.97
X_{13}	Help H_2 (ACRONYM)	-0.0031	-0.02
X_{14}	Help H_3 (Tutor)	-0.1906	-1.15
X_{15}	Task T_2	0.4794	1.54
X_{16}	Task T_3	-0.9443	-2.90
X_{17}	Task T_4	0.1818	0.54
X_{18}	Task T_5	-0.1622	-0.52
X_{19}	Task T_6	-1.4351	-4.69
X_{20}	Task T_7	-1.5989	-5.22
X_{21}	Task T_8	0.2018	0.63
X_{22}	Task T_9	1.1099	3.57
X_{23}	Task T_{10}	0.2173	0.68
X_{24}	Task T_{11}	0.0249	0.07
X_{25}	Task T_{12}	0.0189	0.06
X_{26}	Task T_{13}	-0.0823	-0.27
X_{27}	Task T_{14}	0.5323	1.74
X_{28}	Task T_{15}	1.4253	4.57
X_{29}	Task T_{16}	1.2113	3.89
X_{30}	Task T_{17}	0.2489	0.81
X_{31}	Task T_{18}	1.2841	4.20
X_{32}	Task T_{19}	0.0642	0.18
X_{33}	Task T_{20}	-0.1733	-0.54
X_{34}	Task T_{21}	0.0156	0.05
X_{35}	Task T_{22}	-0.2482	-0.81

to the general regularity of the data, the major differences suggested by large differences in T ratios *are* borne out by a closer analysis of the data (regression with one of the conditions in question as the "all zeroes" case). This is discussed in more detail in Appendix E.

Table 7-5: Comparison of Three Measures of Novice Variation

Help System	H_1 Hybrid	H_2 ACRONYM	H_3 Tutor	H_4 English
Average improvement in raw task time (seconds)	-30.1	-50.3	-121.4	-65.6
Average improvement in normalized time (seconds)	-51.6	-64.0	-106.9	-44.0
Improvement indicated by regression (log seconds)	-.27	-.37	-.61	-.30

Table 7-6: Comparison of Three Measures of Expert Variation

Help System	H_1 Hybrid	H_2 ACRONYM	H_3 Tutor
Average improvement in raw task time (seconds)	-32.9	-25.9	-89.3
Average improvement in normalized time (seconds)	-52.3	-30.0	-65.5
Improvement indicated by regression (log seconds)	-.30	-.00	-.19

7.3. The Importance of Text Quality

Possibly the most interesting and surprising of the results of this entire thesis is the performance of the hybrid help system (H_1, the system using the man/key mechanism to present texts from ACRONYM). This system is completely identical to the baseline system in its mechanisms. It is improved only in two ways: the texts are better and the key word indexing is more complete. These simple, non-technical changes produced an enormous and unexpected improvement in the performance of the users. In fact, this hybrid system performed as well as the full ACRONYM system; the two were not significantly distinguishable.

This lack of significance was certainly not due to any lack of effort to find it. In fact, it was so surprising that a supplementary experiment was performed. In that experiment, the subjects each used both the hybrid system and the full ACRONYM system, instead of just one of those two and the standard system. It was hoped that this more direct comparison would establish a significant difference, as the earlier data allowed the possibility that a small difference was being dwarfed by the effects of subject variation. However, no such difference was found; the supplementary experiment merely reinforced the lack of significant difference between ACRONYM and the hybrid system.

The first conclusion that comes to mind from this result is that help access methods matter little, if at all, and that designers of future help systems should make do with simple mechanisms, devoting all of their efforts to clear and well-indexed texts. This conclusion, however, is somewhat premature. An alternative hypothesis is that each of the two systems have certain advantages which balance each other roughly equally. This hypothesis is far more likely, given the results of previous research which suggest the superior readability and comprehensibility of printed text when compared to on-line texts [73]. The ACRONYM system is totally on-line, and is geared to producing short pieces of texts in response to various forms of help requests, which makes it difficult to conceive of a version of ACRONYM that would make good use of a printed manual. Thus, it seems likely that the superior readability of the hard copy (printed texts) in the hybrid system (compared to the screen used in ACRONYM) compensates for some superiority in the access mechanisms of ACRONYM. (In fact, there must be some such compensation going on, because ACRONYM is the first reported help system that performs as well *without* a printed manual as a simple help system that does include such a manual. This is itself extremely important for those concerned with issues of documentation currency; if it is in fact reasonable, with a system like ACRONYM, to throw away paper manuals entirely, it will be much easier to ensure that all users will have up-to-date documentation at their disposal.) Thus it is safe to conclude from the performance of the hybrid system that text quality is one of the most important factors in help systems,[26] but it is *not* safe to conclude that help access mechanisms are entirely irrelevant. The role of the improved key word indexing in the hybrid system should also not be neglected, although this was probably not as important as the variation in text quality.

An interesting problem, in large part beyond the scope of this thesis, is to try to determine precisely what makes the texts in the hybrid system so much better than the standard UNIX manual texts. Although no comprehensive, detailed study was made, a few informal tests did yield some hypotheses about the major source of the difference.

Three people who study documentation professionally --one professional writer of technical documentation, and two researchers in the area of document design --were shown samples of the two

[26]Indeed, these results clearly speak very well for the book from which the ACRONYM texts were taken [112].

sets of texts and asked to compare them. All three chose the texts from the hybrid system as the better texts without hesitation, suggesting at least that the intuition of the professional can be trusted to choose the texts that are objectively most useful. However, they offered varying opinions about *why* this text was better. One suggested that a major factor was the instructional orientation of the text; whereas the standard texts spoke in largely passive voice, or at least in third person, the hybrid texts were largely second person and imperative. Another of the evaluators suggested that the most important factor was the organization of the text into short, coherent subsections, with meaningful and highly-visible section headings. The third evaluator suggested that the texts differed simply in their readability --that is, in the complexity of the words and sentences used in the explanations. Of course these explanations are by no means mutually exclusive.

One hypothesis that was easily testable was that readability -- as defined by standard metrics of text readability -- was a major factor in the relative usefulness of the two sets of texts. The UNIX system includes a utility program called style [15, 16] which evaluates texts according to four standard scores of readability. The two sets of texts being discussed here were each subjected to this evaluation, with the results shown in Table 7-7. Despite the enormous differences between the texts, as shown both by the experiments and by the immediate impressions of everyone who has looked at them, none of the four metrics yielded any significant differences between the two sets of texts. This will come to no surprise to the many who have argued against the utility of such readability scores [28].

Table 7-7: Readability Analyses of the Two UNIX Manuals

Metric	Sobell/ACRONYM texts	Standard manual texts average (SD)	Average Difference (SD) texts average (SD)
Kincaid	9.9 (4.1)	9.9 (4.5)	-0.0 (2.8)
auto	9.9 (4.8)	9.4 (4.7)	-0.5 (3.2)
Coleman-Liau	9.4 (4.3)	9.3 (4.3)	-0.1 (1.8)
Flesch	10.2 (4.1)	10.3 (4.1)	0.1 (2.6)

Because no final conclusions could be reached about the exact causes of the performance improvement obtained with the better texts, it seems likely that readers will want to compare samples of these texts themselves. Such samples are included as Appendix D. The interested reader can compare the excerpts presented there from the two versions of the manual and reach his own conclusions regarding the factors that account for the great difference in their usefulness.

7.4. User Expertise and Help System Design

The hypotheses listed in Section 1.3, on page 10, demonstrate a strong belief, when the experiment began, in the idea that novice and expert users would have strongly differing needs and uses with regard to help systems. This expectation, based on simple intuition, has since been rendered suspect by Draper [27]. However, separate experiments were conducted on these two categories of users in order to observe any differences.

The expert experiments paralleled the novice experiment, except that a simulated natural language system was not studied. (Such as system is generally alleged to be most useful for novices anyway.) The differences between the two groups were indeed striking, but not in a way that was expected prior to the experiment. Those results were summarized in the preceding sections, and show two major differences between experts and novices: The experts fared much more poorly with ACRONYM and with a tutor than did the novices.

The poor performance of ACRONYM for experts was puzzling at first, but at least one plausible explanation (besides the obvious but unlikely explanation that ACRONYM is simply a bad system for experts although a good one for novices) can be advanced. Draper [27], in his studies of the nature of UNIX expertise, suggests that the only clear criterion which differentiates experts and novices on a complex system such as UNIX is their relative ability to find the information they do not already have. That is, the best definition of expertise in a system may be the ability to use that system's help facilities, both on and off line. If this is the case, then the results of this study make perfect sense: by improving the texts but keeping the mechanisms the same, the hybrid system allowed the experts to utilize their expertise with the help system while gaining access to better help texts. ACRONYM, on the other hand, made up for the fact that it provided better help texts by making the experts learn entirely new help mechanisms. An attempt was made to test this hypothesis by studying UNIX experts who used ACRONYM over a longer period of time, but this proved inconclusive.[27]

It should be noted that the novice and expert tasks did include three overlapping tasks; these were tasks that involved simple commands, but the commands were invented by the experimenter and

[27] Approximately a dozen UNIX users used a modified version of ACRONYM as their primary help system for a month. At the end of that month, however, it became clear that the average user had only needed help a very few times, and in most cases had needed help with regard to system calls, subroutine libraries, or other aspects of UNIX not covered by the ACRONYM database. Thus the experts never had a chance to become as familiar with ACRONYM as with the standard system. To be successful, it seems, an experiment like this would have to monitor novices who start with ACRONYM and keep using it until they become experts. Such a study is beyond the scope of this thesis.

hence were unknown to the experts. (Specifically, the commands were ***del***, ***lsd***, and ***undel***, which reversibly delete files, list deleted files, and restore deleted files, respectively.) However, the amount of data provided by these three overlapping tasks was inadequate to shed any additional light on the differences between experts and novice performance.

7.5. Variation due to Tasks

As the regression analysis (Tables 7-3 and 7-4) shows, there was a significant amount of variation due to tasks. This is not surprising; some tasks are inherently harder than others, and thus take longer regardless of the help system used. However, the regression analysis merely tells us which tasks were hardest; it does not shed any light on the question of whether harder or easier tasks fared particularly better with one help system or another.

In fact, the results become considerably clearer when the difficulty of the tasks is taken into account. For example, Figures 7-1 and 7-3 plot the average time taken on a given task using the baseline help system against the average time taken using each of the other help systems. These graphs show quite clearly that the differences detected by the study are more meaningful for the longer tasks.

This might suggest to the alert reader that, by omitting the data from the shorter tasks, a formal analysis might yield statistically significant differences between help systems that are grouped together as not distinguishable in Tables 7-3 and 7-4. This indeed is a reasonable supposition, but not one that proved to be correct when the analysis was performed. The significance levels did increase as the cutoff for task times was increased, but the amount of data became inadequate for the regression analysis before a level was reached at which further significant results could be detected. ACRONYM seemed to benefit the most by eliminating the shortest tasks, possibly because its more complicated mechanisms introduce a relatively large constant factor into each task execution, regardless of its length. The effect of omitting data for those tasks with average baseline times less than 2.5 minutes is shown in Figures 7-2 and 7-4. The relative performance of ACRONYM and the hybrid system in the expert data is most interesting in this regard. Clearly ACRONYM fared much better for the more difficult tasks than for the shorter ones.

It is also interesting to consider the possibility that different help systems may have been the most useful on different tasks. In Figures 7-5 and 7-6, the average time on each task is plotted for each of the different help systems. In these graphs, the x axis simply represents 22 points, one for each of the tasks. The tasks here are ordered by decreasing average task time with the baseline help system, in

Figure 7-1: Difficulty of Novice Tasks and Help System Variation

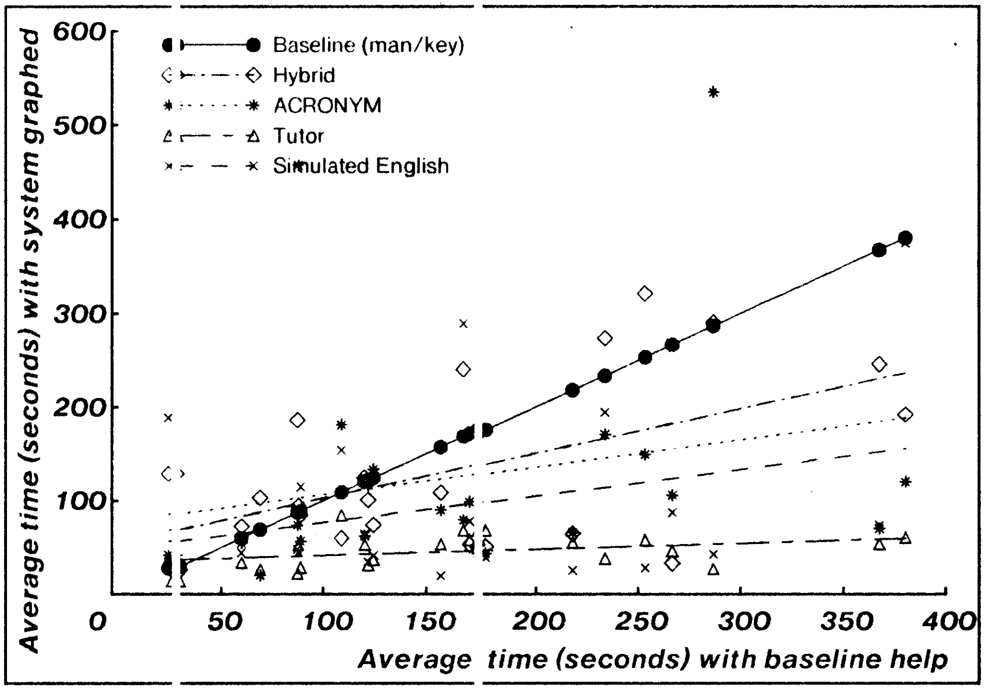


Figure 7-2: Help System Variation on the Hardest Novice Tasks

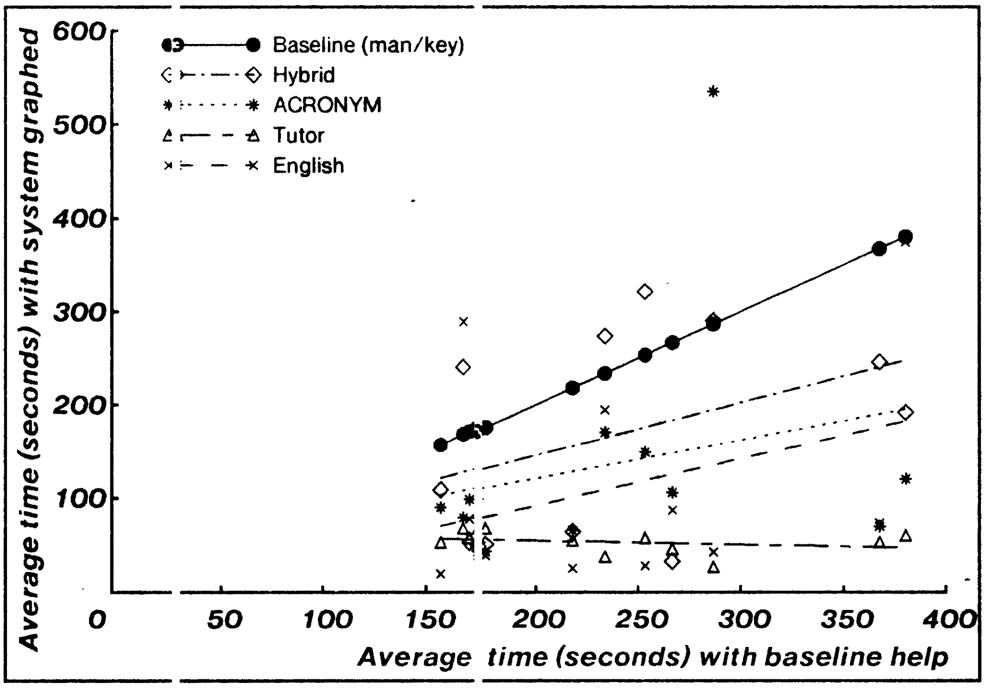


Figure 7-3: Difficulty of Expert Tasks and Help System Variation

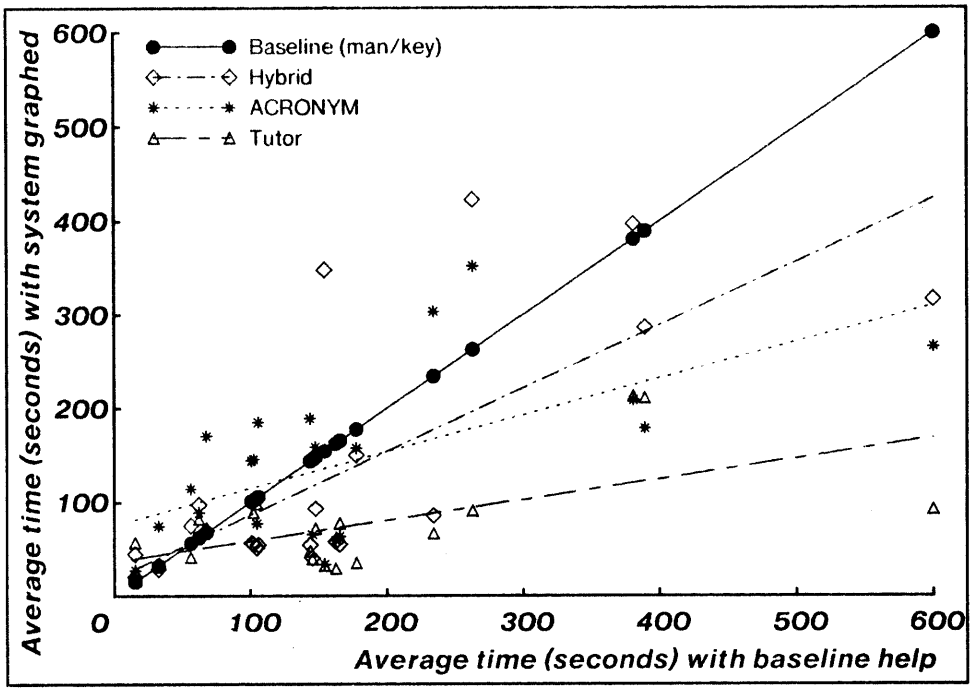
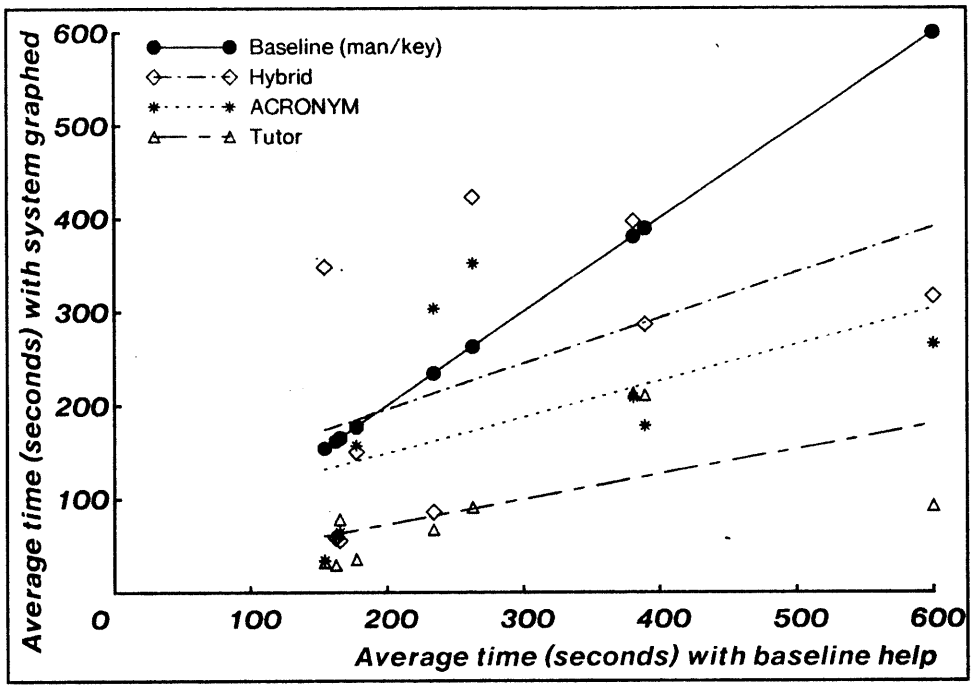


Figure 7-4: Help System Variation on the Hardest Expert Tasks



order to make the graphs more readable. The tasks are listed in that order in Tables 7-8 and 7-9, so that the anomalies in the figures may be analyzed easily. Figure 7-8 suggests, for example, that ACRONYM could be fine-tuned by improving its help for the "rev" (O_2) and "ls" (O_{16}) commands.

Obviously, it is possible to interpret these results in several ways. By isolating the tasks on which ACRONYM performed most poorly, it is clearly possible to find several particular ways in which ACRONYM and (more certainly) its database can be improved. Although this would be an obvious next step in the development of a real-world system, its usefulness in this methodology is somewhat suspect. In particular, if further experiments were conducted to verify the improvements, then they would be susceptible to charges of having tailored the system to suit the tasks being tested. On the other hand, if the data from the tasks on which ACRONYM fared most poorly were simply omitted, this would be committing the classic experimental error of only considering the data that matched the hypotheses. The best that can be said is that the results clearly demonstrate ACRONYM's usefulness on certain tasks, while calling into question its usefulness for a few other tasks. Nonetheless, there is no reason to assume that the particular improvements this analysis suggests would not significantly increase ACRONYM's overall performance in the experiments.

The varying difficulties of the experimental tasks are a primary source of variation in the experiment. Observation of the *distributions* of average task times for each help system can yield additional insights into the way in which varying the help system affects the task execution time. This is depicted for the novice data in Figure 7-7 and for the expert data in Figure 7-8. In each of these figures, the first graph shows the average time to execute each task for each help system, ordered in decreasing time within each help system. Thus, the various values at the same point on the X-axis are not comparable in this graph, which should be viewed only for the distribution of average task times. The remainder of the figures shows the distribution of the average task times as a separate histogram for each help system, to further clarify the diferent distributions. Figure 7-7 suggests again, for example, that while ACRONYM and the hybrid system were not distinguishable overall, this may be due to a few glaring imperfections of ACRONYM with regard to certain tasks, and that the fundamental ACRONYM paradigm may well be better.

Figure 7-5: Relative Performance of the Help Systems on Each Novice Task

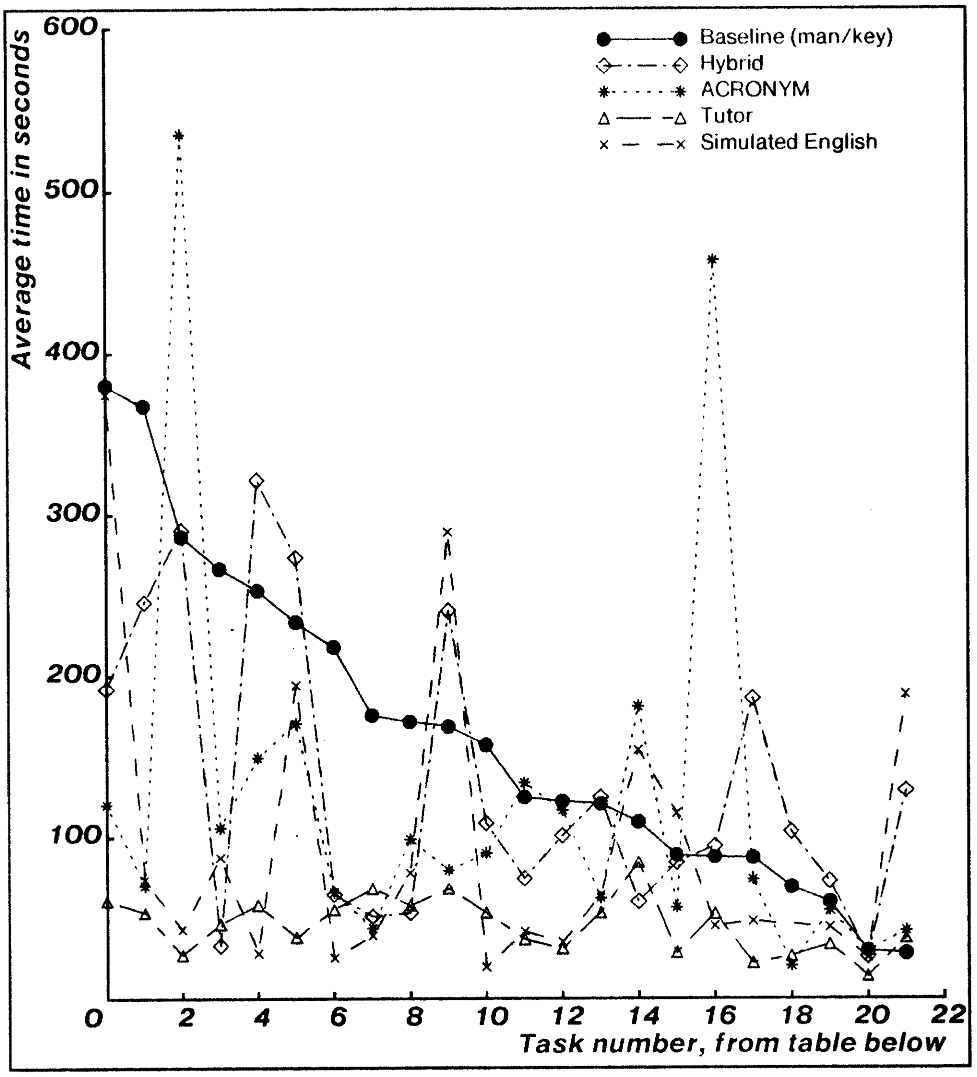


Table 7-8: List of Novice Tasks, Ordered as in Figure 7-5

O_0	Find string in file *(grep)*	O_{11}	List deleted files *(lsd)*
O_1	Move file *(mv)*	O_{12}	Time of day *(date, uptime, whenis)*
O_2	View file backwards *(rev)*	O_{13}	Rename file *(mv)*
O_3	Print file on dover *(cz)*	O_{14}	Change password *(passwd)*
O_4	Delete full directory *(rm -r)*	O_{15}	Delete file reversibly *(del)*
O_5	Message to another user *(send, write)*	O_{16}	List files *(ls)*
O_6	View file *(cat, pr, more)*	O_{17}	Print working directory *(pwd)*
O_7	Delete empty directory *(rmdir, rm -r)*	O_{18}	List current users *(u, users, w, who, finger)*
O_8	Print calendar *(cal)*	O_{19}	Copy file *(cp)*
O_9	Send mail *(mail)*	O_{20}	Restore deleted files *(undel)*
O_{10}	Make new directory *(mkdir)*	O_{21}	Change directory *(cd, chdir)*

Figure 7-6: Relative Performance of the Help Systems on Each Expert Task

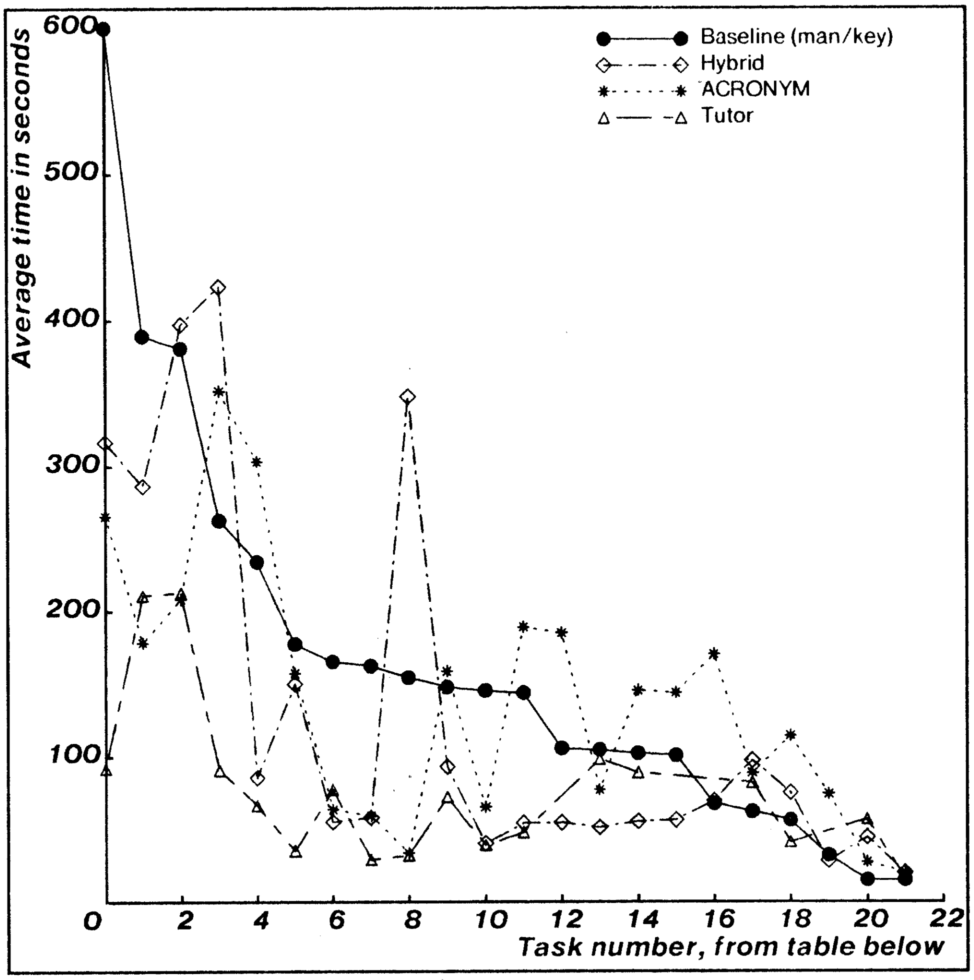


Table 7-9: List of Expert Tasks, Ordered as in Figure 7-6

O_0	View only printable strings *(strings)*	O_{11}	List processes with no terminal *(ps tp?)*
O_1	Execute remote command *(cmuftp r g - = "/date")*	O_{12}	Change file protection *(chmod o-r, chmod 640)*
O_2	Retrieve file from Onyx server· *(ecp -u guest guest "[onyx]<AltoDocs>chat.tty" chat.tty)*	O_{13}	Send to user on remote machine *(rsend user@host, send user @host)*
O_3	Set setuid bit *(chmod u+ s, chmod 4xxx)*	O_{14}	Print file on dover with header *(cz -h "...")*
O_4	Cancel pending mail requests *(mailq -retain)*	O_{15}	List processes for all users *(ps a)*
O_5	Sort in reverse order *(sort -r)*	O_{16}	Print on dover specifying font *(cz -f)*
O_6	Undelete old version of file *(undel -g)*	O_{17}	Delete file reversibly *(del)*
O_7	Sort ignoring capitalization *(sort -f)*	O_{18}	List processes on terminal ttypa *(ps tpa)*
O_8	Find i number *(ls -i)*	O_{19}	List files by time modified *(ls -t)*
O_9	Send to user logged in twice *(send -all, send user ttyxx)*	O_{20}	List deleted files *(lsd)*
O_{10}	List space occupied by deleted files *(lsd -t)*	O_{21}	Restore deleted file *(undel)*

7.6. Variation due to Subjects

Variation between subjects is a common problem in studies of this kind. The design of the experiments reported here was carefully structured to minimize the impact of such variation; this is the reason that each subject used both a variable help system *and* a baseline help system. While this technique proved effective enough to allow real differences to be detected, it is worth considering in some detail how much the actual subjects varied in their performance.

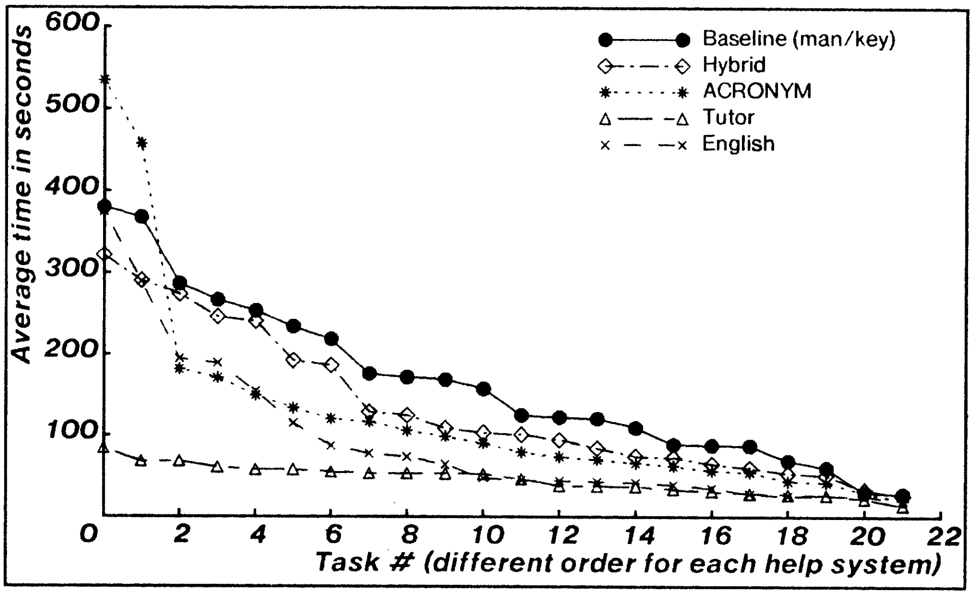
A regression analysis identical to the one reported above, but omitting all data on the correspondence between users of the variant help systems and the baseline help system, was performed. In Tables 7-10 and 7-11, the results of that analysis are summarized. Here, the same subject using two different help systems was treated as two different subjects. As is clear from that table, fewer significant differences were visible in this analysis, and the difference between the tutor and the baseline system was exaggerated somewhat. (This latter phenomenon might be coincidental, or might indicate that users with a human tutor in the first half performed better with the baseline system in the second half because of that initial positive experience.)

It is also instructive to consider the degree to which certain help systems performed best for different individuals. In Tables 7-12 and 7-13, the normalized ratios of average task time a subject needed with a given help system to the average for that subject on the baseline system are given for each subject in the experiment. (The normalization is necessary because the tasks in the second half of the experiment are, on average, harder than those in the first half.) These tables show clearly the degree to which the success of the help systems depended on the nature of the individual using them. Most notably, one of the four novices using the simulated English system did unusually poorly, as did one of the experts using a human tutor.

7.7. Retention Results

After each experiment, the subject was given a retention test. This test was designed to see what portion of the experimental tasks were immediately forgotten. The hypothesis was that the help systems which facilitated quick learning would show a reduction in retention, on the theory that people remember best what they have to work the hardest to learn. The results are summarized in Tables 7-14 and 7-15. The only way in which help systems were found to have any significant effect was that ACRONYM actually seemed to help experts to remember what they had learned. However, even this result is of very low significance (*$p < .08$*).

Figure 7-7: The Distribution of Average Novice Task Times



(See Section 7.5 for an explanation of these figures.)

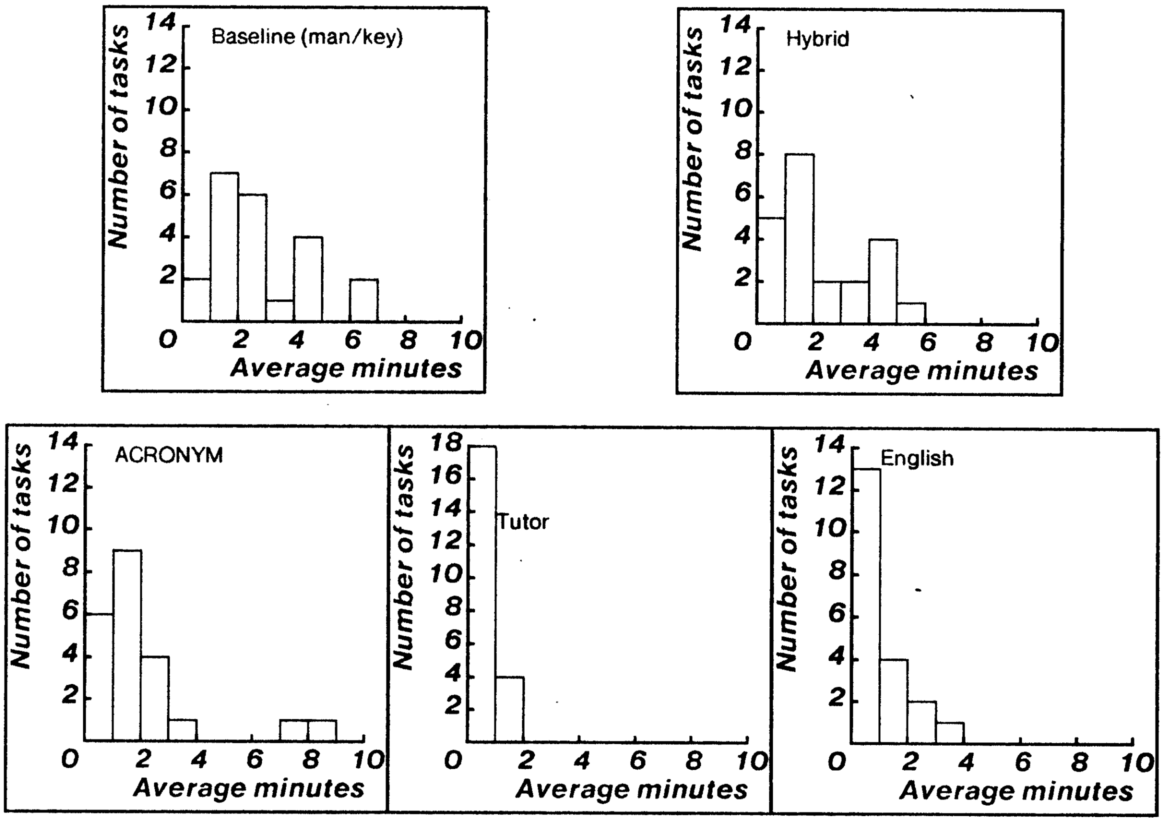
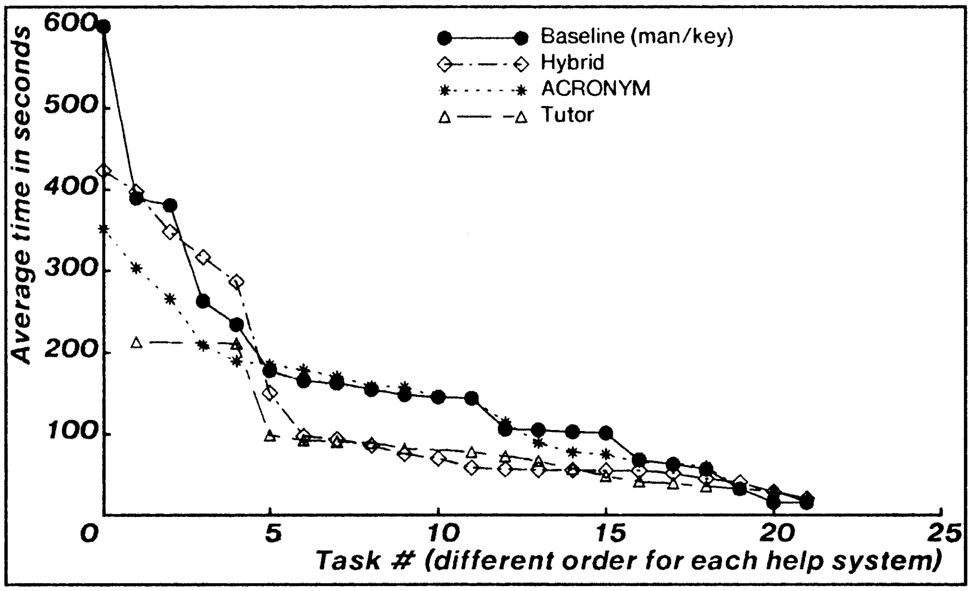


Figure 7-8: The Distribution of Average Expert Task Times



(See Section 7.5 for an explanation of these figures.)

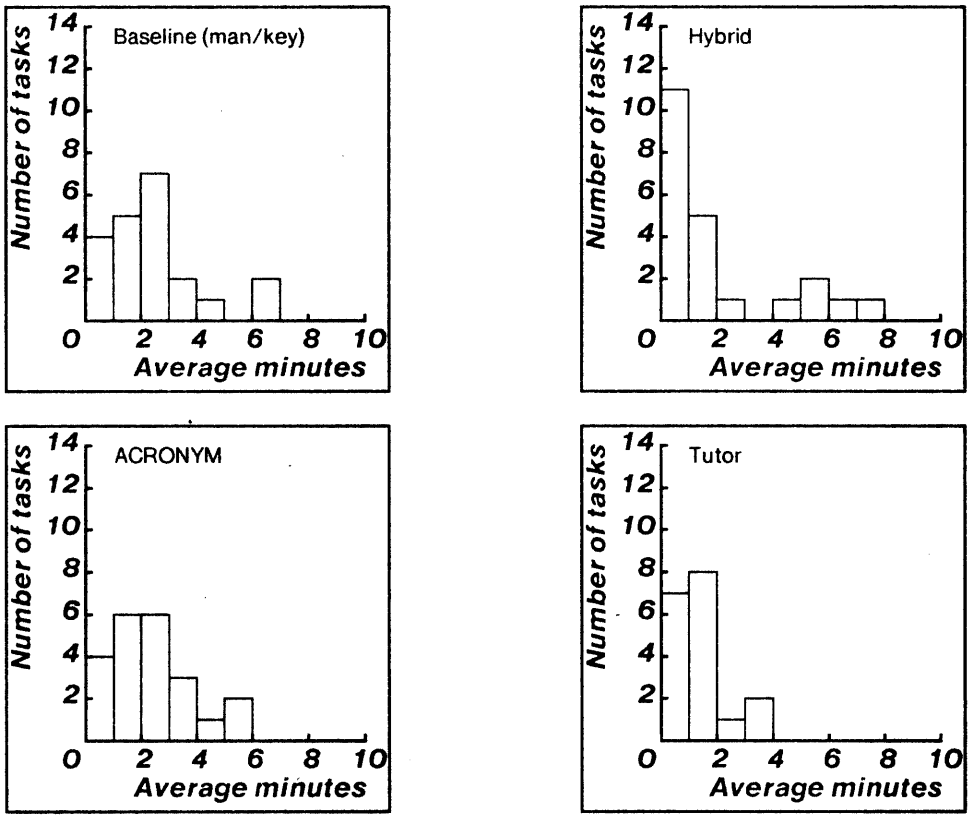


Table 7-10: Novice Regression Analysis, Omitting Within-Subjects Comparisons

$$y=4.29+\sum_{i=1}^{25} c_i x_i$$

Variable	Meaning	Coefficient	T Ratio (Coefficient/SD)
Y	Log of task execution time		
Constant	Task T_1, Help H_0	4.3858	17.87
X_1	Help H_1	-0.0598	-0.39
X_2	Help H_2	-0.1745	-1.16
X_3	Help H_3	-0.8542	-5.73
X_4	Help H_4	-0.4685	-2.92
X_5	Task T_2	0.3699	1.12
X_6	Task T_3	0.1713	0.52
X_7	Task T_4	0.1410	0.43
X_8	Task T_5	-0.3102	-0.95
X_9	Task T_6	0.0817	0.25
X_{10}	Task T_7	0.4983	1.49
X_{11}	Task T_8	-0.1319	-0.41
X_{12}	Task T_9	-0.3133	-0.96
X_{13}	Task T_{10}	-1.1350	-3.49
X_{14}	Task T_{11}	0.7345	2.26
X_{15}	Task T_{12}	0.4455	1.35
X_{16}	Task T_{13}	0.6929	2.07
X_{17}	Task T_{14}	0.0166	0.05
X_{18}	Task T_{15}	0.0442	0.14
X_{19}	Task T_{16}	-0.6390	-1.94
X_{20}	Task T_{17}	0.8428	2.59
X_{21}	Task T_{18}	0.0944	0.29
X_{22}	Task T_{19}	0.4603	1.40
X_{23}	Task T_{20}	-0.4134	-1.10
X_{24}	Task T_{21}	1.1498	3.54
X_{25}	Task T_{22}	0.7622	2.35

Table 7-11: Expert Regression Analysis, Omitting Within-Subjects Comparisons

$$y = 4.29 + \sum_{i=1}^{24} c_i x_i$$

Variable	Meaning	Coefficient	T Ratio (Coefficient/SD)
Y	Log of task execution time		
Constant	Task T_1, Help H_0	4.4350	17.36
X_1	Help H_1	-0.1217	-0.92
X_2	Help H_2	0.0909	0.72
X_3	Help H_3	-0.5106	-3.55
X_4	Task T_2	0.5026	1.53
X_5	Task T_3	-0.9955	-2.89
X_6	Task T_4	0.2351	0.66
X_7	Task T_5	-0.1830	-0.55
X_8	Task T_6	-1.4237	-4.39
X_9	Task T_7	-1.5876	-4.90
X_{10}	Task T_8	0.1498	0.45
X_{11}	Task T_9	1.0905	3.31
X_{12}	Task T_{10}	0.3051	0.91
X_{13}	Task T_{11}	0.0397	0.11
X_{14}	Task T_{12}	0.0582	0.18
X_{15}	Task T_{13}	-0.0709	-0.22
X_{16}	Task T_{14}	0.5437	1.68
X_{17}	Task T_{15}	1.3879	4.21
X_{18}	Task T_{16}	1.2374	3.75
X_{19}	Task T_{17}	0.2603	0.80
X_{20}	Task T_{18}	1.2955	4.00
X_{21}	Task T_{19}	0.0458	0.12
X_{22}	Task T_{20}	-0.2174	-0.65
X_{23}	Task T_{21}	0.0524	0.16
X_{24}	Task T_{22}	-0.2368	-0.73

Table 7-12: The Effect of Novice Subject Variation

Subject	Normalized Baseline Score	Other Help System Used	Normalized Score on Other System	Ratio of Other Score to Baseline Score
S_7	1.0	H_1 (Hybrid)	0.6	0.6
S_0	0.8	H_1 (Hybrid)	0.6	0.7
S_9	2.3	H_1 (Hybrid)	2.1	0.9
S_6	0.7	H_1 (Hybrid)	0.9	1.3
S_8	1.1	H_2 (ACRONYM)	0.5	0.4
S_4	0.8	H_2 (ACRONYM)	0.7	0.9
S_2	1.7	H_2 (ACRONYM)	1.6	0.9
S_{11}	0.9	H_2 (ACRONYM)	0.8	1.0
S_{10}	0.7	H_3 (Tutor)	0.3	0.4
S_5	0.9	H_3 (Tutor)	0.3	0.4
S_1	1.0	H_3 (Tutor)	0.5	0.5
S_3	0.6	H_3 (Tutor)	0.4	0.6
S_{12}	0.5	H_4 (English)	0.3	0.6
S_{13}	0.8	H_4 (English)	0.5	0.6
S_{14}	1.3	H_4 (English)	1.1	0.9
S_{15}	1.0	H_4 (English)	1.9	1.9

Table 7-13: The Effect of Expert Subject Variation

Subject	Normalized Baseline Score	Other Help System Used	Normalized Score on Other System	Ratio of Other Score to Baseline Score
S_1	1.4	H_1 (Hybrid)	0.5	0.4
S_9	1.0	H_1 (Hybrid)	0.7	0.7
S_3	1.1	H_1 (Hybrid)	1.1	1.0
S_5	1.3	H_1 (Hybrid)	1.6	1.3
S_2	1.0	H_2 (ACRONYM)	0.7	0.7
S_4	0.9	H_2 (ACRONYM)	1.0	1.1
S_6	1.1	H_2 (ACRONYM)	1.3	1.2
S_0	1.3	H_2 (ACRONYM)	1.7	1.4
S_{11}	0.6	H_3 (Tutor)	0.3	0.5
S_8	1.0	H_3 (Tutor)	0.6	0.6
S_7	0.5	H_3 (Tutor)	0.7	1.2
S_{10}	0.8	H_3 (Tutor)	2.0	2.6

Table 7-14: Short-term Retention of Solutions by Novices

$$y = 4.29 + \sum_{i=1}^{24} c_i x_i$$

Variable	Meaning	Coefficient	T Ratio (Coefficient/SD)
Y	Average Retention Score		
Constant	Task Set 1, S_1, H_0	9.775	4.53
X_1	Subject S_2	11.000	4.11
X_2	Subject S_3	10.500	3.92
X_3	Subject S_4	10.500	3.92
X_4	Subject S_5	5.250	1.67
X_5	Subject S_6	1.250	0.40
X_6	Subject S_7	6.250	1.99
X_7	Subject S_8	11.750	3.74
X_8	Subject S_9	1.792	0.61
X_9	Subject S_{10}	10.000	3.34
X_{10}	Subject S_{11}	-0.292	-0.10
X_{11}	Subject S_{12}	11.292	3.84
X_{12}	Subject S_{13}	13.208	4.49
X_{13}	Subject S_{14}	9.708	3.30
X_{14}	Subject S_{15}	7.500	2.51
X_{15}	Subject S_{16}	3.292	1.12
X_{16}	Subject S_{17}	6.000	2.01
X_{17}	Subject S_{18}	8.792	2.99
X_{18}	Subject S_{19}	7.500	2.51
X_{19}	Subject S_{20}	13.208	4.49
X_{20}	Task Set 2	-1.0500	-1.24
X_{21}	Help H_1 (Hybrid)	-0.917	-0.59
X_{22}	Help H_2 (ACRONYM)	0.917	0.59
X_{23}	Help H_3 (Tutor)	-1.500	-0.79
X_{24}	Help H_4 (English)	0.500	0.26

7.8. Subjective Results

Finally, an attempt was made to obtain the subjective impressions of a few subjects. As was mentioned previously, a supplementary experiment was conducted in which the same experimental methodology was used, but the subjects were never given the baseline help system; instead, each subject got both H_2 and H_1, ACRONYM and the hybrid system. As was stated previously, this experiment yielded no significant differences in performance between the two systems. However, after each experiment was over, the subject was given a questionnaire asking him to rate the two systems on a number of criteria; for each criterion, subjects were asked whether they strongly or mildly preferred one of the systems, or had no opinion. The results of this survey proved mildly favorable to ACRONYM, as shown in Table 7-16. However, the results of this survey of only four subjects have no statistical significance.

Table 7-15: Short-term Retention of Solutions by Experts

$$y = 4.29 + \sum_{i=1}^{15} c_i x_i$$

Variable	Meaning	Coefficient	T Ratio (Coefficient/SD)
Y	Average Retention Score		
Constant	Task Set 1, S_1, H_0	21.667	13.41
X_1	Subject S_2	-0.000	-0.00
X_2	Subject S_3	-3.000	-1.36
X_3	Subject S_4	-0.500	-0.25
X_4	Subject S_5	0.500	0.25
X_5	Subject S_6	-3.500	-1.58
X_6	Subject S_7	-1.500	-0.68
X_7	Subject S_8	-7.000	-3.16
X_8	Subject S_9	-0.500	-0.23
X_9	Subject S_{10}	-3.500	-1.58
X_{10}	Subject S_{11}	0.000	0.00
X_{11}	Subject S_{12}	-3.000	-1.36
X_{12}	Task Set 2	-1.8333	-2.27
X_{13}	Help H_1 (Hybrid)	-0.500	-0.36
X_{14}	Help H_2 (ACRONYM)	2.500	1.79
X_{15}	Help H_3 (Tutor)	0.500	0.36

Table 7-16: Subjective User Preferences: ACRONYM versus the hybrid

For each statement, the subjects circled a number from 1 to 5, where 1 indicated a strong preference for the man/key system and 5 indicated a strong prefernce for ACRONYM.

Mean (SD)	Statement
3.25 (1.30)	The system quickly found the relevant help information.
3.25 (1.09)	The system quickly told me what I wanted to know.
3.00 (1.58)	The system was easy to use.
3.25 (1.09)	The system presented texts that were easy to understand.
4.00 (0.71)	The system reassured me when I was confused.
3.00 (1.22)	The system made learning how to do the tasks easier.
3.25 (1.48)	I felt in control of the help system.
3.25 (1.48)	I enjoyed using the help system.
3.25 (1.09)	I generally understood what the help system was doing.
3.50 (1.12)	I generally found the answers where I expected to find them.
3.25 (1.09)	The system made using the computer more enjoyable.
3.25 (1.48)	The system made using the computer more productive (made me work faster).
3.29 (1.27)	Overall Average Score

Individual mean ratings ranged from 2.0 (SD 0.91), a mild preference for man/key, to 5.0 (SD 0.00), a strong preference for ACRONYM.

Chapter 8
Conclusions

The research reported in this thesis began with a simple goal: to find out why on-line help systems are commonly so bad, and to see what can be done to make them better. Such a broad goal tends to be elusive, but a number of important discoveries have been made, which will be summarized below.

8.1. What Has Been Learned About Help Systems

The results reported in Chapter 7 include a number of important specific facts about help systems:

1. A good help system can easily make up half the difference between an ordinary bad help system and a human tutor.

2. The most important determining factor in the "goodness" of a help system seems to be the quality and nature of the texts it presents, rather than the details of the help access mechanisms.

3. Although people read faster from paper, a paper manual is not essential; at the worst, moderately sophisticated access mechanisms at a high enough bandwidth can apparently compensate for the total absence of paper.

4. Without spoken input or output or dynamic generation of text using feedback from the user, English does not seem to offer any significant benefit as an interaction language for help systems.

5. Expertise in a large and complex domain seems to be in large part a function of the user's familiarity with the available methods for getting help. Thus expert users benefit less than novices from the presence of a human tutor or a sophisticated but new and different help system.

6. Implementing a moderately sophisticated help system, integrating several different help access mechanisms, is actually a relatively simple and straightforward programming task. Its cost and difficulty are sufficiently low that it seems reasonable to hold designers of future help systems to a much higher standard than the help systems in common use today.

The research clearly indicates that the quality of help texts is far more important than the way those texts are accessed or presented, inasmuch as "important" means "contributes to quickly learning to execute a predefined set of tasks." However, it is surely premature to assume, on the basis of this one set of experiments, that human intuition is entirely unreliable where help systems are concerned. Several people who "should have known better" completely expected that either ACRONYM or the simulated English system would be the clearly superior help system. It is likely that there is some validity to their intuitions, but that those systems offer benefits in ways not measured by this methodology.

In particular, it seems possible that having English available may have reduced the subjects' perception of the difficulty of their tasks, and hence made the learning situation less stressful even if it did not measurably improve performance. Moreover, it should also be noted that the simulated English system was not, in some ways, a complete test of natural-language based help systems. In particular, better results might have been obtained with a system which responded to spoken help requests or which dynamically generated texts to suit the individual. The system tested here did neither. Finally, even if the results do suggest that natural language may not be terribly useful in help systems, it should not be assumed that this result will transfer to other task domains.

As far as ACRONYM is concerned, there are several reasons to suspect that the effort spent on its mechanisms was not wasted. Most important, the common perception of those who use it that it is a good help system should not be discounted: these opinions are, in the end, all that really matters for a system's acceptance. In this light, the most important conclusion of these experiments may be the discovery that user's perceptions of a system's usefulness are not directly related to that system's value for enhancing users' productivity. Additionally, it seems likely that ACRONYM's power actually penalized it on occasion during the experiments reported here. By providing users with a constantly updated menu of related help topics, ACRONYM encouraged them to digress from the task at hand and indulge in exploratory learning. This may have inflated ACRONYM users' scores, while any resulting additional learning could not be measured in the experiment. Moreover, the detailed analysis of the help systems' performance showed a few specific outlying data points for which ACRONYM did unusually badly. It seems likely that an ACRONYM Mark II, with improvements suggested by this data and other observations, might perform significantly better on the same tests. (A brief analysis of the data omitting ACRONYM's outlying data points more than doubled the level of significance with which ACRONYM could be said to be better than the hybrid system, but the difference was still below any reasonable threshold of significance.)

The bottom line, however, appears to be that while fancy features may make the users happy, the most essential factor for getting the job done is almost certainly the quality of the help texts. Those interested in improving the productivity of training programs should therefore concentrate their resources on technical writing rather than elaborate help mechanisms, however flashy or impressive the latter might be.

8.2. What Has Been Learned About Interface Design and Evaluation

Besides the specific information about help system, this thesis has produced a few more general insights into how user interfaces can be designed and evaluated.

8.2.1. The Use of Test-beds in User Interface Design

First of all, the very existence of this thesis is a powerful argument for the usefulness of sophisticated test-beds for interface design. In this case, even though the test-bed, UNIX Emacs [44], was not designed for such purposes, it proved an invaluable tool. In particular, by providing screen, process, and string handling facilities at a very high level, the system allowed ACRONYM to be built quickly and without regard for many irrelevant details. Without such a tool, it is unlikely that a single thesis could have included both the building of ACRONYM and the series of evaluative experiments reported here.

Emacs, however, is far from an ideal test-bed for such purposes. Because Mock Lisp is an interpreted language, it imposes a noticeable performance penalty for computation-dependent features. In ACRONYM's case, this was compensated for by writing the crucial subroutines in C and communicating via the IPC [86], but this mechanism imposes unnecessary low-level details on the interface designer, in large part defeating the entire purpose of a having testbed environment. A true compiler for the interpreted language would have been far preferable.

Mock Lisp also offers too meager a set of data types for serious programming; even for interface design, a wider range of data types is desirable. ACRONYM itself implemented a stack in Mock Lisp as a buffer in which each text line was an element in the stack. Others have even implemented rational numbers, but the effort involved was too large to be believed by anyone but an Mock Lisp programmer [39].

More important, however, a good user interface test-bed would incorporate a number of high-level features beyond the much-appreciated ones provided by Emacs. Such well-known interface paradigms as menus and command grammars could be supported at a much higher level. This is done in a few existing systems, most notably the COUSIN system [50, 52]. In designing such paradigms, it is important not to make the choices available to the interface designer "all-or-nothing" decisions. That is, if the system provides a basic mechanism for menu interfaces, it should be possible to implement a slightly different style of menu interaction without totally scrapping the form provided. This implies that wherever possible the high level mechanisms should be implemented in the same extension language commonly used by the interface designer, so that they can be most easily modified. This is a common principle in the design of software extension languages, as noted by Donner and Notkin [26].

UNIX Emacs as a test-bed for user interface design is discussed at greater length in [9].

8.2.2. Iterative Interface Development

Informal iterative design methodologies are commonly practiced by successful designers of user interfaces. Unfortunately, except in rare cases, such methodologies often cause designs to evolve until they only faintly resemble their original design. This causes well-known problems of software maintenance, as data and program structures are increasingly stretched in directions they were never intended to go.

The methodology of this thesis offers an exciting prospect for avoiding such problems. By building ACRONYM hastily within the framework of a general interface test-bed, the system was in a position to evolve and be evaluated without concern for the integrity of its implementation. After formal and informal evaluations, the system can be scrapped and totally rewritten, without much worry over the cost of the initial version. Thus, the existence of the test-bed not only facilitates more extensive evaluation of experimental systems, it also offers a valuable tool to the interface designer in the real world.

A real world development team with such a test-bed at its disposal might begin with two teams of programmers. One would immediately begin with a prototype system (or series of prototypes) built on the test-bed, while the other team began with the serious implementation of the lowest-level programs in the final system. By the time the interface team's design had evolved to a relatively stable state, the other team would be well-prepared to implement the already-tested interaction paradigms. Just as modern programming methodology has established the desirability of the separation of programs and data, and of data abstraction, similarly future interface test-beds may finally establish the desirability of separating functionality from interface, and of abstracting the details of the functional implementation.

8.3. Practitioner's Summary: Advice for Builders of Future Help Systems

The dominant finding of this thesis was that quality of help texts is far more important than the methods by which those texts are accessed. Therefore, in designing an on-line help system with limited development resources, it makes sense to devote the lion's share of those resources to producing texts of high quality, rather than to building fancy help features. Those features may well be useful and desirable, and may in fact be essential if the goal is a system in which a paper manual is unnecessary, but they should not be implemented at the expense of high-quality texts.

The greatest problem affecting builders of help system in the past has been lack of knowledge of what has already been done. None of the techniques used in ACRONYM, for example, were new. However, each has apparently been invented anew, in a vacuum, by each of its implementors, and no one has studied help systems enough before implementing them to become aware of the good work that has come before. Thus, the most important advice for builders of future help systems is simply to be aware, from the survey in this thesis and whatever material can be found elsewhere, of what kinds of help systems have been built. This makes it more likely that anything you invent will actually be new, and it also allows you to spend time choosing the most appropriate of known techniques rather than inventing new ones.

Additionally, it is essential to think of help information as a knowledge database, which is what it really is, no matter how it is implemented. If the implementation reflects this conception, the knowledge will be stored in a format that is readily accessible to multiple methods of help access. This will not only simplify the task of building a multi-modal integrated help system, but it will make it easier to extend whatever help system *is* built to include new access methods in the future. (Advice like this sounds so painfully obvious that it must be repeated that this has simply never been done in any real world help system!)

Finally, the experience of developing ACRONYM led to a strong belief in the great value of iterative testing of interfaces such as help system. ACRONYM was built only after extensive user surveys and protocols, and hence incorporated a wide variety of techniques founded on a broad base of knowledge about help system. Despite all of this, the experience of observing actual human beings using ACRONYM was an enlightening one; many major and minor flaws and misfeatures became apparent in this process, most of which are listed in Section 5.5. There is simply no substitute for observing real users.

8.4. Researcher's Agenda: Topics for the Future

The evaluation methodology used in this thesis was very successful in detecting differences in the utility of different help systems. However, many questions were left unanswered about which the methodology could still shed much light. Various alternative help system designs could be tested, and it would be particularly interesting to see if a modified version of the hybrid man/key/ACRONYM system which did *not* include a paper manual would perform significantly more poorly than ACRONYM or the hybrid version that included a paper manual. (The hypothesis that it would do so was put forward in Section 7.3.)

Beyond this, the success of the help system evaluation methodology, coming just a few years after the success of the Roberts and Moran text editor evaluation methodology [8, 93, 94, 95], should offer new stimulation to those interested in developing broader objective measures of the quality of user interfaces. Indeed, the methodology used in this thesis was probably unnecessarily narrow in its focus. An interesting direction for future research will be to try to broaden the orientation and scope of the task list, so that the methodology can become a more general method for evaluating the basic learnability of any operating system interface, including its help system.

Methodologies such as the one described in this thesis and the one described by Roberts and Moran generally yield only crude measures, detecting very large differences such as "TECO is not as good as BRAVO" or "ACRONYM is better than the standard UNIX help system". They are inadequate for answering questions about the lower-level details of user interface design. To answer these detailed questions, a whole host of unique experiments must be designed, in the tradition of human factors experiments. Such research is well-established in other disciplines, but is almost virgin territory for those who would evaluate user interfaces.

Finally, the results of the survey and taxonomy of help systems presented in this thesis lead inevitably to the conclusion that there have been very few interesting ideas in the history of on-line help systems. The same paradigms have been reinvented many times, and implemented in many and varied (but usually deficient) particular systems, but the actual number of ways to give the user help is still very small. This may be seen either as a challenge to the creativity of software engineers, who may yet come up with new methods, or as a phenomenon worthy of deeper understanding. Is there some fundamental set of facts about the way people interact with computers that dictates the rather small set of ways in which we have been able to make the computers help people to learn to use them? Or have we not yet really opened our eyes to the possibilities before us?

Part Four

Appendices and Bibliography

Appendix A
Annotated Experimental Materials

This appendix contains all of the materials actually used in the experiments described in this thesis.

The first part of this material is an initial questionnaire, which was used to determine which, if any, of the expertise classifications a subject belonged to. The next sections are the introductory/explanatory material which each subject read, followed by a typing test which each subject was required to perform. After these preliminaries come all of the actual materials presented to novice subjects, and then the materials presented to expert subjects.

Comments like this one, in italics off to the right, are not part of the experimental materials themselves, but are used to explain those materials. They were not seen by the subjects. Other than these comments, the only difference between the materials presented here and those seen by the subjects are differences of page numbering and other minor formatting differences.

Initial Questionnaire

What follows is a list of tasks that can be done on a computer. For each task, please tell whether you have ever performed that task on a computer, whether you have ever performed it on a computer running the UNIX operating system, and whether you have ever performed it on a computer running the TOPS-20 operating system. If you don't understand the description of a task, you may ask the experimenter for clarification.

For each task, check "Any" if you have ever performed the task on any computer. Also check "UNIX" if you have performed the task using UNIX and could do so again with minimal effort, and check "TOPS-20" if you have performed the task using TOPS-20 and could do so again with minimal effort. "Minimal effort" means simply that you would need no more than a brief reminder to jog your memory.

_Any	_UNIX	_TOPS20	Get a list of all the files in your current directory or account.
_Any	_UNIX	_TOPS20	View the contents of a file stored on the disk.
_Any	_UNIX	_TOPS20	Make a second copy of a file on the disk.
_Any	_UNIX	_TOPS20	Change the name of a file on the disk.
_Any	_UNIX	_TOPS20	Get a copy of a file printed on paper.
_Any	_UNIX	_TOPS20	Delete a file from the disk.
_Any	_UNIX	_TOPS20	Get a list of all the files deleted from your current directory, but still recoverable (not permanently deleted).
_Any	_UNIX	_TOPS20	Restore (undelete) a deleted file.
_Any	_UNIX	_TOPS20	Find out the name of your current directory or account.
_Any	_UNIX	_TOPS20	Create a new directory as a subdirectory of your current one.
_Any	_UNIX	_TOPS20	Change to another working directory.
_Any	_UNIX	_TOPS20	Move a file from one directory to another.

_Any	_UNIX	_TOPS20	Delete an empty directory
_Any	_UNIX	_TOPS20	Delete a directory and all its contents.
_Any	_UNIX	_TOPS20	Send mail to another computer user.
_Any	_UNIX	_TOPS20	Read mail from another computer user.
_Any	_UNIX	_TOPS20	Find out if a certain person is currently using the computer.
_Any	_UNIX	_TOPS20	Copy a file *from* your machine *to* another machine on the same computer network.
_Any	_UNIX	_TOPS20	Copy a file *to* your machine *from* another machine on the same computer network.
_Any	_UNIX	_TOPS20	Find all occurrences of a certain word in a file on the disk, without using a text editor.
_Any	_UNIX	_TOPS20	Find out what time it is.
_Any	_UNIX	_TOPS20	Change your password.
_Any	_UNIX	_TOPS20	Send a message to another user, making it appear immediately on his screen.
_Any	_UNIX	_TOPS20	List everyone currently using the computer.

What You'll Be Doing

In this experiment, you will be asked to perform certain tasks on a computer using the UNIX operating system. Most of these tasks will probably be things you've never done before. For each task, you will be given an explanation of what it is you are supposed to do, and then you will attempt to learn how to do it. Please try as hard as you can, but don't worry if you can't figure a few of the tasks out.

Before we start, please take a few moments to familiarize yourself with the keyboard. Practice typing a few things, to get used to typing on it. Don't worry about anything the computer might say in response to what you type to it.

One key that may be unfamiliar to you is the "Control" key ("Ctrl"). The control key is like the shift key in that holding it down alters the meaning of the other keys on the keyboard. For example, holding down "ctrl" and typing "h" is called "control-h" and can be used to erase the last character you typed. There are a few such control keys that are very useful in typing on the computer, and these are listed below. If you've never used them before, please try them out and make sure you understand what they do.

- ***control-h*** deletes the last character you typed. You can use it several times in a row to delete several characters.

- ***control-u*** deletes the entire line you have just typed, as long as you haven't pressed the RETURN key.

- ***control-s*** stops the computer from printing output faster than you can read it. When you type ***control-s***, the computer will stop printing until you type ***control-q***.

- ***control-c*** tells the computer to stop whatever it is doing. You can type ***control-c*** if you give some command by mistake and just want the machine to quit what it is doing and start over with a new command.

When you feel comfortable typing on the computer keyboard, please turn the page.

Typing Test

The first part of the experiment is a typing test. Type "typing" on the computer keyboard and then press the RETURN key. When the computer says "Ready for typing test," please turn the page and type the paragraph you see there. If you make an error and you notice it right away, correct it by erasing with control-h. If you notice it after you've typed a few more words, however, just ignore the mistake and go on typing.

At the end of each line of text in the typing test, please press the RETURN key, just as you would on a typewriter.

Type This Paragraph:

I lost the boundary of my physical body. I had my skin, of course, but I felt I was standing in the center of the cosmos. I saw people coming toward me, but all were the same man. All were myself. I had never known this world before. I had believed that I was created, but now I must change my opinion: I was never created. I was the cosmos; no individual existed.

When you have finished, and are ready to go on, please turn the page and type the paragraph you see there.

Type This Paragraph:

Dispute not. As you rest firmly on your own faith and opinion, allow others also the equal liberty to stand by their own faiths and opinions. By mere disputation you will never succeed in convincing another of his error. When the grace of God descends on him, each one will understand his own mistakes.

Please DO NOT turn the page until the experimenter says it is OK. Right now, the experimenter will give you a separate page of instructions to read. While you are reading that page, the experimenter will type a few commands on your keyboard in preparation for the next part of this experiment.

The mentioned page of the instructions varied according to the independent variable, which is the type of help system being used. That page explained the use of whichever help system was being made available to the subject in the first part of the experiment. All of those pages are presented in sequence on the pages that follow.

Instructions for Using the Help Systems

For each help system studied in the experiments, a short explanation of the use of the help system was provided. Since the independent variable in the experiments was the help system, the explanatory material was the only source of variation in the materials seen by the subjects. Before each of the two task sets, each subject saw the explanatory material for one help system only, namely the help system he would be using in the following task set. All of those pages are presented together here, but it should be remembered that this is not how the subjects saw them; what the subjects saw was a single set of instructions at this point, corresponding to the help system they used in the first half of the experiment.

Getting Help

Help condition H_0: Naked UNIX with ***man*** *and* ***key*** *or H_1: Hybrid system with man and key using ACRONYM database.*

As stated before, you will be given a series of tasks to try to perform on the computer. Since you won't generally know how to do these tasks in advance, you will have to learn the right way to do them. To do this, you will need to use the UNIX help system.

The UNIX help system consists of two separate help commands. The first is the *man* command. If, for example, you are trying to find out how to use a program called "build" you may type "***man build***" to the computer. It will then print out a full description of the *build* command. If the description is longer than will fit on your screen, it will print out one screen full and will then type "-- More --". It will then wait for you to press the space key before continuing with the description. If you don't want it to finish the explanation, you can press the "q" key instead of the space key. This will cause it to stop giving you that help message.

The other help command available to you is called *key*. This command may be used when you don't know the name of the command you are looking for. For example, if you wanted to construct something, but you didn't know that the right command for doing this was called "build," you wouldn't be able to use the *man* command to get help. Instead, you could type "***key construct***" to the computer. The *key* command takes the word you give it (a keyword) and looks for commands that might be relevant to that key word. Thus, if you typed "***key construct***" it might tell you about the *build* command. Actually, you don't even have to give "key" complete words; a piece of a word such as "con" can often be more useful than a complete word such as "construct."

Key prints only short descriptions of commands; its purpose is to help you find the command you're looking for. You can then get detailed information about it by using the *man* command, as described above. Note: The ***key*** command will list a large number of manual entries which are not actually commands, and should be ignored. Each entry will have a number in parenthesis after it when listed by the key command. Only those entires with the number "(1)" are relevant to the tasks you will be performing. Entries with any other number should be ignored.

In addition to the *key* and *man* commands, you will also be given a paper copy of the UNIX manual. The entries in this manual correspond precisely to the explanations printed by the *man* command. Thus it is equivalent to type "man build" or to look up "build" in the paper manual.

In addition to the paper manual, you will also have at your disposal a booklet titled "UNIX for Beginners," which you may use as an additional source of information if you so desire. It is less complete than the other information, but somewhat easier to read and understand. for this experiment, it is not recommended that you try to read that booklet in its entirety, but rather that you scan through it in search of particular information as you need it.

The paper manual and the commands described above are the only sources of help you will have for this experiment. You will not be allowed to ask the experimenter for help in performing the tasks. However, you may ask the experimenter if you have any questions about how to use the help commands or the paper manual. You may also ask the experimenter to clarify the descriptions of the tasks if you don't understand them.

If you have any questions at this time, please ask the experimenter.

Getting Help

Help condition H_2: ACRONYM

As stated before, you will be given a series of tasks to try to perform on the computer. Since you won't generally know how to do these tasks in advance, you will have to learn the right way to do them. To do this, you will use the ACRONYM help system. ACRONYM is simply a system that gives you help in using the computer in several ways.

You will notice that your screen is divided into five white areas, separated by dark lines. The top such area is not used by ACRONYM; this is the small white area at the very top of the screen. You should simply ignore this section of the screen.

The second part of your screen is called the "Help Texts". In this area, ACRONYM will print explanations of various sorts; this is where you will actually read the help that ACRONYM is giving you.

The third part of your screen is called the "Help Menu". In this area, you will find a list of topics for which further help is available.

The fourth white area, the last large white area on your screen, is called the "Commands Window". In this area of the screen you will actually type commands to the computer and the computer will respond to you.

Finally, there is a very small white area -- just one line -- at the bottom of your screen. ACRONYM will occasionally use this to give you short messages and to allow you to ask certain questions, as described below.

Now, before you read any further, please make sure that you recognize each of the parts of the screen described above.

How to get Help

ACRONYM gives you help in four different ways. The first, and simplest, is that it simply watches what you type and updates the help in the Help Texts accordingly. At any given moment, therefore, the help in the Help Texts should be at least somewhat relevant to what you are currently doing.

Sometimes you can make ACRONYM give you more help based on what you're currently doing by typing a question mark ("?"). The question mark can also be used to return to ACRONYM's best guess about what kind of help you need, if you have caused the help to be changed by giving any of the further help commands described below. Often, however, you will find that a question mark doesn't cause the help to change at all; in such a case, you'll need to use one of the two remaining methods of getting help.

(NOTE: Sometimes you need to type a question mark as part of a command, rather than as a request for help. On these rare occasions, you can actually cause a question mark to be inserted like any other character you type by preceding it with either ↑Q (control-q) or a back-slash ("\").)

The most common way of requesting more help from ACRONYM is to choose one of the items in the Help Menu. If you want to see the help described on a line in the Help Menu, you should simply use the "mouse" to point to it. The "mouse" is the little box-shaped object near your keyboard. You will notice that as you move the mouse around on the plastic sheet, an arrow will move around on the screen. Try it and see. You can use the mouse to position the arrow so that it is pointing at the menu item that interests you. Then, if you press any button on the mouse, ACRONYM will show you the help messages on the topic you pointed to. You should especially note that the help menu for every command includes an "examples" section, a "summary" section, and more detailed sections to explain various points and concepts.

The mouse you are using, incidentally, is a little picky about where exactly the arrow is pointing. You should make sure that the body of the arrow is mostly BELOW the line you are pointing at; the arrowhead should be pointing at the bottom of the line you are pointing at, not the middle or top.

Often, the message in the Help Texts or the choices in the Help Menu are too long to fit in the appropriate region of your screen. In such a case, a special phrase will appear in the line below that region, saying "Press HERE to scroll forward" or "Press HERE to scroll backward". By pointing at the word "HERE" with the mouse, and pressing a mouse button, you can cause the part of the Help Texts or Help Menu that is not now visible to be moved onto the screen. It is important to notice when the "Press HERE to scroll..." messages appear, because if you don't notice you may never see the help messages you need most.

Finally, if none of the help on the screen seems to be doing you any good, just type the word "help" and press the SPACE key. ACRONYM will ask you to type a key word about which you want help. (It will ask you for this key word in the message area, the small white area at the very bottom of the screen.) Thus, if you type "help" and press SPACE, you can then type the word "cherry" to get ACRONYM to give you whatever help it can find related to the word "cherry". It won't always succeed in finding the help you want, but it often will.

Please experiment with the commands above and try to make sure you understand how they work. If you have any questions at this time, please ask the experimenter.

Getting Help

Help condition H_3: Human Tutor

As stated before, you will be given a series of tasks to try to perform on the computer. Since you won't generally know how to do these tasks in advance, you will have to learn the right way to do them. To do this, you can ask the experimenter any questions you want. He will answer whatever questions you ask, so all you have to do is figure out the right questions to ask him.

The experimenter is the only source of help you will have for this experiment. You will not be given an instruction manual, and the normal UNIX help facilities will not be available to you.

If you have any questions at this time, please ask the experimenter.

Getting Help

Help condition H_4: Simulated English

As stated before, you will be given a series of tasks to perform on the computer. Since you won't generally know how to do these tasks in advance, you will have to learn the right way to do them. To do this, you can ask the computer any question you want, in English.

This computer is running a new English-language help system. You can't actually give the computer commands in English; all of its commands are in a strict command language, which must be conformed to exactly. However, your questions *about* that command language may be typed in normal, everyday English.

The computer will be the only source of help you will have for this experiment. You will not be given an instruction manual, and the normal UNIX help facilities will not be available to you.

If you have any questions at this time, please ask the experimenter.

The Experimental Tasks for Novices

On the pages that follow are the actual tasks as they were given to the novice subjects. Only one task is given on each page, to prevent the subjects from reading ahead.

Task:

Task N_1: date, uptime, whenis

Get the computer to tell you the correct time of day.

Note that this is deliberately worded to distract the subject away from the correct guess. Had the subject been asked to get the computer to print out the date and time, "date" might be a common guess. However, getting the computer to tell the time is a more likely desire, and is more likely to require help, since the correct command for obtaining the time is "date."

Task:

Task N_2: passwd

Everyone who uses the computer has a secret password which he uses at the beginning of each session with the computer, in order to prove his identity to the computer. You haven't needed to provide this password because the experimenter took care of that before you began. However, your task now is to change that password.

The password before now was "dinner". Your task is to change the password to "breakfast".

Task:

Task N_3: ls

A computer *file* is an organized unit of information, such as a manuscript. The computer you are using has thousands of files. To make it easier to find things, and for other reasons, these files are arranged into groups called "directories". At any given moment, you are directly connected to one of these directories, and you may refer to the files in that directory simply by their names.

Your task is to get the computer to list the files in your current directory.

Task:

Task N_4: cat, pr, more

One of the files in your current directory, you may have noticed, is a file named "readme". Your task is to read that file -- that is, to view its contents on your screen.

Task:

Task N_5: cp

Sometimes it is desirable to have two copies of the same file. Your task now is to make a second copy of the file called "readme" and to name that second copy "readme2".

Task:

Task N_6: mv

File names can be changed. Often it becomes clear that a file was not given a very good name in the first place, and it needs to be renamed. Your task now is to change the name of the file "readme2" to "copyofreadme". Do this *without* making a new copy of the file.

Task:

Task N_7: cz

Your task now is to create a paper copy of the file "readme" that you looked at before. The device that is used in our department to get paper copies of files is called the *Dover*. Thus, your task is to print the file "readme" on the Dover.

Task:

Task N_8: del

Since we don't really need the file "copyofreadme", your next task is to get rid of it. However, our version of UNIX has two different commands that can be used to get rid of unwanted files. One of these commands, "rm," gets rid of files permanently, while the other allows you to change your mind later and "undelete" them again. Your task is to get rid of "copyofreadme", but in such a way that we can get it back again later if we so desire. In other words, don't use "rm" -- use the other file deletion command.

Task:

Task N_9: lsd

Just as you earlier got a list of the files in your current directory, your new task is to get a listof the files that have been *deleted* from your current directory.

Task:

Task N_{10}: undel

Restore the file that you deleted before, “copyofreadme”, back into your current directory.

Task:

Task N_{11}: send, write, rsend

As it turns out, the person called nsb is currently using this computer in another location. Your task is to get the computer to print a message on his screen that says "Dinner is served." Note that you don't want to start a conversation with him, but merely to print that single message on his terminal.

New Instructions

In the next part of the experiment, you will be given more tasks to perform on the computer. However, now the method in which you get help from the computer will change. Instead of getting help as you have so far, you will have to use a different method. The new method of getting help is described in a new set of instructions which the experimenter will now give to you. While you are reading the new instructions, the experimenter will type a few commands on your keyboard.

At this point in the experiment, the experimenter has to type something to turn the old help system off and to turn the new one on. At this point in the experimental materials, another one of the help system explanations, which were presented as a group earlier in this chapter, would be seen by each subject. This new explanation, of course, corresponded to the help system the subject used in the second half of the experiment.

Task:

Task N_{12}: cal

Your task is to get the computer to print on your screen a calendar for the month of August, 1984.

Task:

Task N_{13}: rev

Once again your task is to print the file "readme" on your screen, but this time you should print the entire file *backwards* -- that is, printing each line in reverse order.

Task:

Task N_{14}: pwd

As we have said before, you are always "connected" to some particular directory which contains your files. Until now, we haven't paid any attention to the directory name, and have only dealt with files in your current directory. In order to learn more about directories, you need first to find out the name of the directory you are currently working with. Thus your task is to find out the name of your current directory.

Task:

Task N_{15}: mkdir

Your next task is to create a new directory within your current directory. The new directory should be called "newdir". Note that a directory is just a special kind of file, and will look just like a file if you get a list of the files in your directory with the "ls" command.

Task:

Task N_{16}: cd, chdir

Your current directory has a subdirectory called "urgent". Your task now is to change your current directory to be that "urgent" subdirectory, instead of what it is now.

Task:

Task N_{17}: mv

Now that you have made "urgent" your current directory, you will find that it has a file named "frank" and a subdirectory named "beans". Without copying the file, move the file "frank" out of your current directory and into the "beans" subdirectory.

Task:

Task N_{18}: rmdir

Your current directory has another subdirectory called "empty". This directory is, as its name implies, empty -- there are no files in it. Your task is to delete that empty directory.

Task:

Task N_{19}: rm -r

Your current directory has yet another subdirectory called "full". This directory has several files in it. Your task is to delete the directory and all its files with a single command.

Task:

Task N_{20}*: u, users, finger, w, who*

Your task now is to get a list of all the people currently using this computer.

Task:

Task N_{21}: grep

There is a file in your current directory named "searchme". It is a very long file, too long to read quickly in its entirety. Your task is to get the computer to search through the file and only print out those lines that contain the word "chocolate".

Task:

Task N_{22}: mail

Electronic mail is a convenient way for two computer users to send messages to one another. There is a user of this machine who is known to the computer as "nsb". Your task is to send a piece of mail to nsb.

The *subject* of your mail should be "Good News." The actual message should simply state, "Keep working, you'll get done eventually."

The Experimental Tasks for Experts

On the pages that follow are the actual tasks as they were given to the expert subjects. Only one task is given on each page, to prevent the subjects from reading ahead.

Task:

Task E_I: cz -f

Print the file sample.c on the Dover, using TimesRoman12 as the font.

Task:

Task E_2: sort -r

There is a file in your current directory called "unsorted". Sort that file alphabetically by line in *reverse* alphabetical order, using a single command to sort the text and print the sorted output on our screen.

Task:

Task E_3: ls -t

Get a list of all files in your current directory, sorted by time modified rather than by file name.

Task:

Task E_4: chmod o-r, chmod 640

In your current directory is a file called "shared". Currently it is readable by anyone, but writable only by you. Change its protection status so that only you or members of your login groups can read it.

Task:

Task E_5: del

First, change to the directory "rodeo" by typing "cd rodeo".

This version of UNIX has two commands that can be used to get rid of unwanted files. You are probably already familiar with one of these, ***rm***. The other command, however, gets rid of files without erasing them, so that it is possible to restore them if they were deleted by mistake. Your task now is to delete the file "useless" from your current directory, but to do it in such a way that it can later be restored if necessary.

Task:

Task E_6: lsd

Get a list of all the files that have been deleted from your current directory but can still be restored.

Task:

Task E_7: undel

Restore to your current directory the file "useless" that you deleted a short time ago.

Task:

Task E_8: ls -i

First, type "cd .." to change your back to the parent of your current directory.

There is a file in your new current directory called "whatnumber". Find out what the i-number of that file is.

Task:

Task E_9: chmod u+s, chmod 4xxx

There is a file in your current directory called "runme". It is an executable (runnable) file. Your task now is to change its protection status so that when it runs, whoever runs it has the privileges of its owner during its execution. This is known as "setting the setuid bit".

Task:

Task E_{10}: send -all, send nsb ttyxx

Send the message "Time for dinner" to the terminal of nsb, who is currently using this machine.

nsb was always logged on twice, causing the program to complain and require special instructions.

Task:

Task E_{11}: ps a

Your next task is to get a list of all processes currently running on the system, for all users (not just your processes).

New Instructions

In the next part of the experiment, you will be given more tasks to perform on the computer. However, now the method in which you get help from the computer will change. Instead of getting help as you have so far, you will have to use a different method. The new method of getting help is described in a new set of instructions which the experimenter will now give to you. While you are reading the new instructions, the experimenter will type a few commands on your keyboard.

At this point the experimenter had to type something to turn the old help system off and to turn the new one on. This page in the experimental materials was followed by the description of the help system the subject was to use in the second half of the experiment; all such pages were reproduced earlier in this appendix.

Task:

Task E_{12}: cz -h "..."

In your current directory is a file called "niceday". Please print that file on the dover with the phrase "Have a nice day" at the top of each page.

Task:

Task E_{13}: sort -f

Your next task is to sort the file "sortme" alphabetically by line, *ignoring capitalization.*

Task:

Task E_{14}: mailq -retain

Cancel all of your pending (queued) requests to send mail to other machines on the network. (Even if there aren't any pending requests, type the command that would cancel them if there were.)

Task:

Task E_{15}: strings

There is a press file (binary Dover format file) in your current directory called "out.press". Press files are full of control characters, which are unreadable, as well as the actual text they are supposed to print. Your task now is to view on your screen the readable (text) characters in that file (out.press), without printing the control characters, which could mess up your terminal.

Task:

Task E_{16}: cmuftp r g - = "/date"

Another machine connected to this one on the network is CMU-CS-G. It is possible for you, logged in on one machine, to execute commands on other such machines on the network. Your task now is to get CMU-CS-G to tell you the date and time. Then ask your current machine to tell you the date and time; they rarely agree. (Hint: Use the program "cmuftp".)

A common error in giving this command actually causes the current machine to print the date and time. Thus, the request for comparison is actually an opportunity for the subject to discover his error. This task is not quite satisfactory: what I want is a simple command to be executed on a remote site via the ftp/cmuftp mechanism. However, simple directory listings and listings of users aren't acceptable, because these may now be done without touching ftp or cmuftp. ***finger@g*** *or* ***ls "[g]/usr/foo"*** *will do those jobs without the subject having to learn how to use ftp for remote processes. I may replace the "date and time" task with something else if I come up with any better ideas.*

Task:

Task E_{17}: undel -g

There is a directory in your current directory called "dir2". If you change to that directory ("cd dir2") and type "lsd", you will find that there are four old deleted versions of a file called "deleteme" that are still recoverable. Your task now is to undelete the *oldest* version (original) version of that file, leaving the newer versions deleted. (Note that the oldest file has the lowest generation number.)

Task:

Task E_{18}: ecp -u guest guest "[onyx]<AltoDocs>chat.tty" chat.tty

One machine on the Ethernet is called "onyx". On that machine is a file called <AltoDocs>chat.tty. Your task is to copy that file from Onyx onto your current directory. You can use the account "guest" with password "guest". Warning: Neither "ftp" nor "cmuftp" will work for transfers to and from Onyx.

Task:

Task E_{19}: rsend user@host, send user@host

The user nsb is also logged in on the machine CMU-CS-K. Send a message to his terminal on that machine that says "Time for lunch".

Task:

Task E_{20}: ps tpa

Get a list of all processes that are currently running on the terminal "ttypa". Use a single command to do this.

Task:

Task E_{21}: ps tp?

Get a list of all processes that are currently running on this machine but are not associated with ANY terminal.

Task:

Task E_{22}: lsd -t

Find out how much total disk space your *deleted* files in this directory are taking up. Do this with the "lsd" command.

Discussion and Posttest

After the experimental tasks were all completed, subjects were debriefed and were given a posttest to analyze their short-term retention. The associated materials appear on the following pages.

Discussion

At this point, the experimenter would like to talk to you about what you have done in this experiment. When he is done talking to you, there will be one final short quiz about what you have learned.

The "debriefing" session was a fruitful source of anecdotal information and general insights into the subjects' joys and frustrations involved in the use of the help system.

UNIX Learning: Posttest

Novice version

What follows is a list of tasks, similar to those you performed or attempted to perform during the experiment. Next to each task, please write down the command that should be used to perform the task, if you remember it.

__________________________List the names of the files in your current directory.

__________________________Print the contents a file on your screen.

__________________________Make a copy of a file.

__________________________Rename a file.

__________________________Print a file on paper on the Dover printer.

__________________________Send mail.

__________________________Delete a file.

__________________________Get a list of your deleted files.

__________________________Restore a deleted file

__________________________Print the name of your current directory.

__________________________Create a new directory.

__________________________Change your current directory.

__________________________Move a file from one directory to another.

__________________________Delete an empty directory.

__________________________Delete a non-empty directory and all its contents.

__________________________Get a list of everyone currently using your computer.

__________________________Display only those lines in a large file which contain a certain word.

__________________________Send a message to the screen of someone else who is currently using the computer.

__________________________Print today's date and time.

______________________________Change your password.

______________________________Print the calendar for a certain month.

______________________________Print a file backwards on your screen.

UNIX Learning: Posttest

Expert Version

What follows is a list of tasks, similar to those you performed or attempted to perform during the experiment. Next to each task, please write down the command that should be used to perform the task, if you remember it.

______________________________Print a file on the Dover with every page labeled with a certain phrase at the top.

______________________________Print a file on the Dover using a special font.

______________________________Sort a text file alphabetically by line, ignoring capitalization.

______________________________Sort a file in reverse alphabetical order.

______________________________Copy a file from the machine called Onyx.

______________________________Find out the i-number of a file.

______________________________List all files in your current directory, sorted by time modified.

______________________________Cancel all outgoing mail requests.

______________________________View only the readable characters in a press file.

______________________________Reversibly delete a file.

______________________________Get a list of deleted files.

______________________________Restore a deleted file.

______________________________Restore the oldest of several versions of a deleted file.

______________________________Make a file writable by all members of your groups.

______________________________Make an executable file set the setuid bit when it runs.

______________________________Send a message to the terminal of someone who is logged in at more than one terminal.

____________________________Send a message to the terminal of someone who is logged in on another machine.

____________________________List all the processes currently running for all users.

____________________________List all the processes currently running on a particular terminal.

____________________________List all the processes currently running on no terminal.

____________________________Execute a command on another machine on the network.

____________________________Find the total amount of disk space occupied by your deleted files.

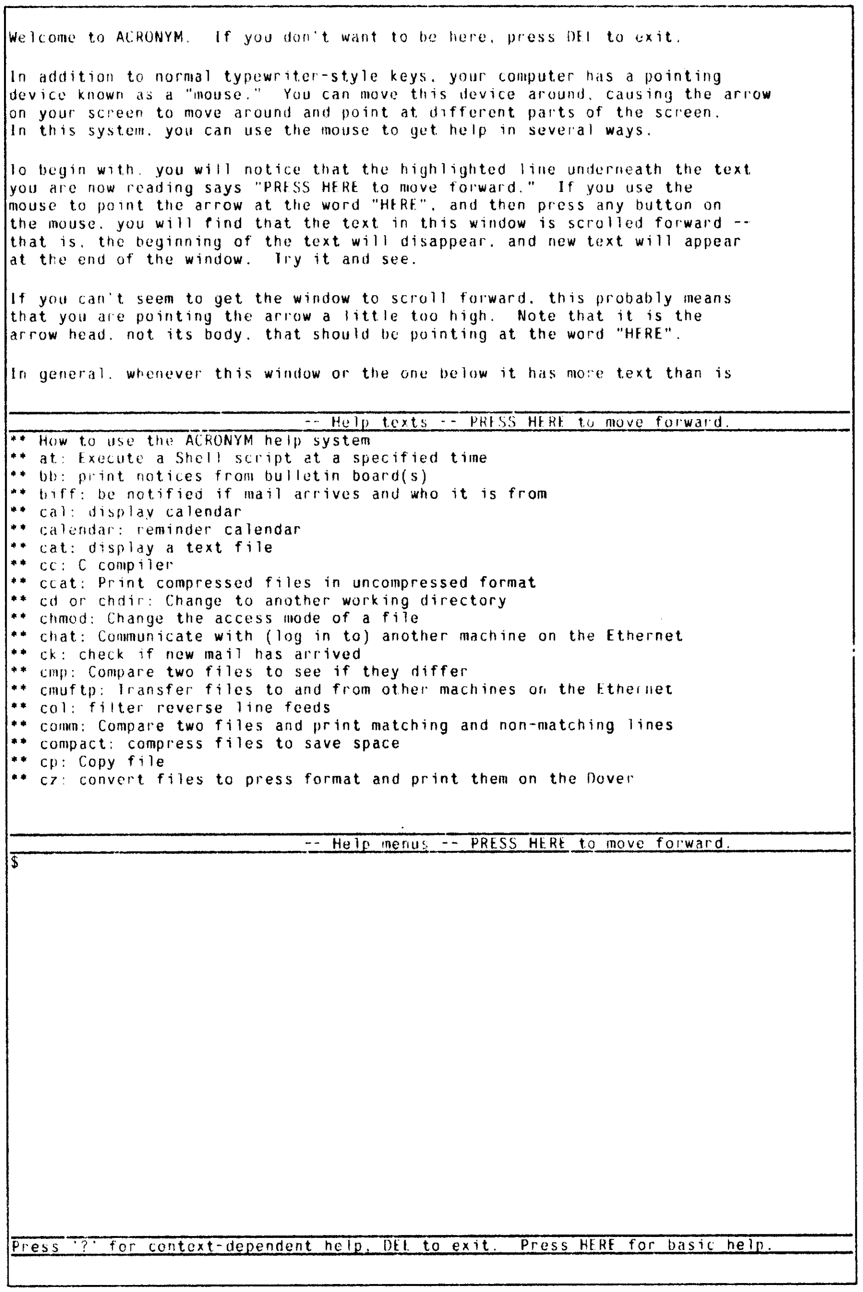
Appendix B
What ACRONYM Looked Like to the User

Without using a piece of software, it is generally difficult to obtain an accurate picture of how that software actually works. Since the degree to which the results of this thesis are interesting depends, in significant measure, on what you think of ACRONYM as a help system, it is especially important, for the understanding of this thesis, to have a good grasp of what ACRONYM was like.

To help readers who have never been able to see ACRONYM in action, this appendix shows how ACRONYM actually looked in use via a series of "snapshot" screens. These screens reproduce actual ACRONYM screens as faithfully as possible. Unfortunately, they do not capture the location of the cursor or the pointer controlled by the mouse, so these will be described in running text under each screen picture. Also, the "mode lines" which separate ACRONYM's windows were reverse video (white on black) in ACRONYM, but are here simply shown as ordinary text surrounded by lines. Finally, the rather clumsy technology used to produce these screen images provided an undesired excess of white space at the bottom of each window; in actual use, each ACRONYM window, when full, contained text right up to the mode line.

The first screen picture shows the ACRONYM screen as users saw it at the beginning of the experiments; this is what ACRONYM looks like when it starts.

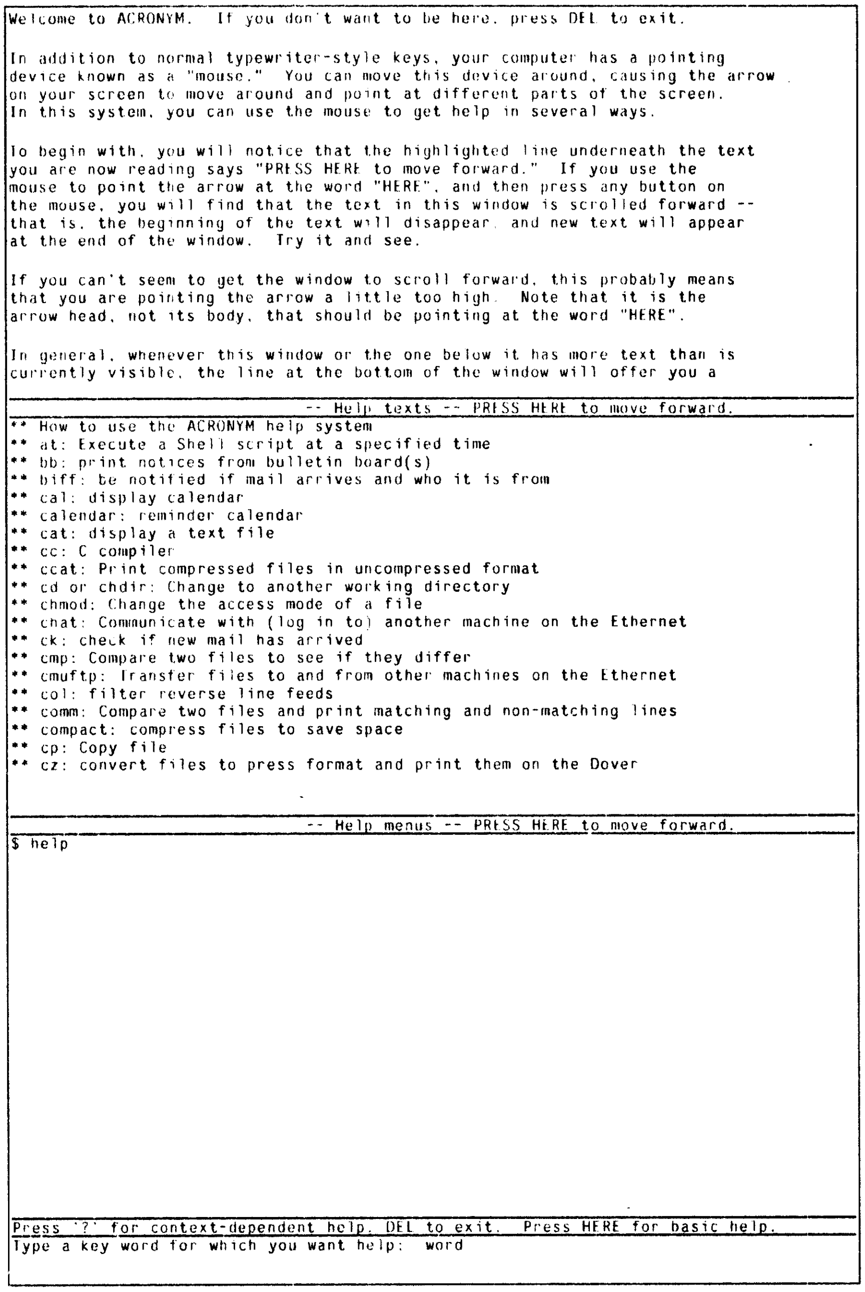
Those not familiar with UNIX should be aware that the dollar sign in the command window is the standard UNIX prompt, signalling UNIX's readiness for a new command.



```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is

                                   -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** at: Execute a Shell script at a specified time
** bb: print notices from bulletin board(s)
** biff: be notified if mail arrives and who it is from
** cal: display calendar
** calendar: reminder calendar
** cat: display a text file
** cc: C compiler
** ccat: Print compressed files in uncompressed format
** cd or chdir: Change to another working directory
** chmod: Change the access mode of a file
** chat: Communicate with (log in to) another machine on the Ethernet
** ck: check if new mail has arrived
** cmp: Compare two files to see if they differ
** cmuftp: Transfer files to and from other machines on the Ethernet
** col: filter reverse line feeds
** comm: Compare two files and print matching and non-matching lines
** compact: compress files to save space
** cp: Copy file
** cz: convert files to press format and print them on the Dover

                                   -- Help menus -- PRESS HERE to move forward.
$









Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

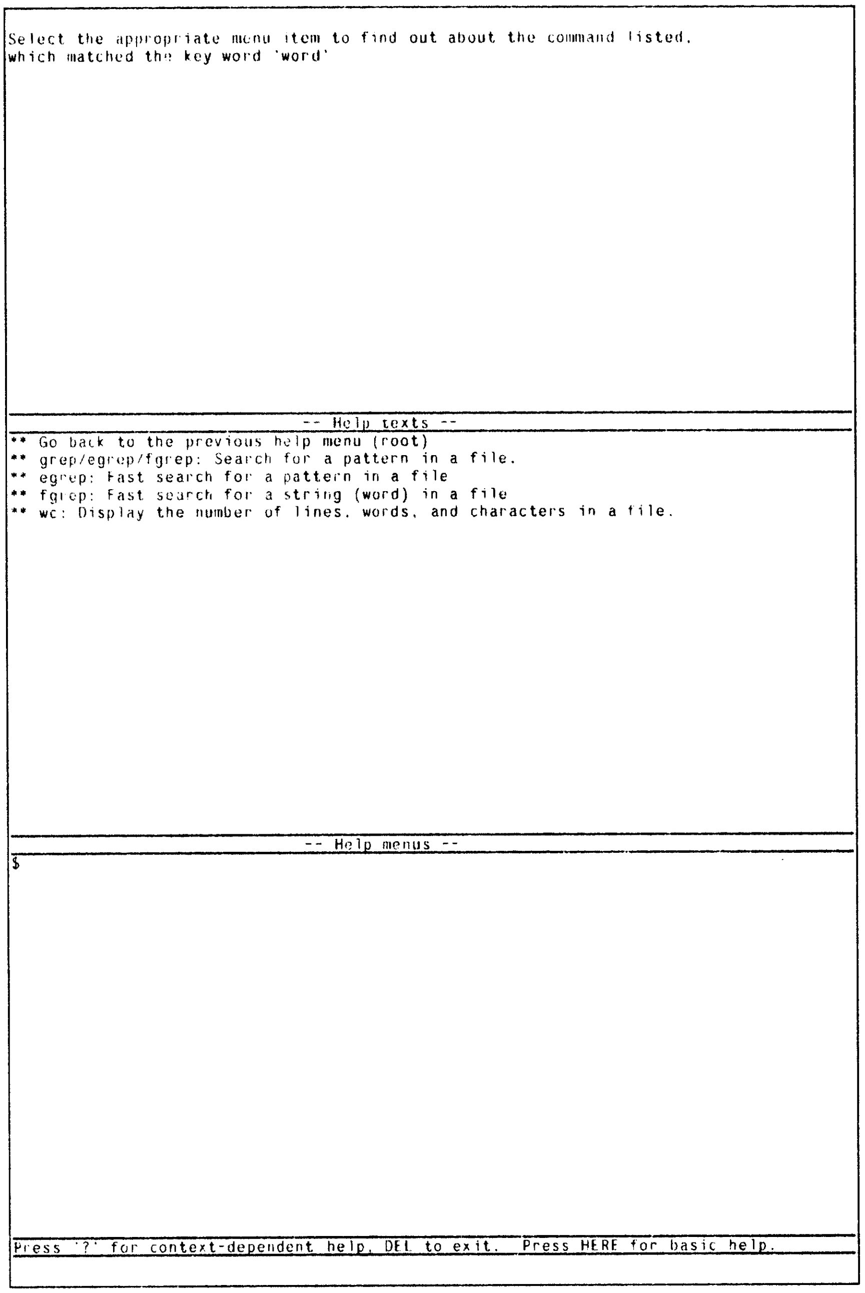
This is ACRONYM's initial screen, which is what the user sees when ACRONYM first starts up. This is also what subjects in the experiments saw when they first started using ACRONYM.

In the example that follows, imagine a user who is trying to scan through a large file looking for occurrences of a single word. He does *not* know about the "grep" utility; this is what he needs to learn from ACRONYM.



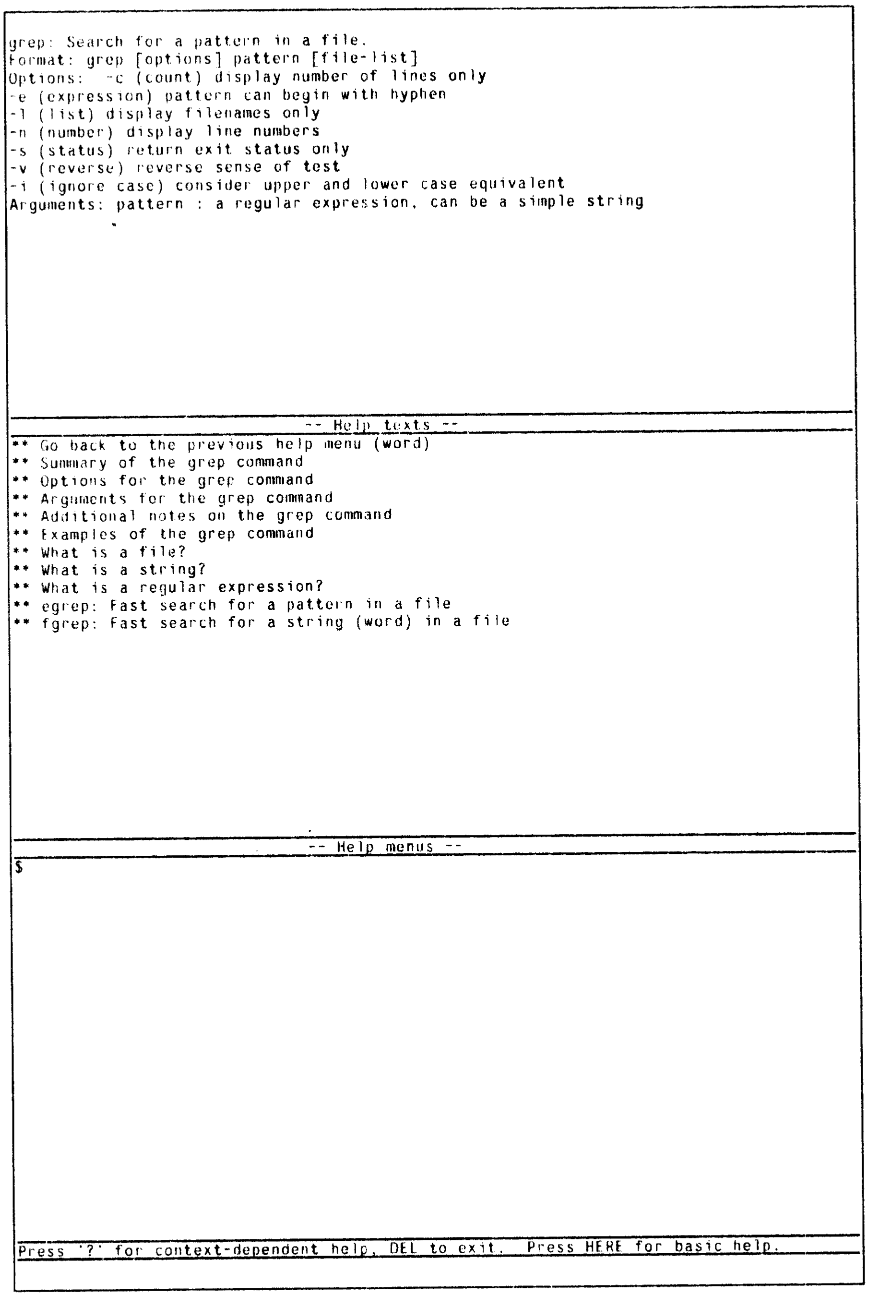
```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is
currently visible, the line at the bottom of the window will offer you a
                              -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** at: Execute a Shell script at a specified time
** bb: print notices from bulletin board(s)
** biff: be notified if mail arrives and who it is from
** cal: display calendar
** calendar: reminder calendar
** cat: display a text file
** cc: C compiler
** ccat: Print compressed files in uncompressed format
** cd or chdir: Change to another working directory
** chmod: Change the access mode of a file
** chat: Communicate with (log in to) another machine on the Ethernet
** ck: check if new mail has arrived
** cmp: Compare two files to see if they differ
** cmuftp: Transfer files to and from other machines on the Ethernet
** col: filter reverse line feeds
** comm: Compare two files and print matching and non-matching lines
** compact: compress files to save space
** cp: Copy file
** cz: convert files to press format and print them on the Dover

                              -- Help menus -- PRESS HERE to move forward.
$ help















Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
Type a key word for which you want help:  word
```

Here the user types "help" to request key word help. ACRONYM, on the very bottom line of the screen, asks him to provide type a key word about which he desires help. The user then types "word".



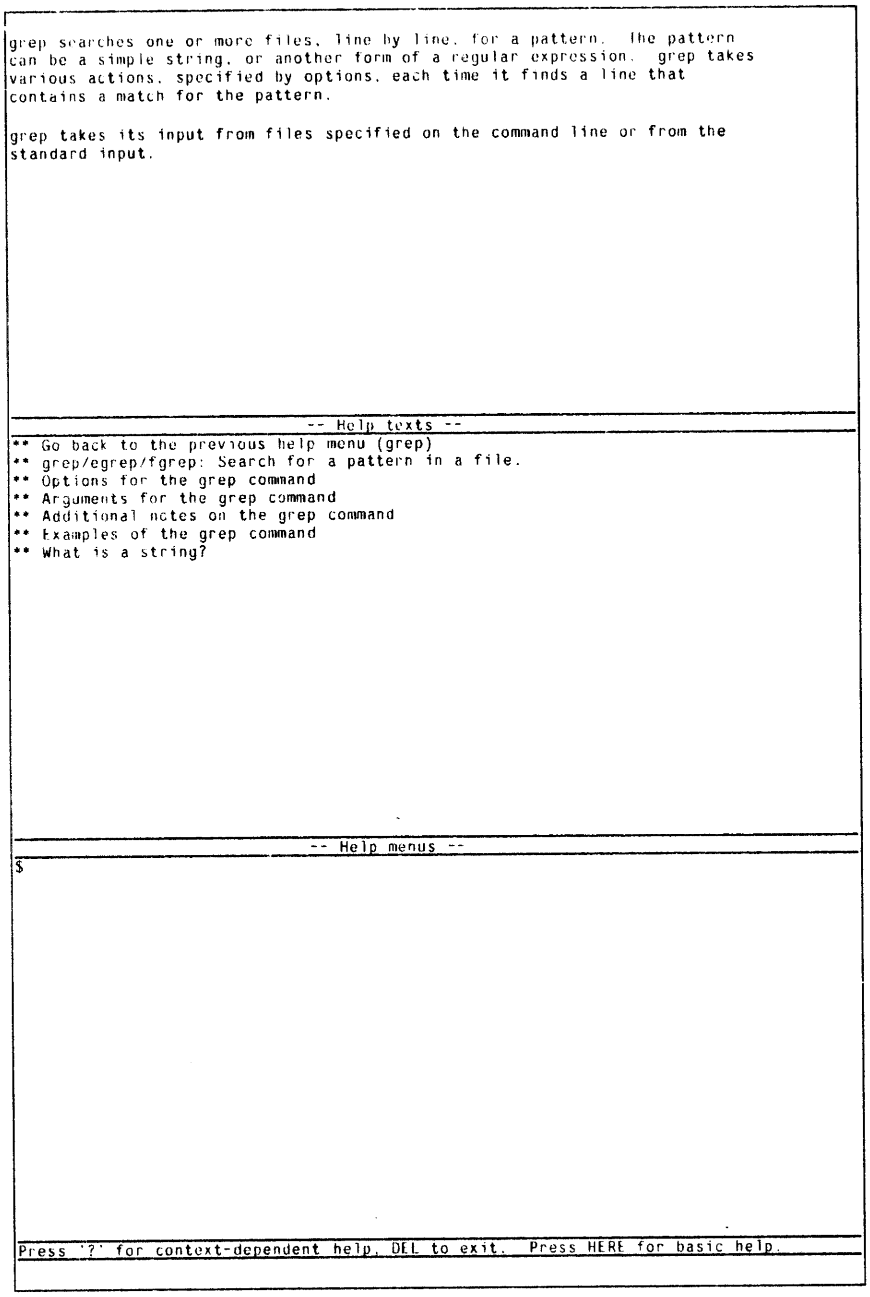
```
Select the appropriate menu item to find out about the command listed,
which matched the key word 'word'

                         -- Help texts --
** Go back to the previous help menu (root)
** grep/egrep/fgrep: Search for a pattern in a file.
** egrep: Fast search for a pattern in a file
** fgrep: Fast search for a string (word) in a file
** wc: Display the number of lines, words, and characters in a file.

                         -- Help menus --
$

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

ACRONYM responds quickly to all help requests. In this case, the key word "word" is ambiguous, in that it matches several help topics. ACRONYM notes the ambiguity in the top window, and provides a menu of choices in the middle window. The user points at the "grep" menu item with a mouse to find out more about the grep family of commands. Note that the word "help" which the user had typed in the third (commands) window has been erased automatically by ACRONYM.



```
grep: Search for a pattern in a file.
Format: grep [options] pattern [file-list]
Options:  -c (count) display number of lines only
-e (expression) pattern can begin with hyphen
-l (list) display filenames only
-n (number) display line numbers
-s (status) return exit status only
-v (reverse) reverse sense of test
-i (ignore case) consider upper and lower case equivalent
Arguments: pattern : a regular expression, can be a simple string

                              -- Help texts --
** Go back to the previous help menu (word)
** Summary of the grep command
** Options for the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Examples of the grep command
** What is a file?
** What is a string?
** What is a regular expression?
** egrep: Fast search for a pattern in a file
** fgrep: Fast search for a string (word) in a file

                              -- Help menus --
$

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

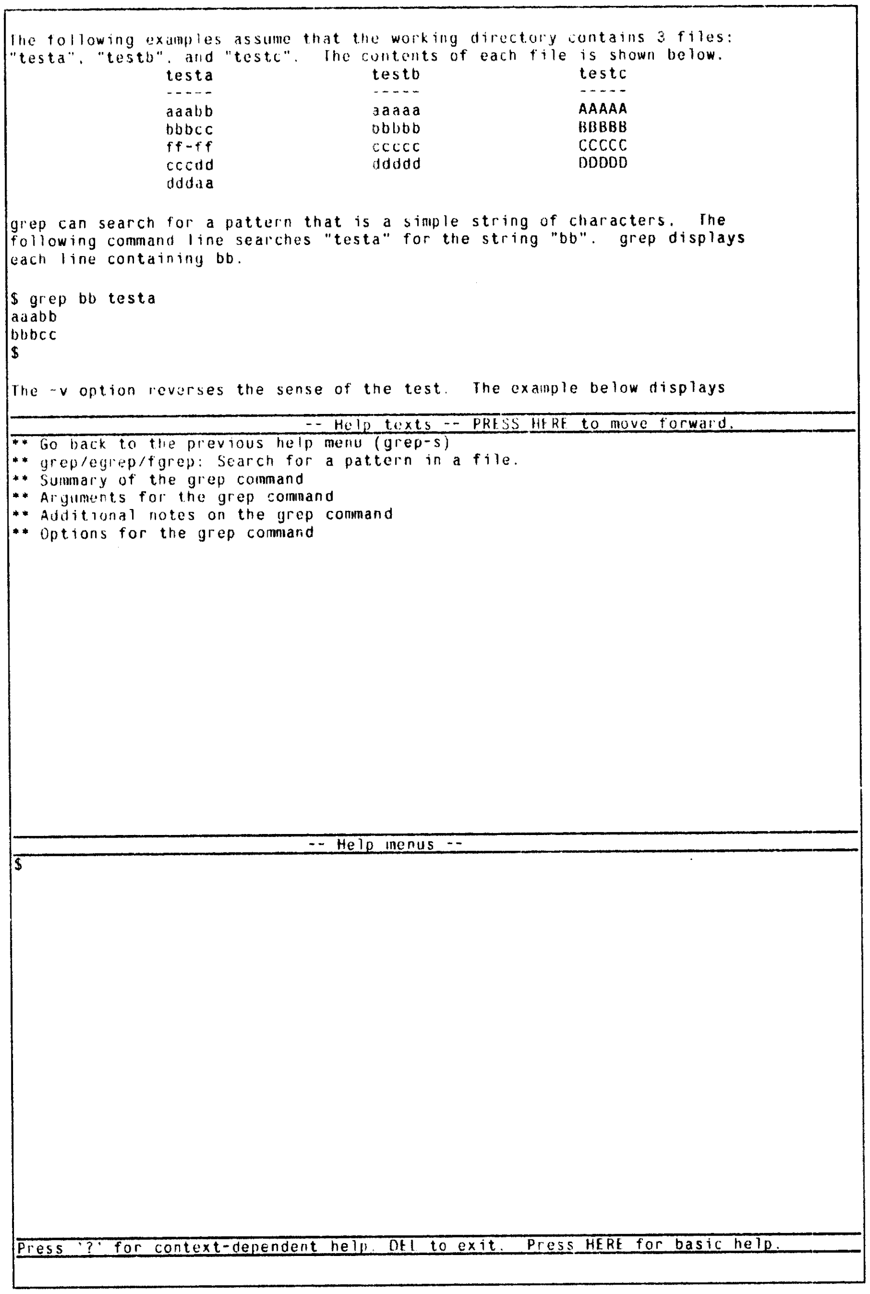
ACRONYM's response to the selection of the "grep" menu item is to update the screen with help about the grep utility. The "basic" help, a brief summary of syntax and options, appears in the top window, while a menu of help messages with further details or dealing with related topics appears in the middle window. In this hypothetical case, in which the user has never used the grep utility before, the basic help does not suffice. Here the user points the mouse at the menu item "Summary of the grep command" for further information.



```
grep searches one or more files, line by line, for a pattern.  The pattern
can be a simple string, or another form of a regular expression.  grep takes
various actions, specified by options, each time it finds a line that
contains a match for the pattern.

grep takes its input from files specified on the command line or from the
standard input.
```

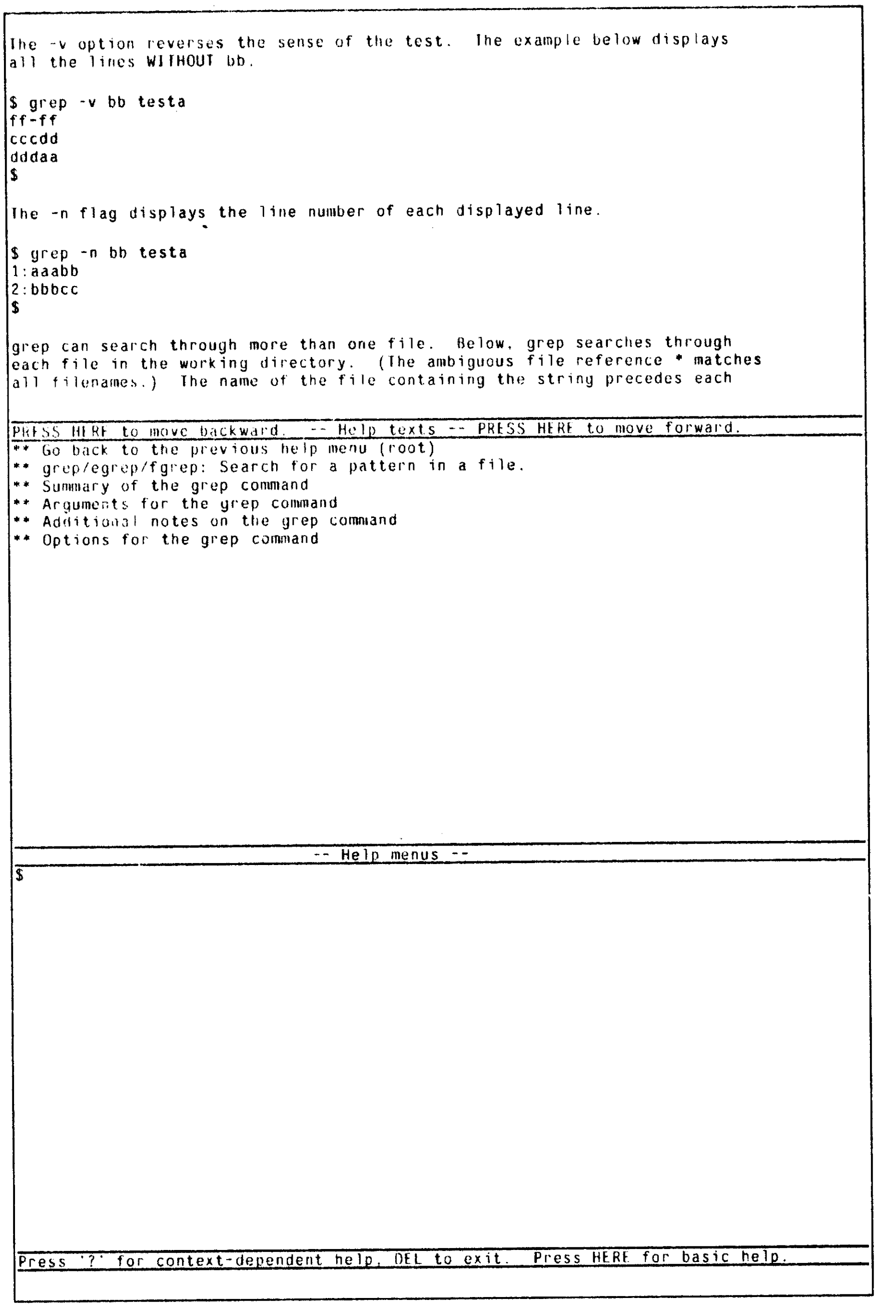
```
                              -- Help texts --
** Go back to the previous help menu (grep)
** grep/egrep/fgrep: Search for a pattern in a file.
** Options for the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Examples of the grep command
** What is a string?
                              -- Help menus --
$
Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here we see the summary information on the grep command, obtained by the previous menu selection of the "Summary of the grep command" menu item. Note that many of the items now in the menu window were also in the menu in the previous two pictures; the help frames for an individual command usually all point at each other. In this case, the user is still uncertain and uses the mouse to select "Examples of the grep command."



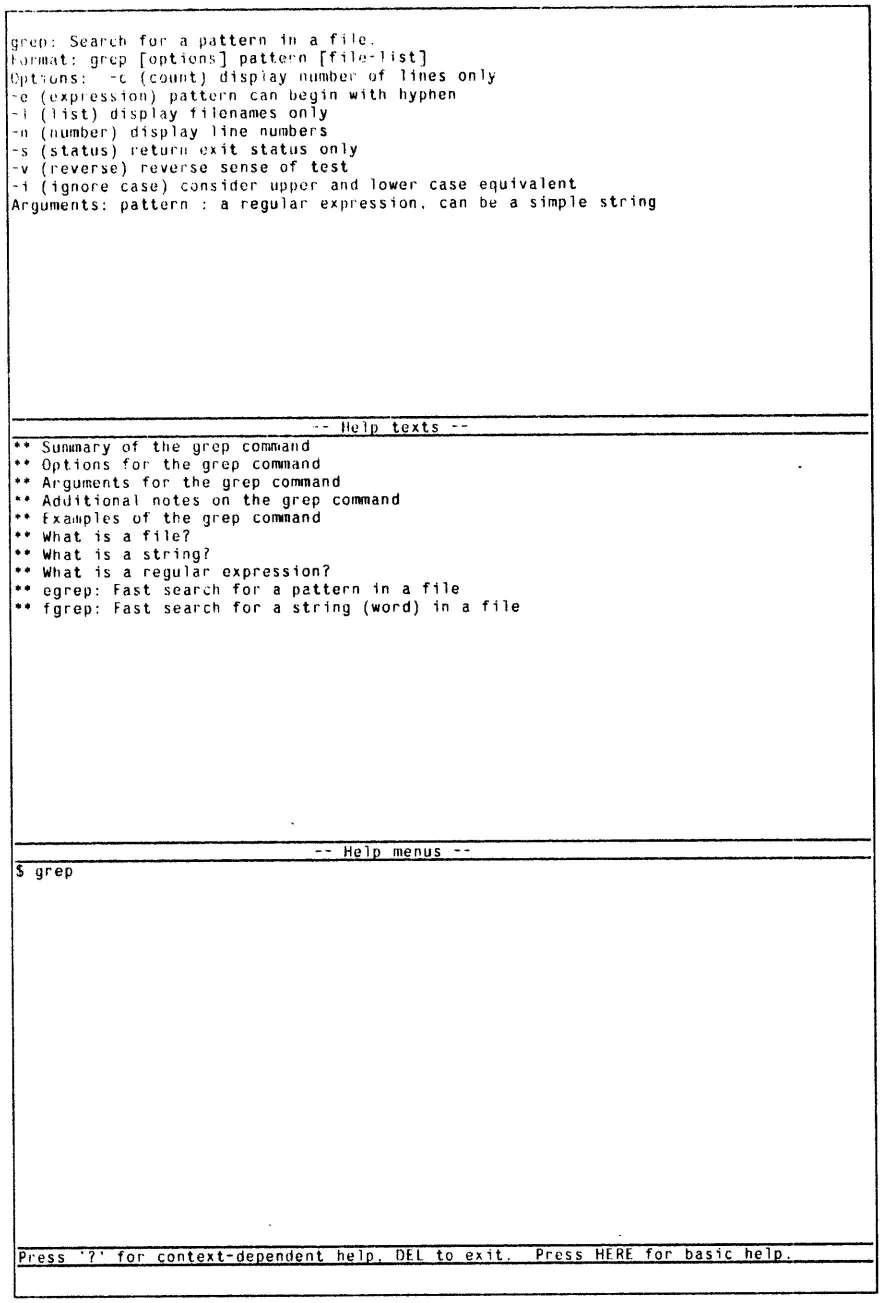
```
The following examples assume that the working directory contains 3 files:
"testa", "testb", and "testc".  The contents of each file is shown below.
                 testa                  testb                  testc
                 -----                  -----                  -----
                 aaabb                  aaaaa                  AAAAA
                 bbbcc                  bbbbb                  BBBBB
                 ff-ff                  ccccc                  CCCCC
                 cccdd                  ddddd                  DDDDD
                 dddaa

grep can search for a pattern that is a simple string of characters.  The
following command line searches "testa" for the string "bb".  grep displays
each line containing bb.

$ grep bb testa
aaabb
bbbcc
$

The -v option reverses the sense of the test.  The example below displays
                                  -- Help texts -- PRESS HERE to move forward.
** Go back to the previous help menu (grep-s)
** grep/egrep/fgrep: Search for a pattern in a file.
** Summary of the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Options for the grep command
                                  -- Help menus --
$
Press '?' for context-dependent help. DEL to exit.  Press HERE for basic help.
```

Here ACRONYM has updated the screen in response to the user's selection of the "Examples" menu item. Note that the examples, in this case, do not entirely fit in the "Help texts" window. Therefore the mode line at the bottom of this window says "Press HERE to move forward." In order to see more of the examples, the user points at the word HERE with the mouse to scroll the help texts forward.



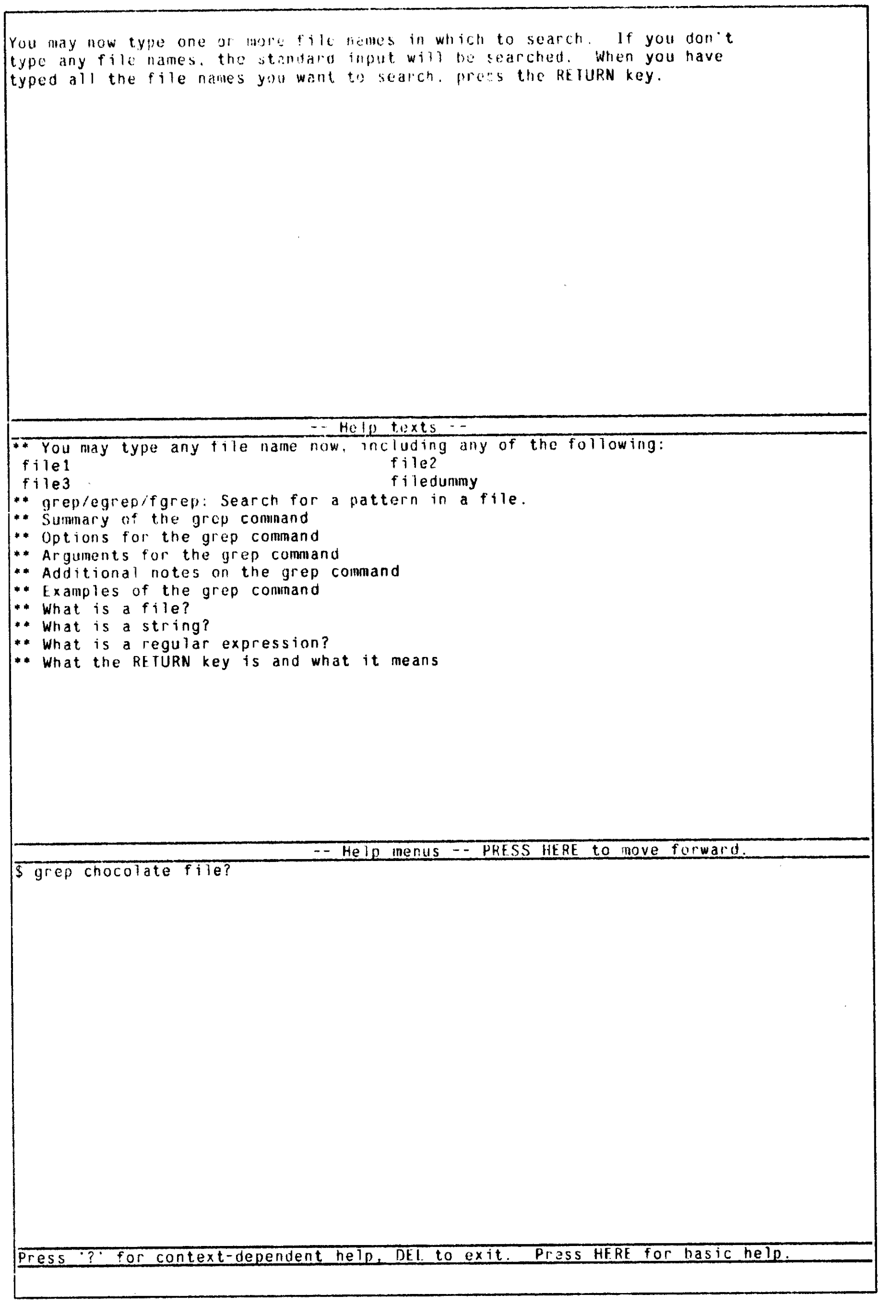
```
The -v option reverses the sense of the test.  The example below displays
all the lines WITHOUT bb.

$ grep -v bb testa
ff-ff
cccdd
dddaa
$

The -n flag displays the line number of each displayed line.

$ grep -n bb testa
1:aaabb
2:bbbcc
$

grep can search through more than one file.  Below, grep searches through
each file in the working directory.  (The ambiguous file reference * matches
all filenames.)  The name of the file containing the string precedes each
PRESS HERE to move backward.  -- Help texts -- PRESS HERE to move forward.
** Go back to the previous help menu (root)
** grep/egrep/fgrep: Search for a pattern in a file.
** Summary of the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Options for the grep command
                              -- Help menus --
$
Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here the top window has been scrolled forward to show more of the examples. Note that the mode line for that window now offers the user the chance to scroll it either forward, to see still more of the examples, or backward, to see what he had been looking at previously (the beginning of the examples). In this case, the user has at last decided that he understands enough to try actually using the grep command.



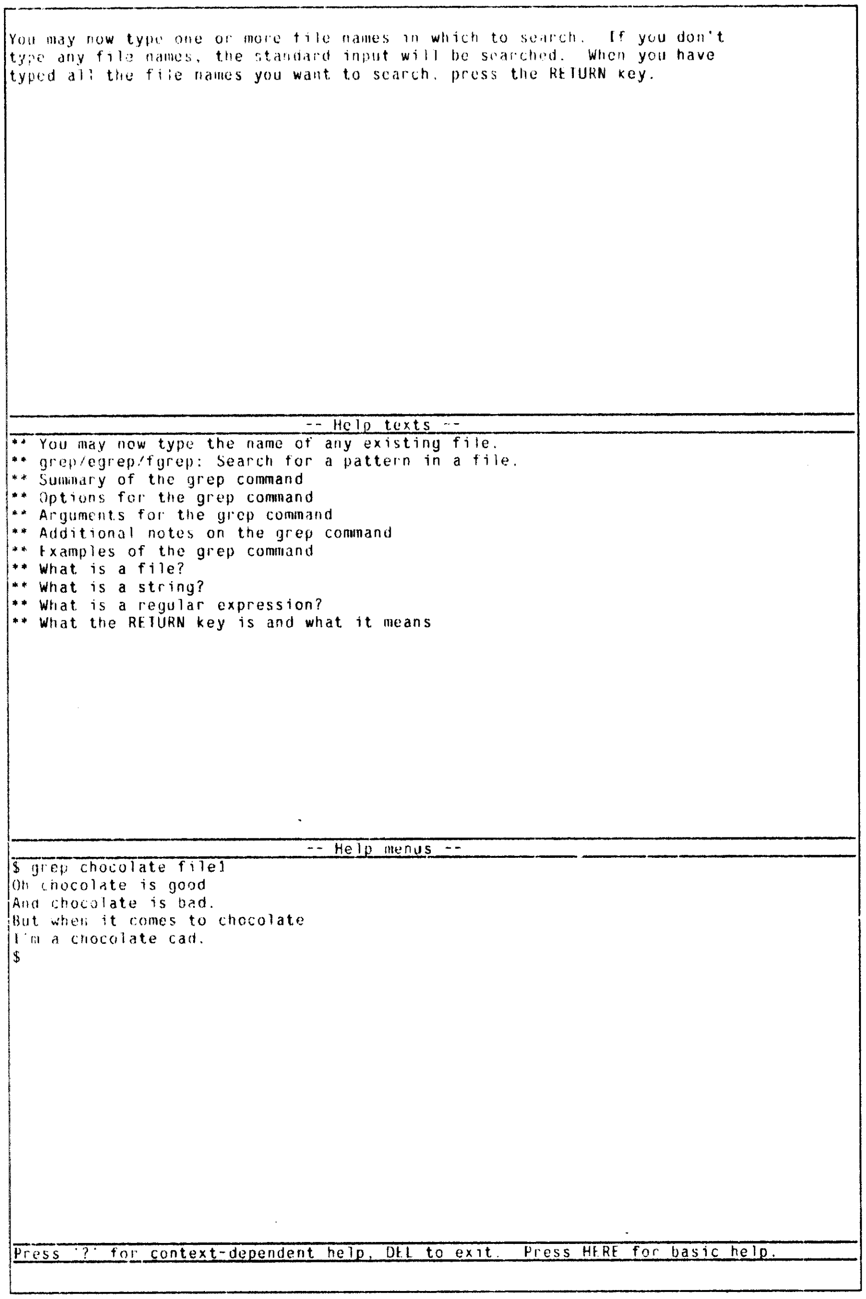
```
grep: Search for a pattern in a file.
Format: grep [options] pattern [file-list]
Options:  -c (count) display number of lines only
-e (expression) pattern can begin with hyphen
-l (list) display filenames only
-n (number) display line numbers
-s (status) return exit status only
-v (reverse) reverse sense of test
-i (ignore case) consider upper and lower case equivalent
Arguments: pattern : a regular expression, can be a simple string

                         -- Help texts --
** Summary of the grep command
** Options for the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Examples of the grep command
** What is a file?
** What is a string?
** What is a regular expression?
** egrep: Fast search for a pattern in a file
** fgrep: Fast search for a string (word) in a file

                         -- Help menus --
$ grep

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here the user has bravely typed "grep" and pressed the SPACE bar. Although the user has not yet typed a complete command, ACRONYM knows that spaces delimit the various parts of UNIX commands. It therefore parses (interprets) the partial command line -- in this case, simply "grep" -- and updates its help accordingly. In this case, that update yields simply the original basic help regarding the "grep" command.



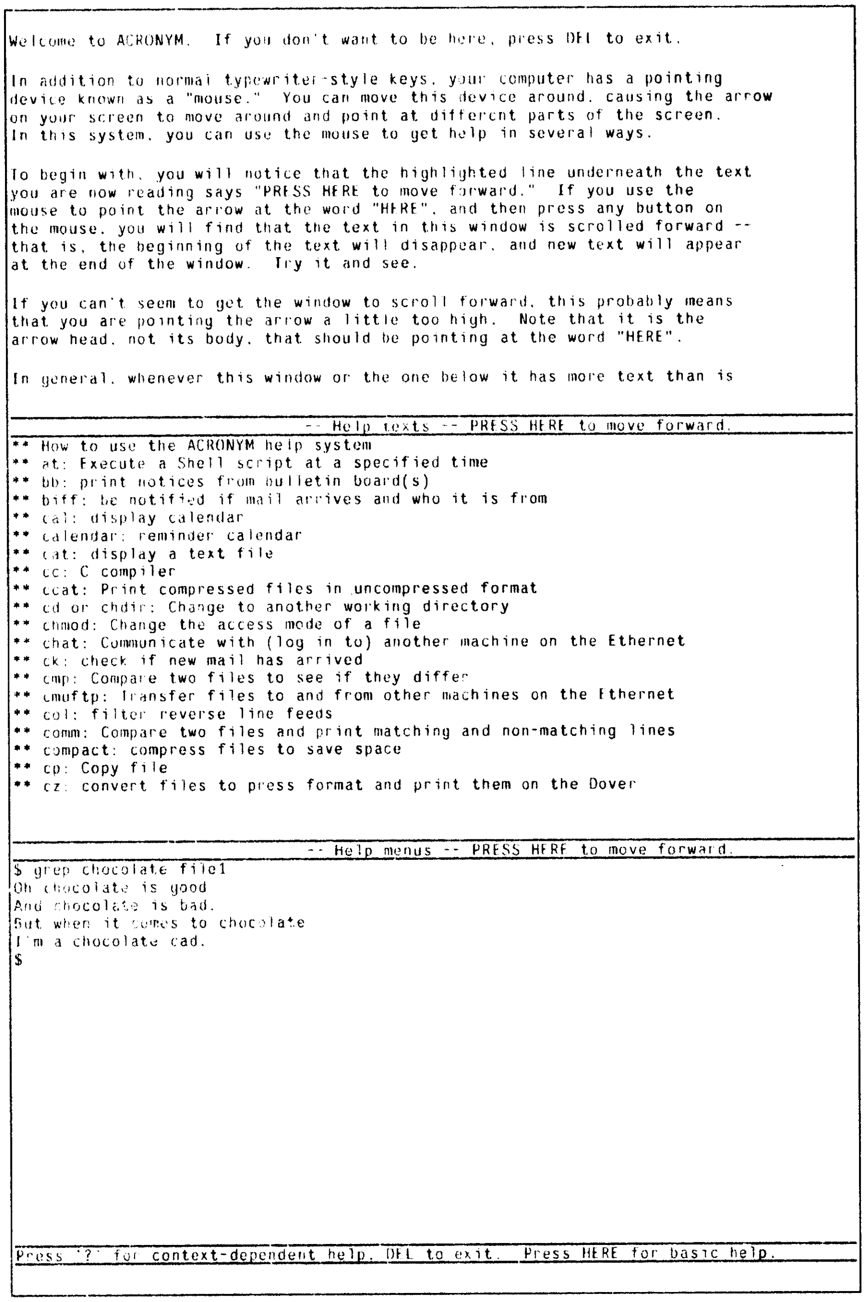
```
You may now type one or more file names in which to search.  If you don't
type any file names, the standard input will be searched.  When you have
typed all the file names you want to search, press the RETURN key.

                         -- Help texts --
** You may type any file name now, including any of the following:
 file1                                   file2
 file3                                   filedummy
** grep/egrep/fgrep: Search for a pattern in a file.
** Summary of the grep command
** Options for the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Examples of the grep command
** What is a file?
** What is a string?
** What is a regular expression?
** What the RETURN key is and what it means

                         -- Help menus -- PRESS HERE to move forward.
$ grep chocolate file?

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here the user has typed more of the grep command. He has typed "chocolate", the word he is looking for, and has begun to type the name of a file. After typing the first four letters, "file", he typed a question mark. ACRONYM then updated the screen as shown above, with a list of possible completions of the file name included in the menu window. After this happens, the question mark is automatically erased by ACRONYM.



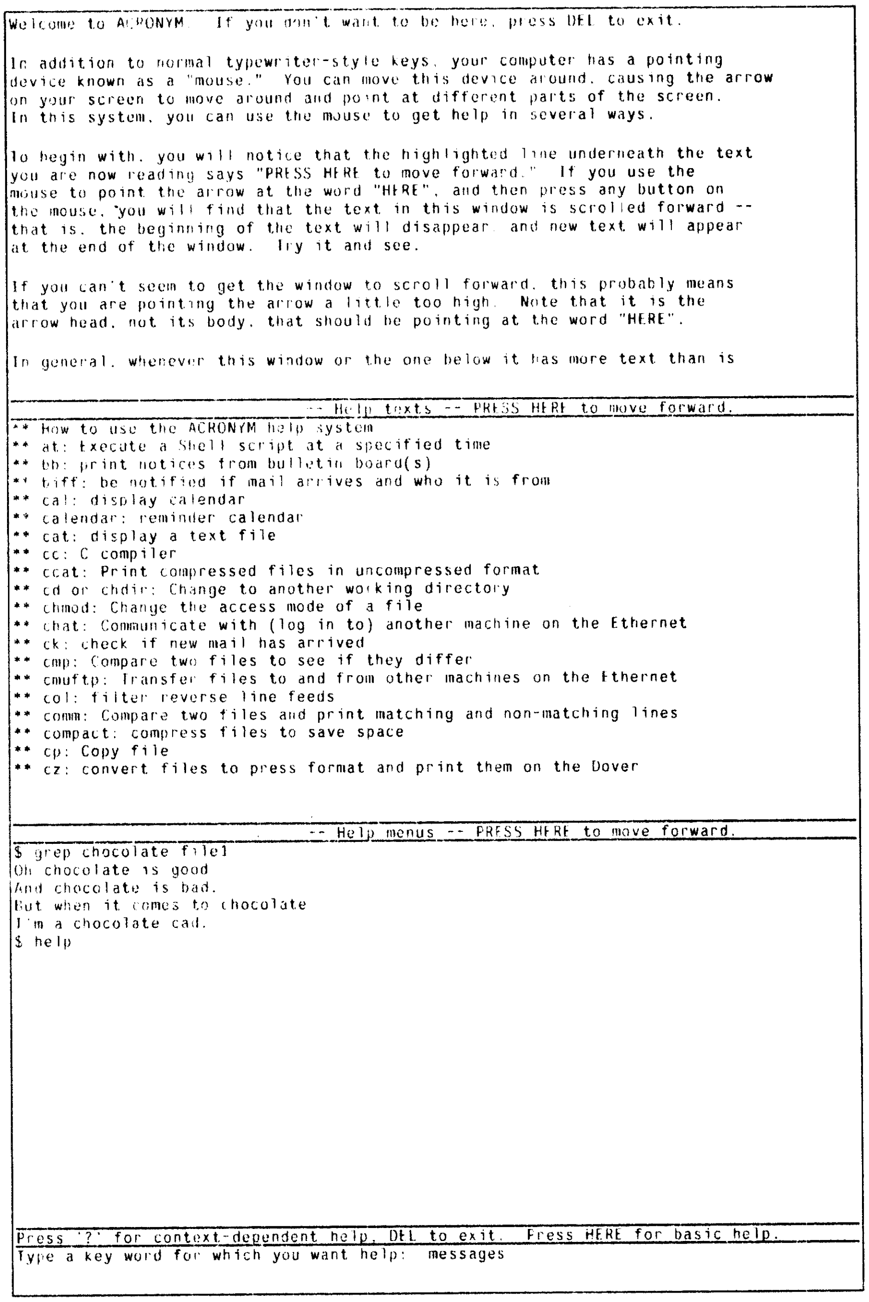
```
You may now type one or more file names in which to search.  If you don't
type any file names, the standard input will be searched.  When you have
typed all the file names you want to search, press the RETURN key.

                              -- Help texts --
** You may now type the name of any existing file.
** grep/egrep/fgrep: Search for a pattern in a file.
** Summary of the grep command
** Options for the grep command
** Arguments for the grep command
** Additional notes on the grep command
** Examples of the grep command
** What is a file?
** What is a string?
** What is a regular expression?
** What the RETURN key is and what it means

                              -- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here we see the completed result of the grep command. The user completed the file name as "file1", and pressed RETURN. ACRONYM then passed the completed command line on to the UNIX shell, which executed the command "grep chocolate file1". The four lines in that file which contained the word "chocolate" were then printed.



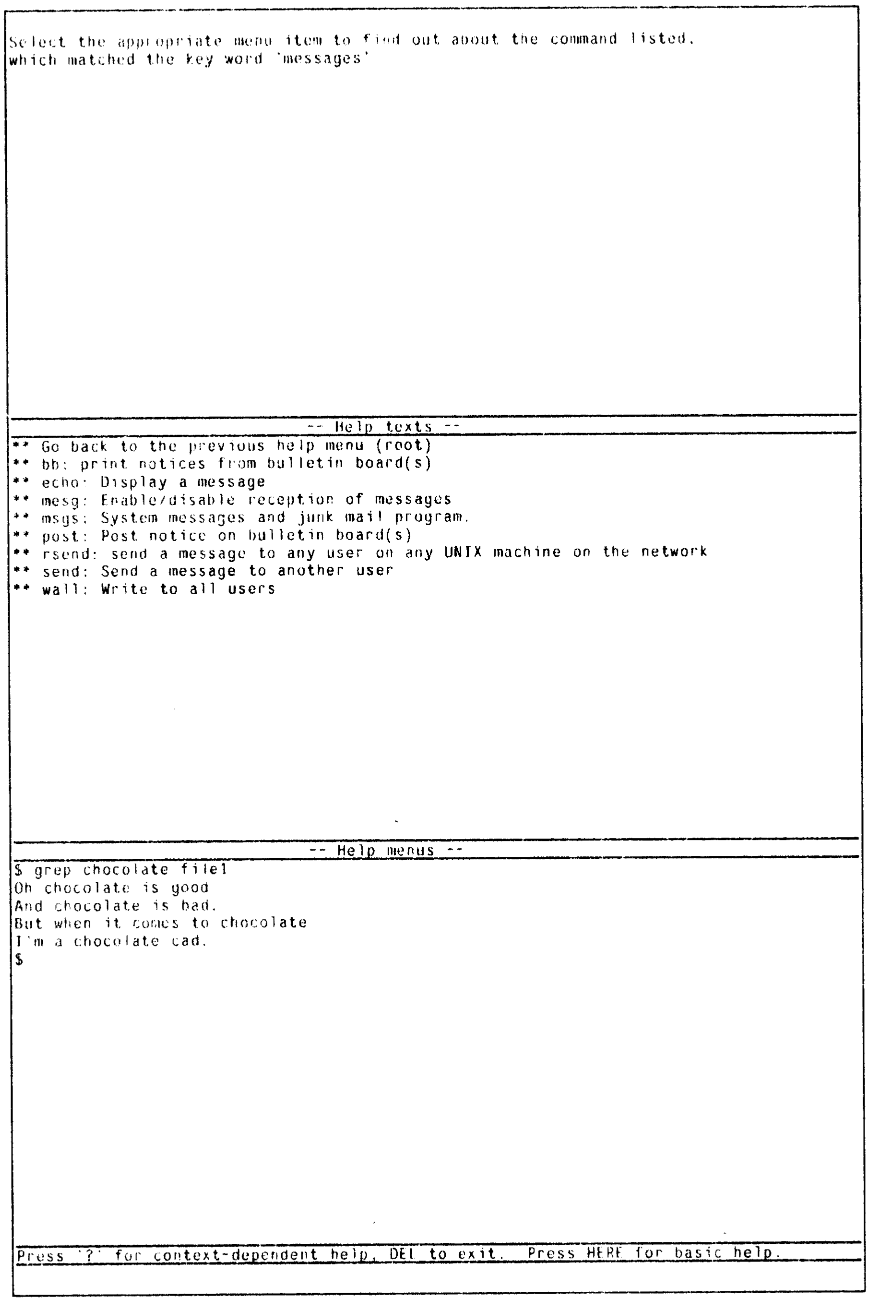
```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is

                                   -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** at: Execute a Shell script at a specified time
** bb: print notices from bulletin board(s)
** biff: be notified if mail arrives and who it is from
** cal: display calendar
** calendar: reminder calendar
** cat: display a text file
** cc: C compiler
** ccat: Print compressed files in uncompressed format
** cd or chdir: Change to another working directory
** chmod: Change the access mode of a file
** chat: Communicate with (log in to) another machine on the Ethernet
** ck: check if new mail has arrived
** cmp: Compare two files to see if they differ
** cmuftp: Transfer files to and from other machines on the Ethernet
** col: filter reverse line feeds
** comm: Compare two files and print matching and non-matching lines
** compact: compress files to save space
** cp: Copy file
** cz: convert files to press format and print them on the Dover

                                   -- Help menus -- PRESS HERE to move forward.
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

After completing a command, ACRONYM generally leaves the help message from that command showing on the screen, as the previous page showed. This is useful when commands are not quite right; often the corrective information is already showing on the screen. However, this means that users must explicitly request a return to ACRONYM's basic help menu. Here, that basic menu has been obtained by typing a question mark before any new command had been typed. It can also be obtained by pointing with the mouse at the "HERE" in "Press HERE for basic help" at the bottom of the screen.



```
Welcome to ACRONYM  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse." You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward." If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is
                                   -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** at: Execute a Shell script at a specified time
** bb: print notices from bulletin board(s)
** biff: be notified if mail arrives and who it is from
** cal: display calendar
** calendar: reminder calendar
** cat: display a text file
** cc: C compiler
** ccat: Print compressed files in uncompressed format
** cd or chdir: Change to another working directory
** chmod: Change the access mode of a file
** chat: Communicate with (log in to) another machine on the Ethernet
** ck: check if new mail has arrived
** cmp: Compare two files to see if they differ
** cmuftp: Transfer files to and from other machines on the Ethernet
** col: filter reverse line feeds
** comm: Compare two files and print matching and non-matching lines
** compact: compress files to save space
** cp: Copy file
** cz: convert files to press format and print them on the Dover

                                   -- Help menus -- PRESS HERE to move forward.
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$ help









Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
Type a key word for which you want help:  messages
```

For a second example, imagine a user trying to learn how to send electronic mail. Beginning with a key word request, the user types "help" and then, when asked for a key word, types "messages".



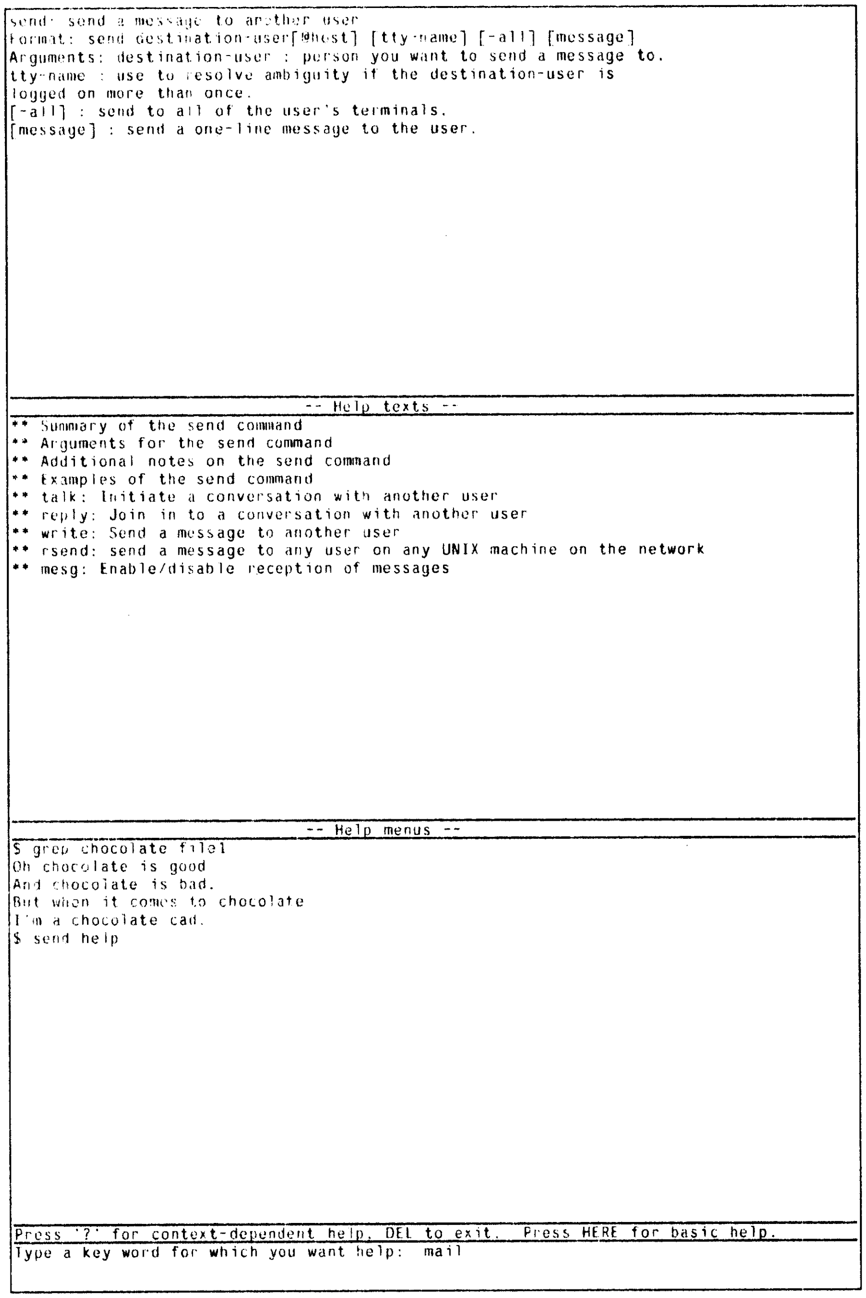
```
Select the appropriate menu item to find out about the command listed,
which matched the key word 'messages'
```

```
                              -- Help texts --
** Go back to the previous help menu (root)
** bb: print notices from bulletin board(s)
** echo: Display a message
** mesg: Enable/disable reception of messages
** msgs: System messages and junk mail program.
** post: Post notice on bulletin board(s)
** rsend: send a message to any user on any UNIX machine on the network
** send: Send a message to another user
** wall: Write to all users
```

```
                              -- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$
```

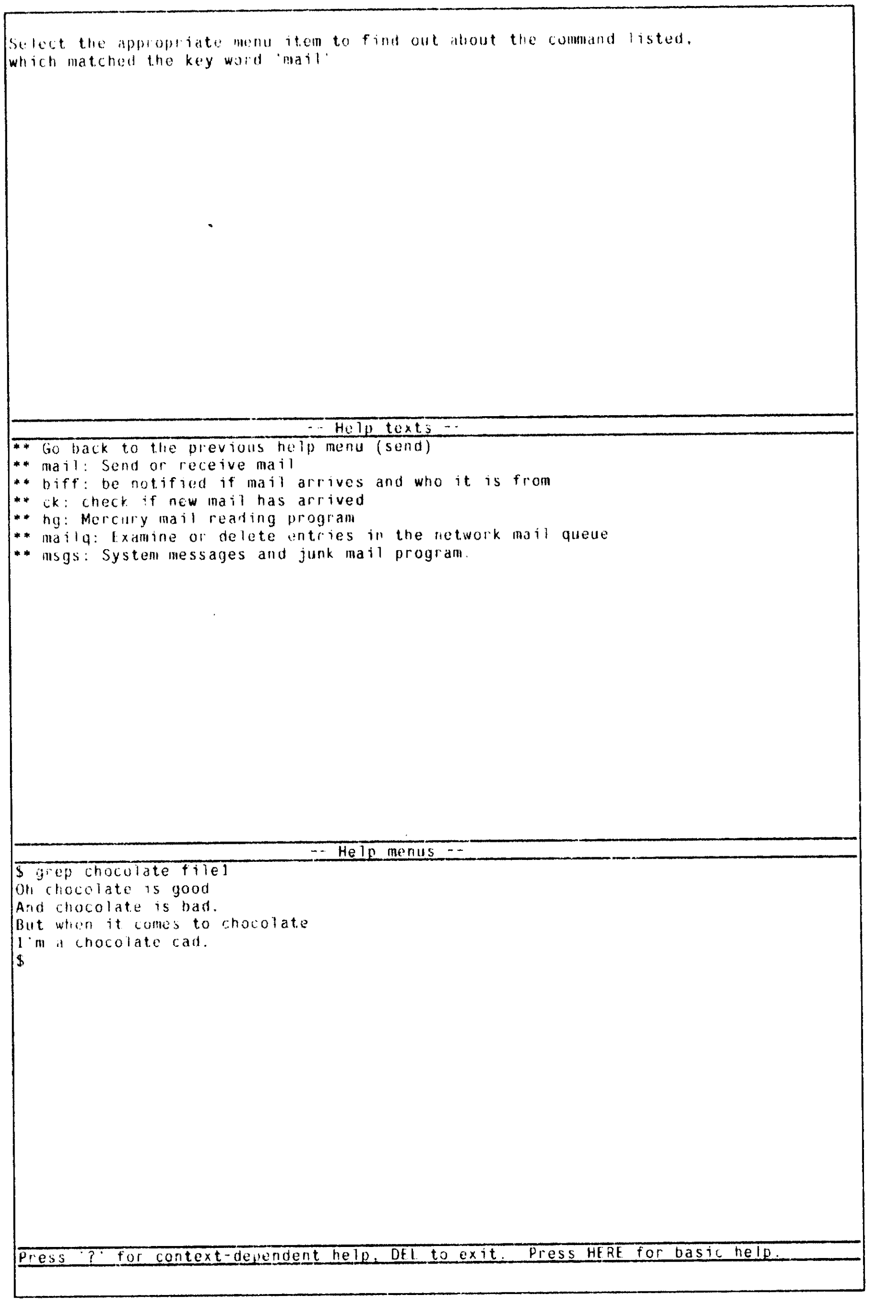
```
Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

In responses to the request for help on the key word "messages", ACRONYM updates the display as shown above. Now, imagine that the user incorrectly decides that the "send" command is what he wants, and begins typing such a command.



```
send: send a message to another user
Format: send destination-user[@host] [tty-name] [-all] [message]
Arguments: destination-user : person you want to send a message to.
tty-name : use to resolve ambiguity if the destination-user is
logged on more than once.
[-all] : send to all of the user's terminals.
[message] : send a one-line message to the user.

                          -- Help texts --
** Summary of the send command
** Arguments for the send command
** Additional notes on the send command
** Examples of the send command
** talk: Initiate a conversation with another user
** reply: Join in to a conversation with another user
** write: Send a message to another user
** rsend: send a message to any user on any UNIX machine on the network
** mesg: Enable/disable reception of messages

                          -- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$ send help

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
Type a key word for which you want help:  mail
```

Here the user has typed "send" and pressed SPACE, and ACRONYM has updated its help to the basic help for the send utility. From this help, the user can tell he's got the wrong command, and he tries another key word help request, this time using the key word "mail".



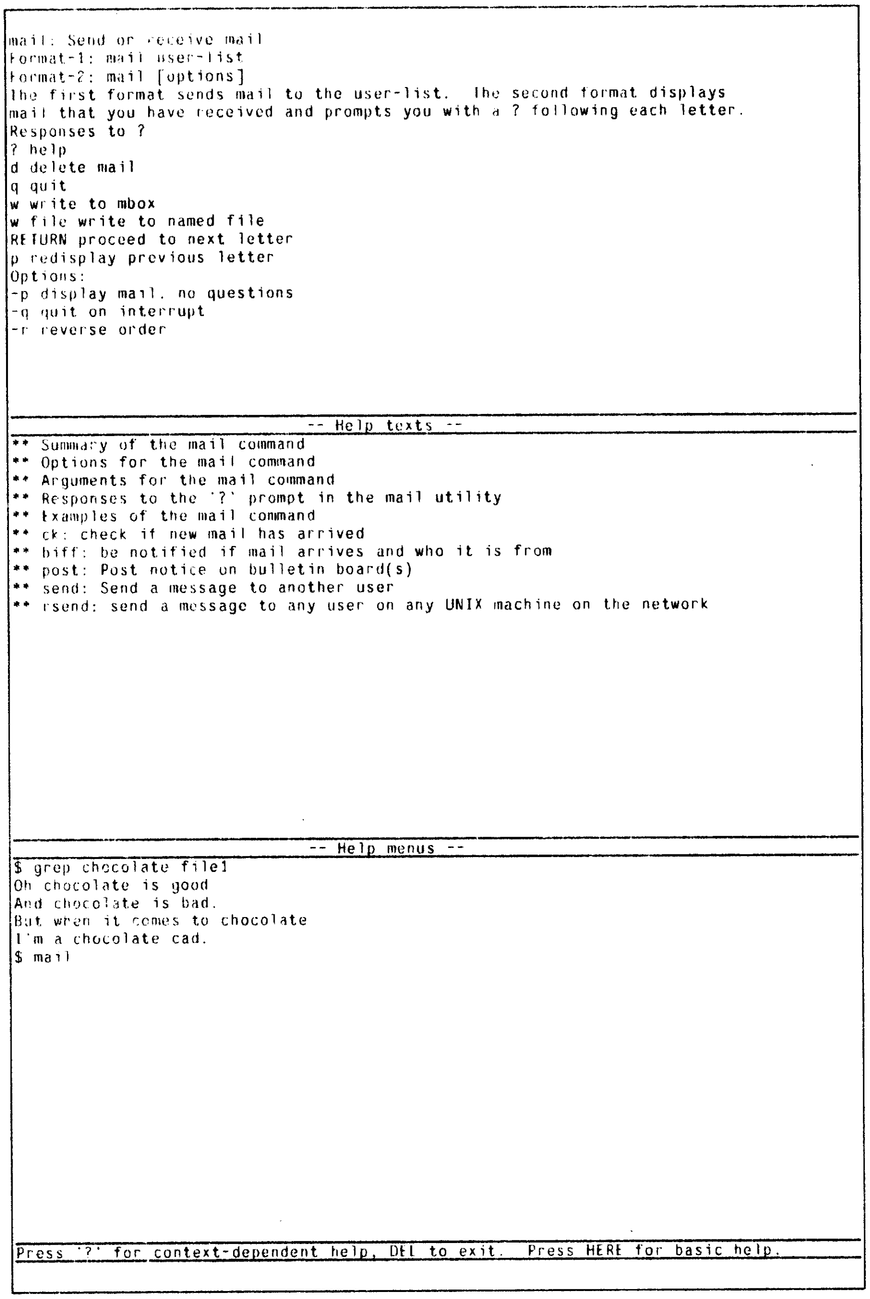
```
Select the appropriate menu item to find out about the command listed,
which matched the key word 'mail'
```

```
-- Help texts --
** Go back to the previous help menu (send)
** mail: Send or receive mail
** biff: be notified if mail arrives and who it is from
** ck: check if new mail has arrived
** hg: Mercury mail reading program
** mailq: Examine or delete entries in the network mail queue
** msgs: System messages and junk mail program.
```

```
-- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$
```

```
Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here the user has gotten help on the key word mail, and the menu makes it fairly obvious that what he wants is the "mail" command. Note that he must himself explicitly erase the erroneous "send" command he had already typed; in this picture, he has already done so.



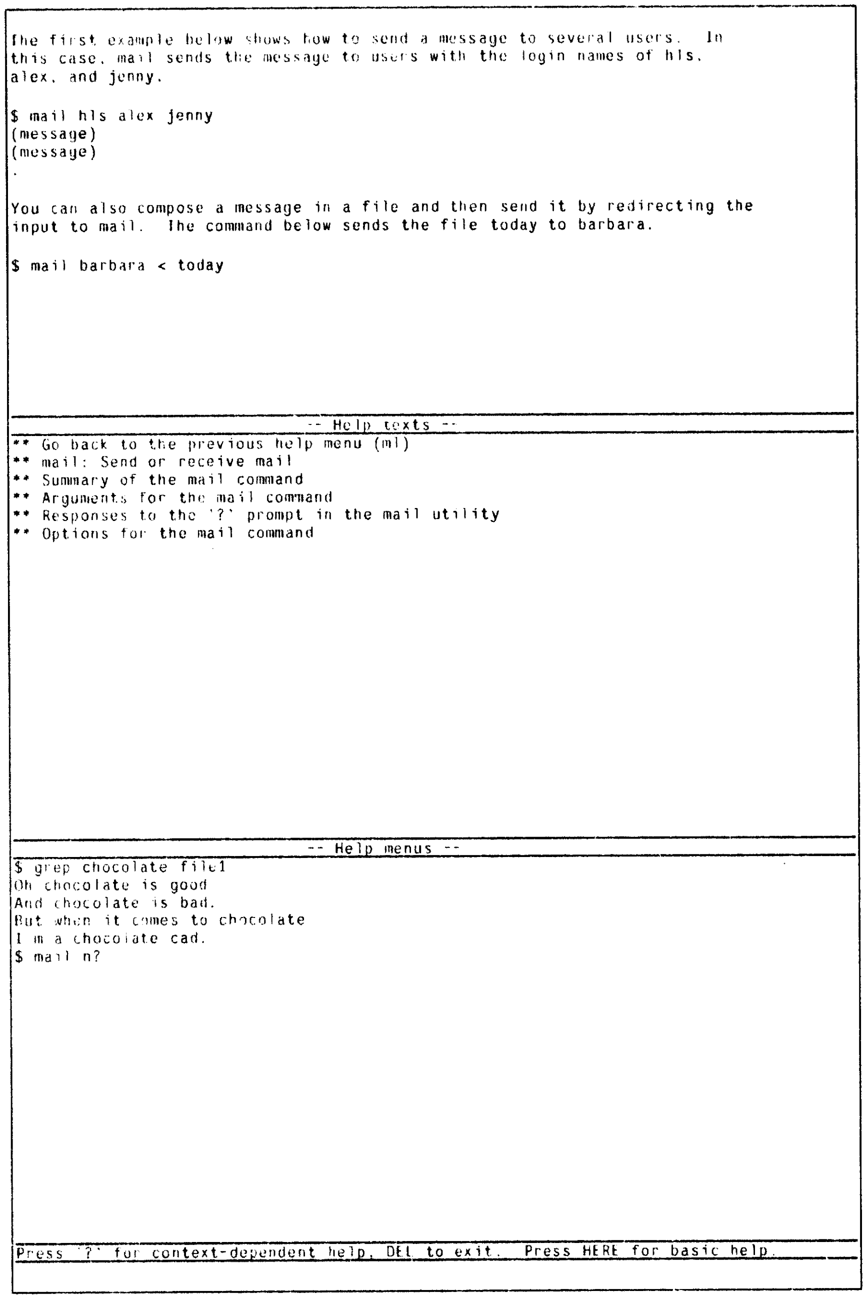
```
mail: Send or receive mail
Format-1: mail user-list
Format-2: mail [options]
The first format sends mail to the user-list.  The second format displays
mail that you have received and prompts you with a ? following each letter.
Responses to ?
? help
d delete mail
q quit
w write to mbox
w file write to named file
RETURN proceed to next letter
p redisplay previous letter
Options:
-p display mail, no questions
-q quit on interrupt
-r reverse order
```

```
                         -- Help texts --
** Summary of the mail command
** Options for the mail command
** Arguments for the mail command
** Responses to the '?' prompt in the mail utility
** Examples of the mail command
** ck: check if new mail has arrived
** biff: be notified if mail arrives and who it is from
** post: Post notice on bulletin board(s)
** send: Send a message to another user
** rsend: send a message to any user on any UNIX machine on the network
```

```
                         -- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$ mail
```

```
Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here the user has typed "mail", as the first part of his mail command, and ACRONYM has updated the help windows accordingly. Not sure what to do next, the user selects (with the mouse) the menu item entitled "Examples of the mail command".



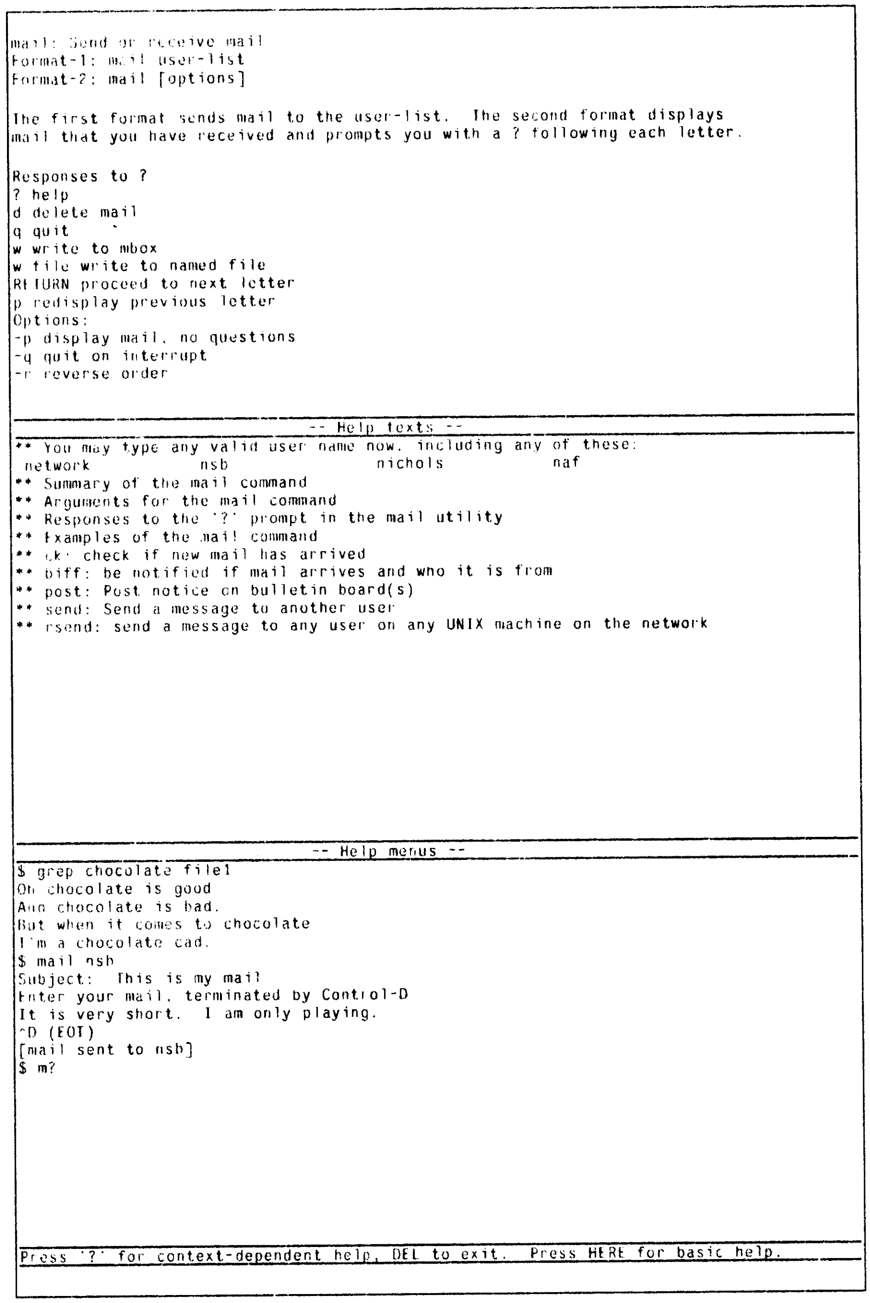
```
The first example below shows how to send a message to several users.  In
this case, mail sends the message to users with the login names of hls,
alex, and jenny.

$ mail hls alex jenny
(message)
(message)
.

You can also compose a message in a file and then send it by redirecting the
input to mail.  The command below sends the file today to barbara.

$ mail barbara < today
```

```
                              -- Help texts --
** Go back to the previous help menu (ml)
** mail: Send or receive mail
** Summary of the mail command
** Arguments for the mail command
** Responses to the '?' prompt in the mail utility
** Options for the mail command
```

```
                              -- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I m a chocolate cad.
$ mail n?
```

```
Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

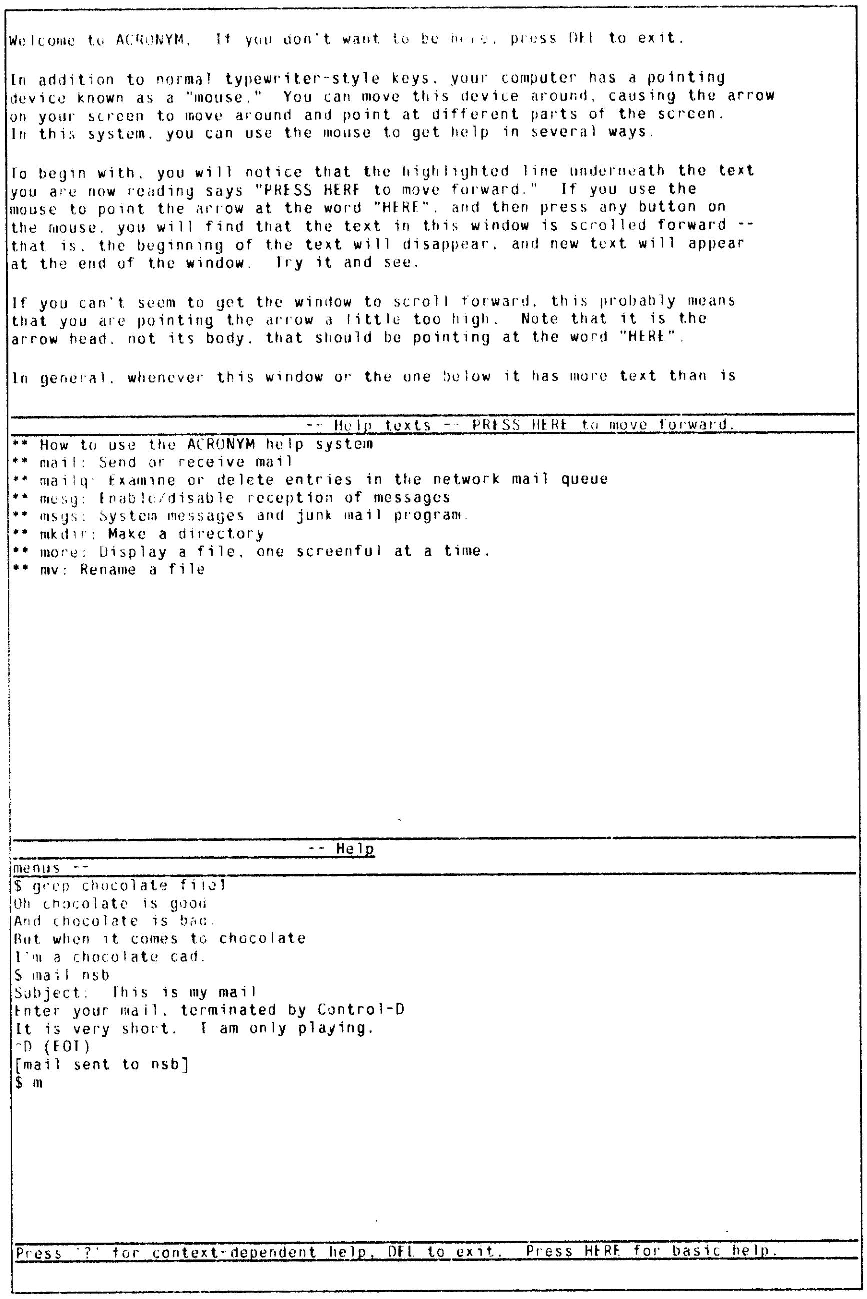
This picture shows the screen after ACRONYM has responded to the previous request for "Examples of the mail command." From the example, the user can tell that what he needs to do is to type the name of the recipient of the mail. He then types "n", the first letter of the name of the person to whom he is sending mail, but then is unsure about the name's spelling. By typing a question mark, he requests further help, including possible completions of the word he is currently typing.



```
mail: Send or receive mail
Format-1: mail user-list
Format-2: mail [options]

The first format sends mail to the user-list.  The second format displays
mail that you have received and prompts you with a ? following each letter.

Responses to ?
? help
d delete mail
q quit
w write to mbox
w file write to named file
RETURN proceed to next letter
p redisplay previous letter
Options:
-p display mail, no questions
-q quit on interrupt
-r reverse order

                              -- Help texts --
** You may type any valid user name now, including any of these:
 network            nsb                nichols            naf
** Summary of the mail command
** Arguments for the mail command
** Responses to the '?' prompt in the mail utility
** Examples of the mail command
** ck: check if new mail has arrived
** biff: be notified if mail arrives and who it is from
** post: Post notice on bulletin board(s)
** send: Send a message to another user
** rsend: send a message to any user on any UNIX machine on the network

                              -- Help menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$ mail nsb
Subject:  This is my mail
Enter your mail, terminated by Control-D
It is very short.  I am only playing.
^D (EOT)
[mail sent to nsb]
$ m?

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

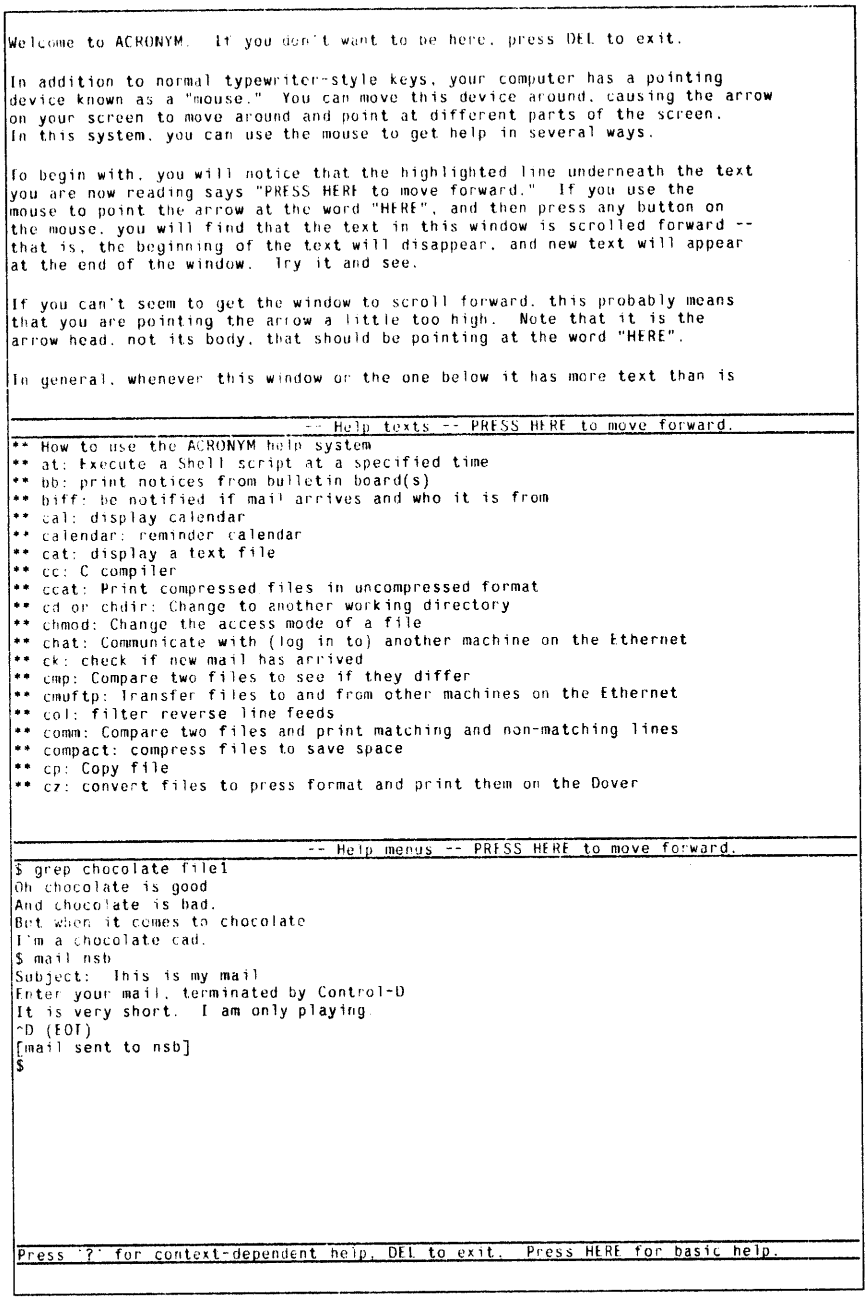
In this picture, we see the final help screen from the mail command. The menu window includes a list of valid mail recipients that start with "n", in response to the user's typing a question mark after only typing "n" for a user name. Once he had the spelling right, the user completed the interaction with the mail command in the usual way, with the results shown here.

Now, just for fun, the user types only an "m" on a command line and then types a question mark.



```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is

                                   -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** mail: Send or receive mail
** mailq: Examine or delete entries in the network mail queue
** mesg: Enable/disable reception of messages
** msgs: System messages and junk mail program.
** mkdir: Make a directory
** more: Display a file, one screenful at a time.
** mv: Rename a file

                                   -- Help
menus --
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad
But when it comes to chocolate
I'm a chocolate cad.
$ mail nsb
Subject:  This is my mail
Enter your mail, terminated by Control-D
It is very short.  I am only playing.
^D (EOT)
[mail sent to nsb]
$ m

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here, in response to an "m" followed by a question mark, ACRONYM has retained the standard help text, but has updated it menu to eliminate those commands which do not start with "m".



```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "PRESS HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is

                          -- Help texts -- PRESS HERE to move forward.
** How to use the ACRONYM help system
** at: Execute a Shell script at a specified time
** bb: print notices from bulletin board(s)
** biff: be notified if mail arrives and who it is from
** cal: display calendar
** calendar: reminder calendar
** cat: display a text file
** cc: C compiler
** ccat: Print compressed files in uncompressed format
** cd or chdir: Change to another working directory
** chmod: Change the access mode of a file
** chat: Communicate with (log in to) another machine on the Ethernet
** ck: check if new mail has arrived
** cmp: Compare two files to see if they differ
** cmuftp: Transfer files to and from other machines on the Ethernet
** col: filter reverse line feeds
** comm: Compare two files and print matching and non-matching lines
** compact: compress files to save space
** cp: Copy file
** cz: convert files to press format and print them on the Dover

                          -- Help menus -- PRESS HERE to move forward.
$ grep chocolate file1
Oh chocolate is good
And chocolate is bad.
But when it comes to chocolate
I'm a chocolate cad.
$ mail nsb
Subject:  This is my mail
Enter your mail, terminated by Control-D
It is very short.  I am only playing.
^D (EOT)
[mail sent to nsb]
$

Press '?' for context-dependent help, DEL to exit.  Press HERE for basic help.
```

Here the user has erased the "m" he typed, and returned to ACRONYM's basic help. Though somewhat artificial, the examples in this chapter have at least demonstrated the important mechansisms ACRONYM uses to provide help to its users. Further information, including a videotape of the system in use, is available from the author.

Appendix C
The Implementation of ACRONYM

In this appendix, a few raw files from ACRONYM's database are presented in order to convey the flavor of the implementation. That flavor, incidentally, is not terribly tasty; ACRONYM was designed and implemented in a hurry, with reliability and structural integrity entirely ignored when I thought I could save myself a few hours. It is straightforward to see that a more "correct" implementation could produce equal or better results.

ACRONYM's view of the world is that it is a great big network of information. That network has a starting node, called the root, and two distinct types of links between nodes. Those links may be referred to as *syntactic* and *semantic* links.

A *syntactic* link specifies a transition in the current world-state which is made on the basis of a partial command from the user. Thus, for example, when the user types "ls" from the root state, a transition is made to the state corresponding to a partial "ls" command. Naturally, the help provided in this state is specific help for the ls utility.

A *semantic* link specifies a topical connection between two information nodes. Thus, if the user is currently in the "del" state, a semantic link to the "lsd" command ensures that one item in the help menu will be a pointer to the "lsd" command. However, there is nothing that the user can type in the current state that will move him directly into the "lsd" state. (Of course, he can erase the "del" command and type "lsd", but these are two separate operations.)

In ACRONYM, nodes can be linked either just semantically, just syntactically, or both. The connections are specified in a clumsy but workable language designed for the purpose.

Each node in the network must be given a unique name by the database designer. That name, for reasons related to UNIX file naming conventions, can be at most nine letters long. For each node, the database designer has to crate two files, the "help" and "hcom" files.

The "help" file is just a file in the database directory, named by the node name followed by ".help", which contains the help text exactly as it will appear in the help text window. The ".hcom" file, named by the node name followed by ".hcom", contains a description of links to and from the node.

When the ACRONYM database is compiled, ACRONYM reads in *all* of the hcom files and then produces, for each node, a third file called the "hmen" file. This file, which is named by the node name followed by ".hmen", is the menu of related topics which will be shown to the user when he is seeing the associated ".help" file. The file also contains some pointer information which will be hidden from the user. In general, ACRONYM menus come from these pre-compiled hmen files, although in certain circumstances (those involving dynamic operations such as file name or user name completion), ACRONYM will construct such menus "on the fly".

The language which is used in the hcom files to specify the structure of the database is simple but ugly. The hcom file consists of two parts: the first line is the key line, a short description of the node which will be used in the construction of menus which point *to* that node. This first line might best be thought of as a "long name" for the node, where the actual node name is limited to nine characters because of the file naming conventions.

After that first line, the remainder of an hcom file consists simply of a series of paired words specifying links. In each pair of words, the first word specifies the nature of the link, while the second word specifies the node being linked to. By default, all links are both semantic and syntactic, with the first word interpreted as a description of what the user has to type to make a syntactic transition. Thus, the line which appears in the root hcom file ("root.hcom"), "grep grep" specifies that if, in the root state, a user types "grep", then ACRONYM's world state should be changed to the "grep" node. It also specifies that the root menu includes an entry for grep (this is the semantic link).

In a first version of ACRONYM, the syntactic links could be specified with arbitrary regular expressions; hence a line in an hcom file of the form "xy.* foo" would allow a syntactic transition to node "foo" if the user typed any word beginning with "xy". This mechanism proved unnecessary, and was removed for efficiency considerations. In its place, however, a few special links were created. If the link is specified as "@file", then the transition is made when any file name is typed. Similarly, "@user" allows transitions when user names are typed, "@dir" allows transitions when directory names are typed, and "@filewrite" allows transitions when any potentially valid (writable) file names are typed. Finally, to facilitate a few special cases of the UNIX command language without regular expression parsing, "@opt" matches any string starting with a dash ("-"), and "@any" matches any single word.

All of these mechanisms by default create both syntactic and semantic links; the syntactic links permit state transitions in the parsing process, while the semantic links mandate menu entries. Syntactic-only links are created by preceding the name of the node being linked to with an "@" sign. Thus an hcom line like "@file @foo" specifies a syntactic link to node "foo" whenever a file name is typed, but does not create any entries in the associated help menu.

Semantic only forward links are created via the "@link" link specification. Thus a line in an hcom file of the form "@link foo" creates a menu link to node foo, but does not permit any state transitions during parsing. This is used for concept-related links, as in "examples", "summary", related commands, and so on.

For convenience, the semantic links can be specified in either direction. Thus, if the file "foo.hcom" contains the line "@link bar", this creates a menu item in foo's help menu which refers to the node "bar". However, the "@key" link mechanism can be used to specify links in the other direction; a line in "foo.hcom" that says "@key bar" creates a link from bar's help menu to the node foo. (The word "key" here is intended to suggest key words; in this example, we are saying in effect that bar is a key word for foo, so that people seeing help for bar should also be pointed at foo.)

Such are the mechanisms involved in ACRONYM's database. The deeply interested reader will find, in the rest of this appendix, all of the help, hcom, and hmen files that were used in the examples shown in Appendix B. By consulting these file listings while reading that appendix, it should be possible to obtain a clearer picture of what ACRONYM was actually doing in the examples presented there.

It will be helpful, in reading the hmen files, to note that the first 17 characters of each line of the hmen file is hidden from the user's view when ACRONYM is in use. (Actually, only 14 characters are hidden in the function key version, but 17 are hidden in the mouse version, which is what Appendix B simulates.) The first 14 characters specify the name of the node to which the menu refers, while the next three characters offer a function key to be used to select that menu item, for versions of ACRONYM which use function keys.

One final note about ACRONYM worth knowing is that an alias mechanism is provided to facilitate processing of key word synonyms and of key words longer than the nine letter maximum length for node names. A file named "ALIASES" provides a set of aliases, such as "privileges prot" to indicate that a key word request for "privileges" will point to the node named "prot". In the examples given here, the only time this mechanism is used is for the key word "messages". This is aliased to the key word "message" in the ALIAS file. Thus, the relevant file names are names such as "message.hcom" instead of "messages.hcom".

The files are listed in alphabetical order. All files used in the examples in Appendix B are included, and no others. Of course, other hcom files were used in the compilation process that produced the hmen files, using the @key mechanism. However, no other hmen or help files were used.

grep.hcom

```
grep/egrep/fgrep: Search for a pattern in a file.
@opt grep
@any @grep-1
@Link grep-s
@Link grep-o
@Link grep-a
@Link grep-n
@Link grep-e
@key search
@key pattern
@key string
@key file
@link file
@key find
@Key word
@Link string
@Link regex
@link egrep
@link fgrep
```

grep.help

```
grep: Search for a pattern in a file.
Format: grep [options] pattern [file-list]
Options:        -c (count) display number of lines only
                -e (expression) pattern can begin with hyphen
                -l (list) display filenames only
                -n (number) display line numbers
                -s (status) return exit status only
                -v (reverse) reverse sense of test
                -i (ignore case) consider upper and lower case equivalent
Arguments: pattern : a regular expression, can be a simple string
```

grep.hmen

```
grep-s        f2 ** Summary of the grep command
grep-o        f3 ** Options for the grep command
grep-a        f4 ** Arguments for the grep command
grep-n        f5 ** Additional notes on the grep command
grep-e        f6 ** Examples of the grep command
file          f7 ** What is a file?
string        f8 ** What is a string?
regex         f9 ** What is a regular expression?
egrep         f1 ** egrep: Fast search for a pattern in a file
fgrep         f2 ** fgrep: Fast search for a string (word) in a file
```

grep-1.hcom

```
grep command after pattern is specified
@file grep-1
@link grep
@Link grep-s
@Link grep-o
@Link grep-a
@Link grep-n
@Link grep-e
@link file
@Link string
@Link regex
@Link return
```

grep-1.help

```
You may now type one or more file names in which to search.  If you don't
type any file names, the standard input will be searched.  When you have
typed all the file names you want to search, press the RETURN key.
```

grep-1.hmen

```
file         f2 ** You may now type the name of any existing file.
grep         f3 ** grep/egrep/fgrep: Search for a pattern in a file.
grep-s       f4 ** Summary of the grep command
grep-o       f5 ** Options for the grep command
grep-a       f6 ** Arguments for the grep command
grep-n       f7 ** Additional notes on the grep command
grep-e       f8 ** Examples of the grep command
file         f9 ** What is a file?
string       f1 ** What is a string?
regex        f2 ** What is a regular expression?
return       f3 ** What the RETURN key is and what it means
```

grep-e.hcom

```
Examples of the grep command
@Link grep
@Link grep-s
@Link grep-a
@Link grep-n
@Link grep-o
```

grep-e.help

```
The following examples assume that the working directory contains 3 files:
"testa", "testb", and "testc".  The contents of each file is shown below.
                testa           testb           testc
                -----           -----           -----
                aaabb           aaaaa           AAAAA
                bbbcc           bbbbb           BBBBB
                ff-ff           ccccc           CCCCC
                cccdd           ddddd           DDDDD
                dddaa

grep can search for a pattern that is a simple string of characters.  The
following command line searches "testa" for the string "bb".  grep displays
each line containing bb.

$ grep bb testa
aaabb
bbbcc
$

The -v option reverses the sense of the test.  The example below displays
all the lines WITHOUT bb.

$ grep -v bb testa
ff-ff
cccdd
dddaa
$

The -n flag displays the line number of each displayed line.

$ grep -n bb testa
1:aaabb
2:bbbcc
$

grep can search through more than one file.  Below, grep searches through
each file in the working directory.  (The ambiguous file reference * matches
all filenames.)  The name of the file containing the string precedes each
line of output.

$ grep bb *
testa:aaabb
testa:bbbcc
testb:bbbbb
$

The search that grep performs is case-sensitive.  Because the previous
examples specified lowercase bb, grep did not find the uppercase string,
BBBBB, in testc.  The -i option causes uppercase and lowercase letters
to be regarded as equivalent.

$ grep -i bb *
testa:aaabb
testa:bbbcc
testb:bbbbb
testc: BBBBB
$

The -c option displays the name of each file, followed by the number of
lines in the file that contain a match.

$ grep -c bb *
testa:2
```

```
testb:1
testc:0
$
```

The -e option searches for a string that begins with a hyphen. This option causes grep to accept the hyphen as part of the pattern and not as an indicator that an option follows.

```
$ grep -e -ff *
testa: ff-ff
$
```

The following command line displays lines from the file text2 that contain a string of characters starting with "st", followed by zero or more characters (.*), and ending in "ing".

```
$ grep 'st.*ing' text2
...
...
$
```

grep-e.hmen

```
grep         f2 ** grep/egrep/fgrep: Search for a pattern in a file.
grep-s       f3 ** Summary of the grep command
grep-a       f4 ** Arguments for the grep command
grep-n       f5 ** Additional notes on the grep command
grep-o       f6 ** Options for the grep command
```

grep-s.hcom

```
Summary of the grep command
@Link grep
@Link grep-o
@Link grep-a
@Link grep-n
@Link grep-e
@Link string
```

grep-s.help

grep searches one or more files, line by line, for a pattern. The pattern can be a simple string, or another form of a regular expression. grep takes various actions, specified by options, each time it finds a line that contains a match for the pattern.

grep takes its input from files specified on the command line or from the standard input.

grep-s.hmen

```
grep         f2 ** grep/egrep/fgrep: Search for a pattern in a file.
grep-o       f3 ** Options for the grep command
grep-a       f4 ** Arguments for the grep command
grep-n       f5 ** Additional notes on the grep command
grep-e       f6 ** Examples of the grep command
string       f7 ** What is a string?
```

mail.hcom

```
Commands relating to the key word 'mail'
@link ml
@link biff
@link ck
@link hg
@link mailq
@link msgs
```

mail.help

```
Select the appropriate menu item to find out about the command listed,
    which matched the key word 'mail'
```

mail.hmen

```
ml              f2 ** mail: Send or receive mail
biff            f3 ** biff: be notified if mail arrives and who it is from
ck              f4 ** ck: check if new mail has arrived
hg              f5 ** hg: Mercury mail reading program
mailq           f6 ** mailq: Examine or delete entries in the network mail queue
msgs            f7 ** msgs: System messages and junk mail program.
```

message.hcom

```
Commands relating to the key word 'messages'
@link bb
@link echo
@link mesg
@link msgs
@link post
@link rsend
@link send
@link wall
```

message.help

```
Select the appropriate menu item to find out about the command listed,
    which matched the key word 'messages'
```

message.hmen

```
bb              f2 ** bb: print notices from bulletin board(s)
echo            f3 ** echo: Display a message
mesg            f4 ** mesg: Enable/disable reception of messages
msgs            f5 ** msgs: System messages and junk mail program.
post            f6 ** post: Post notice on bulletin board(s)
rsend           f7 ** rsend: send a message to any user on any UNIX machine on the network
send            f8 ** send: Send a message to another user
wall            f9 ** wall: Write to all users
```

ml-e.hcom

```
Examples of the mail command
@Link ml
@Link ml-s
@Link ml-a
@Link ml-c
@Link ml-o
```

ml-e.help

The first example below shows how to send a message to several users. In this case, mail sends the message to users with the login names of hls, alex, and jenny.

```
$ mail hls alex jenny
(message)
(message)
.
```

You can also compose a message in a file and then send it by redirecting the input to mail. The command below sends the file today to barbara.

```
$ mail barbara < today
```

ml-e.hmen

```
ml          f2 ** mail: Send or receive mail
ml-s        f3 ** Summary of the mail command
ml-a        f4 ** Arguments for the mail command
ml-c        f5 ** Responses to the '?' prompt in the mail utility
ml-o        f6 ** Options for the mail command
```

ml.hcom

```
mail: Send or receive mail
@link @ml
@user @ml-1
@Link ml-s
@opt ml-o
@Link ml-a
@link ml-c
@Link ml-e
@key mail
@link ck
@Link biff
@link post
@link send
@link rsend
```

ml.help

```
mail: Send or receive mail
Format-1: mail user-list
Format-2: mail [options]
```

The first format sends mail to the user-list. The second format displays mail that you have received and prompts you with a ? following each letter.

```
Responses to ?
        ?       help
        d       delete mail
        q       quit
        w       write to mbox
        w file  write to named file
        RETURN  proceed to next letter
        p       redisplay previous letter
Options:
        -p      display mail, no questions
        -q      quit on interrupt
        -r      reverse order
```

ml.hmen

```
ml-s          f2 ** Summary of the mail command
ml-o          f3 ** Options for the mail command
ml-a          f4 ** Arguments for the mail command
ml-c          f5 ** Responses to the '?' prompt in the mail utility
ml-e          f6 ** Examples of the mail command
ck            f7 ** ck: check if new mail has arrived
biff          f8 ** biff: be notified if mail arrives and who it is from
post          f9 ** post: Post notice on bulletin board(s)
send          f1 ** send: Send a message to another user
rsend         f2 ** rsend: send a message to any user on any UNIX machine on the network
```

root.hcom

```
! This is the root file for the ACRONYM help system.  This is where
! I'll explain how all such files work.
!
! First of all, each node has three files.  The .help file is pure help
! text.  The .hcom files, like this, are information used for compiling
! the current version of the help network used for dynamic help and
! dynamic menu generation.  All hcom files must adhere to the same fairly
! stupid syntax, which is explained only here.  (The third file for each
! node is the .hmen file, which is a menu generated automatically from
! the .hcom file by the ack program, as necessary.)
!
! This is a comment.  Comments must start the line with a bang.
!
! The name "root" is special.  ACRONYM knows that "root.hcom" is where
! everything begins.  However, the syntax is the same for each node,
! even the root.  Any file with an ".hcom" extension in the help directory
! (specified as HELPDIR in ack.ml and ack.c) is considered such a node.
!
! The first non-comment line is crucial:  it is the short description of
! the corresponding help node (.help file), and is used for dynamic
! menu generation.
!
! All remaining non-comment lines have two fields, separated by white
! space.  Undetermined havoc may result if the line has only one field.
!
! The first field is the pattern to be matched.  If this doesn't start
! with an "@" sign, it is a fixed pattern (word) that must be matched
! exactly.  The extras are provided by the following possible magic patterns:
!       @any            matches anything
!       @opt            matches anything that starts with a "-"
!       @file           pattern must be a file name
!       @filewrite      pattern must be a potential (creatable) file name
!       @dir            pattern must be a directory name
!       @user           pattern must be a user login name
!       @link           non-syntactic link; used only for menu generation,
!                        never for command parsing.
!       @key            The inverse of @link; creates a link to the current
!                        frame from the one named on the rest of the line.
!                        This is the mechanism used for specifying key words.
!                        A frame exists for each key word, from which each
!                        node declares back-links via @key.
!
! Creating more magic "@" options is fairly easy when it becomes necessary,
! but of course that does involve some reprogramming.
!
! The second field is the node to which a match passes control.  It is the
! name to which ".hcom" or ".help" should be appended.  The @Link and @key
! patterns give node names which are not used for parsing, only for menu
! traversals.  You can do the reverse -- specify a link that is for parsing
! but NOT for menu traversal -- by preceding the node name (the second field)
! with an "@".  Thus, if a line says "@opt @foo", then an option typed at that
! point will indeed pass control to node foo; however, there will be no link
! to node foo in the hmen menu.
!
! WARNING:  Patterns other than the special ones that start with "@" are
! case sensitive.  That is, a pattern of "ls" won't match "LS".  However,
! things that start with "@" are case-insensitive, as are node names.
! (Actually, that's a slight oversimplification:  node names are mapped
! entirely to lower case, so that the file names (.hcom, etc.) MUST be
! all lower case.  References to node names within hcom files are more
! flexible, in that ACRONYM will assume that you meant to type in all lower
! case for node names.
!
```

```
! Remember, the first non-comment line is the magic keyline.
!
Introduction and main menu for the ACRONYM help system
@link acronym
at at
bb bb
biff biff
cal cal
calendar cale
cat cat
cc cc
ccat ccat
cd cd
chdir cd
chmod chmod
chat chat
ck ck
cmp cmp
cmuftp cmuftp
col col
comm comm
compact compact
cp cp
cz cz
date dt
dcat dcat
del del
df df
diff diff
du du
echo echo
ecp ecp
egrep grep
expunge expunge
fgrep grep
file fl
find find
fing fing
finger finger
ftp ftp
grep grep
head head
hg hg
kill kill
ln ln
lpq lpq
lpr lpr
lprm lprm
ls ls
lsd lsd
mail ml
mailq mailq
mesg mesg
msgs msgs
mkdir mkdir
more more
mv mv
od od
opr opr
passwd passwd
pj pj
post post
pr pr
print print
ps ps
```

```
pwd pwd
reply reply
rev rev
rm rm
rmdir rmdir
rsend rsend
see see
send send
sleep sleep
sort sort
spell spell
strings strngs
stty stty
tail tail
telnet telnet
talk talk
time time
touch touch
tset tset
tty tty
ttyinfo ttyinfo
ttyis ttyis
uncompact uncom
undel undel
uniq uniq
uptime uptime
users users
u users
utime utime
w w
wall wall
wc wc
whenis whenis
who who
whoami whoami
write write
```

root.hmen

```
acronym      f2 ** How to use the ACRONYM help system
at           f3 ** at: Execute a Shell script at a specified time
bb           f4 ** bb: print notices from bulletin board(s)
biff         f5 ** biff: be notified if mail arrives and who it is from
cal          f6 ** cal: display calendar
cale         f7 ** calendar: reminder calendar
cat          f8 ** cat: display a text file
cc           f9 ** cc: C compiler
ccat         f1 ** ccat: Print compressed files in uncompressed format
cd           f2 ** cd or chdir: Change to another working directory
chmod        f3 ** chmod: Change the access mode of a file
chat         f4 ** chat: Communicate with (log in to) another machine on the Ethernet
ck           f5 ** ck: check if new mail has arrived
cmp          f6 ** cmp: Compare two files to see if they differ
cmuftp       f7 ** cmuftp: Transfer files to and from other machines on the Ethernet
col          f8 ** col: filter reverse line feeds
comm         f9 ** comm: Compare two files and print matching and non-matching lines
compact      f1 ** compact: compress files to save space
cp           f2 ** cp: Copy file
cz           f3 ** cz: convert files to press format and print them on the Dover
dt           f4 ** date: print today's date and time (and set it if you're the superuser)
dcat         f5 ** dcat: convert troff output to press format for printing on Dover
del          f6 ** del: Delete a file reversibly
df           f7 ** df: Report on free disk space
diff         f8 ** diff: Display the differences between two files in editing-oriented format
du           f9 ** du: Summarize disk usage
echo         f1 ** echo: Display a message
ecp          f2 ** ecp: Ethernet file copy
grep         f3 ** grep/egrep/fgrep: Search for a pattern in a file.
expunge      f4 ** expunge: Permanently get rid of deleted files
fl           f5 ** file: Display file classification
find         f6 ** find: Find files.
fing         f7 ** fing: Front end for finger
finger       f8 ** finger: User information lookup program
ftp          f9 ** ftp: Transfer files to and from other machines on the Internet (ARPAnet)
head         f1 ** head: Give first few lines of a file
hg           f2 ** hg: Mercury mail reading program
kill         f3 ** kill: Terminate a process
ln           f4 ** ln: Make a link to a file
lpq          f5 ** lpq: Display line printer queue
lpr          f6 ** lpr: Print a file on the line printer
lprm         f7 ** lprm: Remove an entry from the line printer queue.
ls           f8 ** ls: Display information about a file
lsd          f9 ** lsd: list deleted files
ml           f1 ** mail: Send or receive mail
mailq        f2 ** mailq: Examine or delete entries in the network mail queue
mesg         f3 ** mesg: Enable/disable reception of messages
msgs         f4 ** msgs: System messages and junk mail program.
mkdir        f5 ** mkdir: Make a directory
more         f6 ** more: Display a file, one screenful at a time.
mv           f7 ** mv: Rename a file
od           f8 ** od: Dump the contents of a file
opr          f9 ** opr: Queue files for Dover or line-printer and/or examine & modify queue
passwd       f1 ** passwd: Change login password
pj           f2 ** pj: print system and user identification
post         f3 ** post: Post notice on bulletin board(s)
pr           f4 ** pr: Paginate file for printing
print        f5 ** Commands relating to the key word 'print'
ps           f6 ** ps: Display process status
pwd          f7 ** pwd:  Print working directory name
reply        f8 ** reply: Join in to a conversation with another user
rev          f9 ** rev: Reverse each line of a file.
rm           f1 ** rm: Delete a file (remove a link)
```

```
rmdir        f2 ** rmdir: Delete a directory
rsend        f3 ** rsend: send a message to any user on any UNIX machine on the network
see          f4 ** see: See what a file has in it
send         f5 ** send: Send a message to another user
sleep        f6 ** sleep: Put the current process to sleep
sort         f7 ** sort: Sort and/or merge files
spell        f8 ** spell: Check a file for spelling errors.
strngs       f9 ** strings: Find and display the printable strings in a binary file
stty         f1 ** stty: Display or establish terminal parameters
tail         f2 ** tail: Display the last part (tail) of a file.
telnet       f3 ** telnet: Communicate with (log in to) another machine on the Internet
talk         f4 ** talk: Initiate a conversation with another user
time         f5 ** time: Time a command
touch        f6 ** touch: Update a file's modification time.
tset         f7 ** tset: Set terminal modes
tty          f8 ** tty: Display the terminal pathname
ttyinfo      f9 ** ttyinfo: Find out information about terminals
ttyis        f1 ** ttyis:  Discover where a terminal is
uncom        f2 ** uncompact: Restore compressed files to their original state
undel        f3 ** undel: Restore a deleted file
uniq         f4 ** uniq: Display lines of a file that are unique
uptime       f5 ** uptime: Show how long system has been up
users        f6 ** u or users: Compact list of users who are on the system
utime        f7 ** utime: Modify the access and/or modified times of a file
w            f8 ** w: List who is on the system and what they are doing
wall         f9 ** wall: Write to all users
wc           f1 ** wc: Display the number of lines, words, and characters in a file.
whenis       f2 ** whenis: calendar expert program that prints dates and times
who          f3 ** who: Display names of users
whoami       f4 ** whoami: Print effective current user id.
write        f5 ** write: Send a message to another user
```

root.help

```
Welcome to ACRONYM.  If you don't want to be here, press DEL to exit.

In addition to normal typewriter-style keys, your computer has a pointing
device known as a "mouse."  You can move this device around, causing the arrow
on your screen to move around and point at different parts of the screen.
In this system, you can use the mouse to get help in several ways.

To begin with, you will notice that the highlighted line underneath the text
you are now reading says "Point HERE to move forward."  If you use the
mouse to point the arrow at the word "HERE", and then press any button on
the mouse, you will find that the text in this window is scrolled forward --
that is, the beginning of the text will disappear, and new text will appear
at the end of the window.  Try it and see.

If you can't seem to get the window to scroll forward, this probably means
that you are pointing the arrow a little too high.  Note that it is the
arrow head, not its body, that should be pointing at the word "HERE".

In general, whenever this window or the one below it has more text than is
currently visible, the line at the bottom of the window will offer you a
place to point with the mouse to scroll the text in the window.  Right now,
for instance, you will notice that there is a part of that line that says
"Point HERE to move backward."  In general, this special line will be the
only way you have of knowing that there are more texts than what you can
read at the moment, so you should keep your eyes open for it.

As you may have guessed, this top window on your screen will be used for
explanatory texts like the one you're reading now.  The second window is a
"menu" of options that you can use to get additional help.  Each line
contains a brief description of another help topic.  If you use the mouse to
point at that line, and then press any button on the mouse, you will be
shown the corresponding help texts.

Sometimes there will be more choices in the menu than will fit in the menu
window.  At that time, the special line at the bottom of the window will
offer you a place where you can point in order to move forward or backward
in the list of choices.  This is just like moving around in the help window.
Feel free to try it and see.

There are many other ways in which you can get help using the ACRONYM help
system.  To find out about them, look through the list of choices in the
menu window and point to the appropriate line using your mouse.  (Depending
on when you are seeing this, you will probably want either the first or
second line in the menu window.)
```

send.hcom

```
send: Send a message to another user
@any send
@Link send-s
@Link send-a
@Link send-n
@Link send-e
@Link talk
@link reply
@Link write
@Link rsend
@key conv
@key user
```

send.help

```
send: send a message to another user
Format: send destination-user[@host] [tty-name] [-all] [message]
Arguments: destination-user : person you want to send a message to.
           tty-name : use to resolve ambiguity if the destination-user is
                  logged on more than once.
           [-all] : send to all of the user's terminals.
           [message] : send a one-line message to the user.
```

send.hmen

```
send-s         f2 ** Summary of the send command
send-a         f3 ** Arguments for the send command
send-n         f4 ** Additional notes on the send command
send-e         f5 ** Examples of the send command
talk           f6 ** talk: Initiate a conversation with another user
reply          f7 ** reply: Join in to a conversation with another user
write          f8 ** write: Send a message to another user
rsend          f9 ** rsend: send a message to any user on any UNIX machine on the network
mesg           f1 ** mesg: Enable/disable reception of messages
```

word.hcom

```
Commands relating to the key word 'word'
```

word.help

```
Select the appropriate menu item to find out about the command listed,
    which matched the key word 'word'
```

word.hmen

```
grep           f2 ** grep/egrep/fgrep: Search for a pattern in a file.
egrep          f3 ** egrep: Fast search for a pattern in a file
fgrep          f4 ** fgrep: Fast search for a string (word) in a file
wc             f5 ** wc: Display the number of lines, words, and characters in a file.
```

Appendix D
The Two UNIX Manuals

The experiments reported in this thesis strongly suggest that the quality of help texts is far more important than the mechanisms by which those texts are accessed or presented. However, several standard "readability" metrics failed to reliably reflect which of the texts studied would be most useful. Although there is a wide literature on the issue of text readability, there are no "fail-safe" formulas to guarantee readability.

In the interest of promoting more informative and readable texts. samples of each are reproduced here. For each of five commands (grep, ls, mail, rm, and sort), the standard UNIX manual entries and the manual entries for the hybrid system are reproduced in this chapter.

The standard manual entries are the copyrighted property of AT&T Bell Laboratories, as modified at Carnegie-Mellon University. The hybrid manual entries are derived from Mark Sobell's book[28], and are reproduced here with permission of the author and publisher, who retain all rights to further reproduction. The form in which they appear here is slightly modified to reflect some variations in the CMU versions of the commands they document, and to correct a few minor omissions. They appear in precisely the form in which they were used in the experiments.

In comparing these sample texts, the reader should bear in mind that, for whatever reason, the Sobell/ACRONYM texts proved significantly better than the standard manual texts, and by a wide margin. It is left to the reader's judgment to determine why this was the case.

[28]From A Practical Guide to the UNIX™ System by Mark G. Sobell. Copyright (c) 1984 by Mark G. Sobell. Published by the Benjamin/Cummings Publishing Company.

D.1. The grep command: standard UNIX manual

```
GREP(1)              UNIX Programmer's Manual              GREP(1)


NAME
     grep, egrep, fgrep - search a file for a pattern

SYNOPSIS
     grep [ option ] ...  expression [ file ] ...

     egrep [ option ] ...  [ expression ] [ file ] ...

     fgrep [ option ] ...  [ strings ] [ file ]

DESCRIPTION
     Commands of the grep family search the input files (standard
     input default) for lines matching a pattern.  Normally, each
     line found is copied to the standard output.  Grep patterns
     are limited regular expressions in the style of ex(1); it
     uses a compact nondeterministic algorithm.  Egrep patterns
     are full regular expressions; it uses a fast deterministic
     algorithm that sometimes needs exponential space.  Fgrep
     patterns are fixed strings; it is fast and compact.  The
     following options are recognized.

     -v   All lines but those matching are printed.

     -x   (Exact) only lines matched in their entirety are
          printed (fgrep only).

     -c   Only a count of matching lines is printed.

     -l   The names of files with matching lines are listed
          (once) separated by newlines.

     -n   Each line is preceded by its relative line number in
          the file.

     -b   Each line is preceded by the block number on which it
          was found.  This is sometimes useful in locating disk
          block numbers by context.

     -i   The case of letters is ignored in making comparisons.
          (E.g. upper and lower case are considered identical.)
          (grep and fgrep only)

     -s   Silent mode.  Nothing is printed (except error mes-
          sages).  This is useful for checking the error status.

     -w   The expression is searched for as a word (as if sur-
          rounded by '\<' and '\>', see ex(1).) (grep only)

     -e expression
          Same as a simple expression argument, but useful when
          the expression begins with a -.

     -f file
          The regular expression (egrep) or string list (fgrep)
          is taken from the file.
```

In all cases the file name is shown if there is more than one input file. Care should be taken when using the characters $ * [^ | ( ) and \ in the expression as they are also meaningful to the Shell. It is safest to enclose the entire expression argument in single quotes ' '.

Fgrep searches for lines that contain one of the (newline-separated) strings.

Egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character other than newline matches that character.

The character ^ ($) matches the beginning (end) of a line.

A . matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string. Ranges of ASCII character codes may be abbreviated as in 'a-z0-9'. A] may occur only as the first character of the string. A literal - must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of 0 or more (1 or more, 0 or 1) matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [] then *+? then concatenation then | and newline.

SEE ALSO
ex(1), sed(1), sh(1)

DIAGNOSTICS
Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

BUGS
Ideally there should be only one grep, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

Lines are limited to 256 characters; longer lines are truncated.

D.2. The grep command: Sobell/ACRONYM Version

```
grep: Search for a pattern in a file.
Format: grep [options] pattern [file-list]
Options:        -c (count) display number of lines only
                -e (expression) pattern can begin with hyphen
                -l (list) display filenames only
                -n (number) display line numbers
                -s (status) return exit status only
                -v (reverse) reverse sense of test
                -i (ignore case) consider upper and lower case equivalent
Arguments: pattern : a regular expression, can be a simple string
```

SUMMARY OF THE GREP COMMAND

grep searches one or more files, line by line, for a pattern. The pattern can be a simple string, or another form of a regular expression. grep takes various actions, specified by options, each time it finds a line that contains a match for the pattern.

grep takes its input from files specified on the command line or from the standard input.

OPTIONS FOR THE GREP COMMAND

If you do not specify any options, grep sends its lines to the standard output. If you specify more than one file on the command line, grep precedes each line that it displays with the name of the file that it came from.

```
        -c (count) grep only indicates the number of lines in each file that
              contain a match.
        -e (expression) This option allows you to use a pattern beginning
              with a hyphen (-). If you do not use this option and specify
              a pattern that begins with a hyphen, grep assumes that the
              hyphen introduces an option and the search will not work.
        -l (list) grep displays the name of each file that contains one or
              more matches. grep displays each filename only once, even
              if the file contains more than one match.
        -n (number) grep precedes each line by its line number in the file.
              The file does not need to contain line numbers -- this number
              represents the number of lines in the file up to and
              including the displayed line.
        -s (status) grep returns an exit status value without any output.
        -v (reverse sense of test) This option causes lines NOT containing
              a match to satisfy the search. When you use this option by
              itself, grep displays all lines that do not contain a match.
        -i (ignore case) This option causes lowercase letters in the pattern
              to match uppercase letters in the file and vice versa.
```

ARGUMENTS FOR THE GREP COMMAND

The pattern is a simple string or a regular expression. You must quote regular expressions that contain special characters, SPACEs, or TABs. An easy way to quote these characters is to enclose the entire expression within apostrophes.

The file-list contains pathnames of plain text files that grep searches.

ADDITIONAL NOTES ON THE GREP COMMAND

grep returns an exit status of zero if a match is found, one if no match is found, and two if the file is not accessible or there is a syntax error.

There are two utilities that perform functions similar to that of grep. The egrep utility can be faster than grep, but may also use more space. fgrep is fast and compact, but can process only simple strings, not regular expressions.

EXAMPLES OF THE GREP COMMAND

The following examples assume that the working directory contains 3 files: "testa", "testb", and "testc". The contents of each file is shown below.

```
                testa          testb          testc
                -----          -----          -----
                aaabb          aaaaa          AAAAA
                bbbcc          bbbbb          BBBBB
                ff-ff          ccccc          CCCCC
                cccdd          ddddd          DDDDD
                dddaa
```

grep can search for a pattern that is a simple string of characters. The following command line searches "testa" for the string "bb". grep displays each line containing bb.

```
$ grep bb testa
aaabb
bbbcc
$
```

The -v option reverses the sense of the test. The example below displays all the lines WITHOUT bb.

```
$ grep -v bb testa
ff-ff
cccdd
dddaa
$
```

The -n flag displays the line number of each displayed line.

```
$ grep -n bb testa
1:aaabb
2:bbbcc
$
```

grep can search through more than one file. Below, grep searches through each file in the working directory. (The ambiguous file reference * matches all filenames.) The name of the file containing the string precedes each line of output.

```
$ grep bb *
testa:aaabb
testa:bbbcc
testb:bbbbb
$
```

The search that grep performs is case-sensitive. Because the previous examples specified lowercase bb, grep did not find the uppercase string, BBBBB, in testc. The -i option causes uppercase and lowercase letters to be regarded as equivalent.

```
$ grep -i bb *
testa:aaabb
testa:bbbcc
testb:bbbbb
testc: BBBBB
$
```

The -c option displays the name of each file, followed by the number of lines in the file that contain a match.

```
$ grep -c bb *
testa:2
testb:1
testc:0
$
```

The -e option searches for a string that begins with a hyphen. This option causes grep to accept the hyphen as part of the pattern and not as an indicator that an option follows.

```
$ grep -e -ff *
testa: ff-ff
$
```

The following command line displays lines from the file text2 that contain a string of characters starting with "st", followed by zero or more characters (.*), and ending in "ing".

```
$ grep 'st.*ing' text2
...
...
$
```

D.3. The ls command: standard UNIX manual

```
LS(1)                UNIX Programmer's Manual                LS(1)


NAME
     ls - list contents of directory

SYNOPSIS
     ls [ -abcdfgilmqrstux1CFR ] name ...
     l [ ls options ] name ...

DESCRIPTION
     For each directory argument, ls lists the contents of the
     directory; for each file argument, ls repeats its name and
     any other information requested.  The output is sorted
     alphabetically by default.  When no argument is given, the
     current directory is listed.  When several arguments are
     given, the arguments are first sorted appropriately, but
     file arguments appear before directories and their contents.

     There are three major listing formats.  The format chosen
     depends on whether the output is going to a teletype, and
     may also be controlled by option flags.  The default format
     for a teletype is to list the contents of directories in
     multi-column format, with the entries sorted down the
     columns.  (Files which are not the contents of a directory
     being interpreted are always sorted across the page rather
     than down the page in columns.  This is because the indivi-
     dual file names may be arbitrarily long.) If the standard
     output is not a teletype, the default format is to list one
     entry per line.  Finally, there is a stream output format in
     which files are listed across the page, separated by ','
     characters.  The -m flag enables this format; when invoked
     as l this format is also used.

     There are an unbelievable number of options:

     -l   List in long format, giving mode, number of links,
          owner, size in bytes, and time of last modification for
          each file.  (See below.) If the file is a special file
          the size field will instead contain the major and minor
          device numbers.

     -t   Sort by time modified (latest first) instead of by
          name, as is normal.

     -a   List all entries; usually '.' and '..' are suppressed.

     -s   Give size in blocks, including indirect blocks, for
          each entry.

     -d   If argument is a directory, list only its name, not its
          contents (mostly used with -l to get status on direc-
          tory).

     -r   Reverse the order of sort to get reverse alphabetic or
          oldest first as appropriate.

     -u   Use time of last access instead of last modification
          for sorting (-t) or printing (-l).
```

```
-c   Use time of file creation for sorting or printing.

-i   Print i-number in first column of the report for each
     file listed.

-f   Force each argument to be interpreted as a directory
     and list the name found in each slot.  This option
     turns off -l, -t, -s, and -r, and turns on -a; the
     order is the order in which entries appear in the
     directory.

-g   Give group ID instead of owner ID in long listing.

-m   force stream output format

-1   force one entry per line output format, e.g. to a tele-
     type

-C   force multi-column output, e.g. to a file or a pipe

-q   force printing of non-graphic characters in file names
     as the character '?'; this normally happens only if the
     output device is a teletype

-b   force printing of non-graphic characters to be in the
     \ddd notation, in octal.

-x   force columnar printing to be sorted across rather than
     down the page; this is the default if the last charac-
     ter of the name the program is invoked with is an 'x'.

-F   cause directories to be marked with a trailing '/' and
     executable files to be marked with a trailing '*'; this
     is the default if the last character of the name the
     program is invoked with is a 'f'.

-R   recursively list subdirectories encountered.

The mode printed under the -l option contains 11 characters
which are interpreted as follows: the first character is

d  if the entry is a directory;
b  if the entry is a block-type special file;
c  if the entry is a character-type special file;
m  if the entry is a multiplexor-type character special
   file;
-  if the entry is a plain file.

The next 9 characters are interpreted as three sets of three
bits each.  The first set refers to owner permissions; the
next to permissions to others in the same user-group; and
the last to all others.  Within each set the three charac-
ters indicate permission respectively to read, to write, or
to execute the file as a program.  For a directory, 'exe-
cute' permission is interpreted to mean permission to search
the directory for a specified file.  The permissions are
indicated as follows:

r  if the file is readable;
w  if the file is writable;
x  if the file is executable;
-  if the indicated permission is not granted.

The group-execute permission character is given as s if the
file has set-group-ID mode; likewise the user-execute per-
```

mission character is given as s if the file has set-user-ID mode.

The last character of the mode (normally 'x' or '-') is t if the 1000 bit of the mode is on. See chmod(1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user ID's for 'ls -l'.
/etc/group to get group ID's for 'ls -g'.

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as ''ls -s'' is much different than ''ls -s | lpr''. On the other hand, not doing this setting would make old shell scripts which used ls almost certain losers.

Column widths choices are poor for terminals which can tab.

D.4. The ls command: Sobell/ACRONYM Version

```
ls: Display information about a file
Format: ls [options] [file-list]
Options: -a     all entries
         -d     directory
         -g     group
         -i     display i-numbers
         -l     long
         -r     reverse
         -s     size in blocks
         -t     modified time
         -u     accessed time

SUMMARY OF THE LS COMMAND

ls displays information about one or more files.  It lists the information
alphabetically by filename unless you use an option to change the order.

OPTIONS FOR THE LS COMMAND

The options determine the type of information, and the order in which the
information is displayed.  When you do not use an option, ls displays a
short listing, containing only the names of files.   The options are:

        -a (all entries) Without a file-list (no argument on the command
               line), this option causes ls to display information about
               all the files in the working directory, including invisible
               (hidden) files.  When you do not use this option, ls does
               not list information about invisible files unless you
               specifically request it.
                 In a similar manner, when you use this option with a
               file-list that includes an appropriate ambiguous file
               reference (wildcard), ls displays information about
               invisible files.
        -d (directory) This option causes ls to display the names of
               directories without displaying their contents.  When
               you give this option without an argument, ls displays
               information about the working directory (.).  This option
               displays plain files normally.
        -g (group) This option causes ls to display group identification.
               When you use this option with the -l option, ls replaces the
               owner name in the display with the group name.
        -i (i-number) This option causes ls to display the i-number of each
               file.  The i-number is the unique identifying number that
               UNIX assigned to the file.  It is usually useless.
        -l (long) This option causes ls to display eight columns of
               information about each file.  These columns are described in
               another help message, "ls -l: long listings of file
               information", which is one of your current help menu choices.
        -r (reverse) This option causes ls to display the list of filenames
               in reverse alphabetical order or, when used in conjunction
               with the -t or -u options, in reverse time order (least
               recently modified/accessed first).
        -s (size) This option causes ls to display the size of each file in
               512 byte blocks.  The size precedes the filename.
                 When used with the -l option, the -s option causes ls to
               display the size in column one and to shift each of the
               other items over one column to the right.
        -t (time modified) This option causes ls to display the list of
               filenames in order by the time of last modification:  It
               displays the files that were modified most recently first.
        -u (time accessed) This option causes ls to display the list of
               filenames together with the last time that each file was
```

accessed. The list is in alphabetical order if you do not use an option that specifies another order.

LS -L: LONG LISTINGS OF FILE INFORMATION

The -l option causes ls to display eight columns of information about each file in a format something like this:

```
drwxrwxr-x 1 jenny      1296 Apr  6 22:56 letter
```

The first column, which contains 10 characters, is divided as follows:

The first character describes the type of file:
- - indicates a plain file
- b indicates a block device file
- c indicates a character device file
- d indicates a directory file

The next nine characters represent all the access permissions associated with the file. These nine characters are divided into three sets of three characters each.

The first three characters represent the owner's access permissions. If the owner has read access permission to the file, an "r" appears in the first character position. If the owner is not permitted to read the file, a hyphen appears in this position. The next two positions represent the owner's write and execute access permissions. A "w" appears in the second position if the owner is permitted to write to the file, and an "x" appears in the third position if the owner is permitted to execute the file. An "s" in the third position indicates that the file has set user ID permission. A hyphen appears if the owner does not have the access permission associated with the character position.

In a similar manner, the second and third sets of three characters represent the access permissions of the user's group and other users. An "s" in the third position of the second set of characters indicates that the file has set group ID permission.

Refer to the chmod utility for information on changing access permissions.

The next column indicates the number of links to the file.

The third column displays the name of the owner of the file.

The fourth column indicates the size of the file in bytes, or, if information about a device file is being displayed, the major and minor device numbers. In the case of a directory, this is the size of the actual directory file, not the size of the files that are entries within the directory.

The fifth and sixth columns display the date and time the file was last modified.

The last column displays the name of the file.

ARGUMENTS FOR THE LS COMMAND

When you do not use an argument, ls displays the names of all the files in the working directory.

The file-list argument contains one or more pathnames of files that ls displays information for. You can use the pathnames of any plain, directory, or device file. These pathnames can include ambiguous file references.

When you give an ambiguous file reference (wildcard), ls displays the names of all the files in any directories specified by the wildcard, in addition to files in the working directory.

When you specify a directory file, ls displays the contents of the directory. ls displays the name of the directory only when it is needed to avoid ambiguity (i.e., when ls is displaying the contents of more than one directory, it displays the names of the directories to indicate which files you can find in which directory). If you specify a plain file, ls displays information about just that file.

EXAMPLES OF THE LS COMMAND

All of the following examples assume that the user does not change from the current working directory.

The first command line shows the ls utility without any options or arguments. ls displays an alphabetical list of the names of the files in the working directory.

```
$ ls
bin     calendar        letters
c       execute         shell
```

Next, the -l (long) option causes ls to display a long list. The files are still in alphabetical order.

```
$ ls -l
total 8
drwxrwxr-x  2 jenny   80 Nov 20 09:17 bin
drwxrwxr-x  2 jenny  144 Sep 26 11:59 c
-rw-rw-r--  1 jenny  104 Nov 28 11:44 calendar
-rwxrw-r--  1 jenny   85 Nov  6 08:27 execute
drwxrwxr-x  2 jenny   32 Apr  6 22:56 letters
drwxrwxr-x 16 jenny 1296 Dec  6 17:33 shell
```

The -a option lists invisible files when you do not specify an argument.

```
.       .profile        c       execute         shell
..      bin     calendar        letters
```

Combining the -a and -l options above causes ls to display a long listing of all the files, including invisible files, in the working directory. The list is still in alphabetical order.

```
$ ls -al
total 12
drwxrwxr-x  6 jenny  480 Dec  6 17:42 .
drwxrwxr-- 26 root   816 Dec  6 14:45 ..
-rw-rw-r--  1 jeny   161 Dec  6 17:15 .profile
drwxrwxr-x  2 jenny   80 Nov 20 09:17 bin
drwxrwxr-x  2 jenny  144 Sep 26 11:59 c
-rw-rw-r--  1 jenny  104 Nov 28 11:44 calendar
-rwxrw-r--  1 jenny   85 Nov  6 08:27 execute
drwxrwxr-x  2 jenny   32 Apr  6 22:56 letters
drwxrwxr-x 16 jenny 1296 Dec  6 17:33 shell
```

The -r (reverse order) option is added to the command line from the previous example. The list is now in reverse alphabetical order.

```
$ ls -ral
total 12
drwxrwxr-x 16 jenny 1296 Dec  6 17:33 shell
drwxrwxr-x  2 jenny   32 Apr  6 22:56 letters
-rwxrw-r--  1 jenny   85 Nov  6 08:27 execute
-rw-rw-r--  1 jenny  104 Nov 28 11:44 calendar
drwxrwxr-x  2 jenny  144 Sep 26 11:59 c
drwxrwxr-x  2 jenny   80 Nov 20 09:17 bin
-rw-rw-r--  1 jeny   161 Dec  6 17:15 .profile
drwxrwxr-- 26 root   816 Dec  6 14:45 ..
```

```
drwxrwxr-x  6 jenny  480 Dec  6 17:42 .
```

The -t (time) option causes ls to list files so that the most recently modified file appears at the top of the list.

```
$ ls -tl
total 8
drwxrwxr-x 16 jenny 1296 Dec  6 17:33 shell
-rw-rw-r--  1 jenny  104 Nov 28 11:44 calendar
drwxrwxr-x  2 jenny   80 Nov 20 09:17 bin
-rwxrw-r--  1 jenny   85 Nov  6 08:27 execute
drwxrwxr-x  2 jenny  144 Sep 26 11:59 c
drwxrwxr-x  2 jenny   32 Apr  6 22:56 letters
```

The -r option, when combined with the -t option, causes ls to list files so that the least-recently modified file appears at the top of the list.

```
$ ls -trl
total 8
drwxrwxr-x  2 jenny   32 Apr  6 22:56 letters
drwxrwxr-x  2 jenny  144 Sep 26 11:59 c
-rwxrw-r--  1 jenny   85 Nov  6 08:27 execute
drwxrwxr-x  2 jenny   80 Nov 20 09:17 bin
-rw-rw-r--  1 jenny  104 Nov 28 11:44 calendar
drwxrwxr-x 16 jenny 1296 Dec  6 17:33 shell
```

The next example shows the ls utility with a directory filename as an argument. ls lists the contents of the directory in alphabetical order.

```
$ ls bin
c       e       lsdir
```

The -l option gives a long listing of the contents of the directory.

```
$ ls -l bin
total 3
-rwxrw-r-x  1 jenny    48 Oct  6 21:38 c
-rwxrw-r--  1 jenny   156 Oct  6 21:40 e
-rwxrw-r--  1 jenny   136 Nov  7 16:48 lsdir
```

To find out information about the directory file itself, use the -d (directory) option. This causes ls to only list information about the directory.

```
$ ls -dl bin
drwxrwxr-x  2 jenny    80 Nov 20 09:17 bin
```

D.5. The mail command: standard UNIX manual

```
MAIL(1)              UNIX Programmer's Manual              MAIL(1)


NAME
     mail - send and receive mail

SYNOPSIS
     mail [ -f [ name ] ] [ people ...  ]

INTRODUCTION
     Mail is a intelligent mail processing system, which has a
     command syntax reminiscent of ed with lines replaced by mes-
     sages.

     Sending mail.  To send a message to one or more other peo-
     ple, mail can be invoked with arguments which are the names
     of people to send to.  You are then expected to type in your
     message, followed by an EOT (control-D) at the beginning of
     a line.  The section below, labeled Replying to or originat-
     ing mail, describes some features of mail available to help
     you compose your letter.

     Reading mail.  In normal usage, mail is given no arguments
     and checks your mail out of the post office, then printing
     out a one line header of each message there.  The current
     message is initially the first message (numbered 1) and can
     be printed using the print command (which can be abbreviated
     p).  You can move among the messages much as you move
     between lines in ed, with the commands '+' and '-' moving
     backwards and forwards, and simple numbers typing the
     addressed message.

     Disposing of mail.  After examining a message you can delete
     (d) the message or reply (r) to it.  Deletion causes the
     mail program to forget about the message.  This is not
     irreversible, the message can be undeleted (u) by giving its
     number, or the mail session can be aborted by giving the
     exit (x) command.  Deleted messages will, however, usually
     disappear never to be seen again.

     Specifying messages.  Commands such as print and delete
     often can be given a list of message numbers as argument to
     apply to a number of messages at once.  Thus ''delete 1 2''
     deletes messages 1 and 2, while ''delete 1-5'' deletes mes-
     sages 1 through 5.  The special name ''*'' addresses all
     messages, and ''$'' addresses the last message; thus the
     command top which prints the first few lines of a message
     could be used in ''top *'' to print the first few lines of
     all messages.

     Replying to or originating mail.  You can use the reply com-
     mand to set up a response to a message, sending it back to
     the person who it was from.  Text you then type in, up to an
     end-of-file (or a line consisting only of a ''.'') defines
     the contents of the message.  While you are composing a mes-
     sage, mail treats lines beginning with the character '~'
     specially.  For instance, typing ''~m'' (alone on a line)
     will place a copy of the current message into the response
     right shifting it by a tabstop.  Other escapes will set up
```

subject fields, add and delete recipients to the message and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options will be given in the summary below.)

Ending a mail processing session. You can end a mail session with the quit (q) command. Messages which have been examined go to your mbox file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. The -f option causes mail to read in the contents of your mbox (or the specified file) for processing; when you quit mail writes undeleted messages back to this file.

Personal and systemwide distribution lists. It is also possible to create a personal distribution lists so that, for instance, you can send mail to ''cohorts'' and have it go to a group of people. Such lists can be defined by placing a line like

```
     alias cohorts bill ozalp sklower jkf mark cory:kridle
```

in the file .mailrc in your home directory. The current list of such aliases can be displayed by the alias (a) command in mail. System wide distribution lists can be created by editing /usr/lib/aliases, see aliases(5) and delivermail(8); these are kept in a slightly different syntax. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to reply to the recipients. System wide aliases are not expanded when the mail is sent, but any reply returned to the machine will have the system wide alias expanded as all mail goes through delivermail. If you edit /usr/lib/aliases, you must run the program newaliases(1).

Network mail (ARPA, UUCP, Berknet) Mail to sites on the ARPA network and sites within Bell laboratories can be sent using ''name@site'' for ARPA-net sites or ''machine!user'' for Bell labs sites, provided appropriate gateways are known to the system. (Be sure to escape the ! in Bell sites when giving it on a csh command line by preceding it with an \. Machines on an instance of the Berkeley network are addressed as ''machine:user'', e.g. ''csvax:bill''. When addressed from the arpa-net, ''csvax:bill'' is known as ''csvax.bill@berkeley''.

Mail has a number of options which can be set in the .mailrc file to alter its behavior; thus ''set askcc'' enables the ''askcc'' feature. (These options are summarized below.)

SUMMARY

(Adapted from the 'Mail Reference Manual') Each command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety - the first command which matches the typed prefix is used. For the commands which take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, mail types ''No applicable messages'' and aborts the command.

-   Goes to the previous message and prints it out. If given a numeric argument n , goes to the n th

	previous message and prints it.
?	Prints a brief summary of commands.
!	Executes the UNIX shell command which follows.
alias	(a) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, adds the users named in the second and later arguments to the alias named in the first argument.
chdir	(c) Changes the user's working directory to that specified, if given. If no directory is given, then changes to the user's login directory.
delete	(d) Takes a list of messages as argument and marks them all as deleted. Deleted messages will not be saved in mbox , nor will they be available for most other commands.
dp	(also dt) Deletes the current message and prints the next message. If there is no next message, mail says ''at EOF.''
edit	(e) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.
exit	(ex or x) Effects an immediate return to the Shell without modifying the user's system mailbox, his mbox file, or his edit file in -f .
from	(f) Takes a list of messages and prints their message headers.
headers	(h) Lists the current range of headers, which is an 18 message group. If a ''+'' argument is given, then the next 18 message group is printed, and if a ''-'' argument is given, the previous 18 message group is printed.
help	A synonym for ?
hold	(ho, also preserve) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in mbox. Does not override the delete command.
mail	(m) Takes as argument login names and distribution group names and sends mail to those people.
next	(n like + or CR) Goes to the next message in sequence and types it. With an argument list. types the next matching message.
preserve	A synonym for hold.
print	(p) Takes a message list and types out each message on the user's terminal.
quit	(q) Terminates the session, saving all undeleted, unsaved messages in the user's mbox file in his login directory, preserving all mes-

	sages marked with hold or preserve or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message ''You have new mail'' is given. If given while editing a mailbox file with the -f flag, then the edit file is rewritten. A return to the Shell is effected, unless the rewrite of edit file fails, in which case the user can escape with the exit command.
reply	(r) Takes a message list and sends mail to each message author just like the mail command. The default message must not be deleted.
respond	A synonym for reply .
save	(s) Takes a message list and a filename and appends each message in turn to the end of the file. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.
set	(se) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form ''option=value'' or ''option.''
shell	(sh) Invokes an interactive version of the shell.
size	Takes a message list and prints out the size in characters of each message.
top	Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable toplines and defaults to five.
type	(t) A synonym for print .
unalias	Takes a list of names defined by alias commands and discards the remembered groups of users. The group names no longer have any significance.
undelete	(u) Takes a message list and marks each one as not being deleted.
unset	Takes a list of option names and discards their remembered values; the inverse of set .
visual	(v) Takes a message list and invokes the display editor on each message.
write	(w) A synonym for save .
xit	(x) A synonym for exit .

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name ''tilde escape'' is somewhat of a misnomer since the actual escape character can be set by the option escape.

~!command	Execute the indicated shell command, then return to the message.

```
~c name ... Add the given names to the list of carbon copy
            recipients.

~d          Read the file ''dead.letter'' from your home
            directory into the message.

~e          Invoke the text editor on the message collected
            so far.  After the editing session is finished,
            you may continue appending text to the message.

~h          Edit the message header fields by typing each
            one in turn and allowing the user to append text
            to the end or modify the field by using the
            current terminal erase and kill characters.

~m messages Read the named messages into the message being
            sent, shifted right one tab.  If no messages are
            specified, read the current message.

~p          Print out the message collected so far, prefaced
            by the message header fields.

~q          Abort the message being sent, copying the mes-
            sage to ''dead.letter'' in your home directory
            if save is set.

~r filename Read the named file into the message.

~s string   Cause the named string to become the current
            subject field.

~t name ... Add the given names to the direct recipient
            list.

~v          Invoke an alternate editor (defined by the
            VISUAL option) on the message collected so far.
            Usually, the alternate editor will be a screen
            editor.  After you quit the editor, you may
            resume appending text to the end of your mes-
            sage.

~w filename Write the message onto the named file.

~|command   Pipe the message through the command as a
            filter.  If the command gives no output or ter-
            minates abnormally, retain the original text of
            the message.  The command fmt(1) is often used
            as command to rejustify the message.

~~string    Insert the string of text in the message pre-
            faced by a single ~.  If you have changed the
            escape character, then you should double that
            character in order to send it.

Options are controlled via the set and unset commands.
Options may be either binary, in which case it is only sig-
nificant to see whether they are set or not, or string, in
which case the actual value is of interest.  The binary
options include the following:

append         Causes messages saved in mbox to be appended
               to the end rather than prepended.  (This is
               set in /usr/lib/Mail.rc on version 7 sys-
               tems.)
```

ask	Causes mail to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field will be sent.
askcc	Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.
autoprint	Causes the delete command to behave like dp - thus, after deleting a message, the next one will be typed automatically.
ignore	Causes interrupt signals from your terminal to be ignored and echoed as s.
metoo	Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.
quiet	Suppresses the printing of the version when first invoked.
save	Causes the message collected prior to a interrupt to be saved on the file ''dead.letter'' in your home directory on receipt of two interrupts (or after a ~q.)

The following options have string values:

EDITOR	Pathname of the text editor to use in the edit command and ~e escape. If not defined, then a default editor is used.
SHELL	Pathname of the shell to use in the ! command and the ~! escape. A default shell is used if this option is not defined.
VISUAL	Pathname of the text editor to use in the visual command and ~v escape.
escape	If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.
record	If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so saved.
toplines	If defined, gives the number of lines of a message to be printed out with the top command; normally, the first five lines are printed.

FILES

/usr/spool/mail/*	post office
~/mbox	your old mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
/bin/mail	to do actual mailing
/etc/delivermail	postman

```
SEE ALSO
     binmail(1), fmt(1), newaliases(1), aliases(5), deliver-
     mail(8)
     'The Mail Reference Manual'


AUTHOR
     Kurt Shoens

BUGS
```

D.6. The mail command: Sobell/ACRONYM Version

```
mail: Send or receive mail
Format-1: mail user-list
Format-2: mail [options]

The first format sends mail to the user-list.  The second format displays
mail that you have received and prompts you with a ? following each letter.

Responses to ?
        ?       help
        d       delete mail
        q       quit
        w       write to mbox
        w file  write to named file
        RETURN  proceed to next letter
        p       redisplay previous letter
Options:
        -p      display mail, no questions
        -q      quit on interrupt
        -r      reverse order

SUMMARY OF THE MAIL COMMAND

mail sends and receives mail between users.  When you log on, the UNIX
system informs you if another user has sent you mail.

Use the first format ("mail user-list") to send mail to other users.  When
sending mail, mail accepts text from the standard input.  This input can be
redirected from a file or entered at the terminal.  If you send mail from
the terminal, it must be terminated by a CONTROL-D or a line with just a
period on it.  If you interrupt (DEL/RUBOUT/CONTROL-C) mail when you are
entering text at the terminal, the mail will not be sent.

Use the second format ("mail [options]") to display mail that you have
received.  When displaying mail, the mail program prompts you with a
question mark following each piece of mail.  The valid responses to this
prompt are discussed in the help text "Responses to the '?' prompt in the
mail utility", which is one of the entries in your current help menu.

OPTIONS FOR THE MAIL COMMAND

The following options affect mail that you have received and are displaying.
They are not for use when you are sending mail.
        -p      (display mail, no questions) This option causes the mail
                program to display mail without prompting you after each
                piece of mail.
        -q      (quit on interrupt) Without this option, the interrupt key
                (usually DEL, RUBOUT, or CONTROL-C) stops mail from
                displaying the current piece of mail and allows it to
                proceed with the next.  When you use this option, an
                interrupt will stop execution of the mail program and return
                you to the Shell without changing the status of your mail.
        -r      (reverse order) Without this option, mail is displayed in a
                last-piece-of-mail-received, first-piece-of-mail-displayed
                order.  This option causes mail to display mail in
                chronological order.

ARGUMENTS FOR THE MAIL COMMAND

The user-list contains the User ID names of the users you want to send mail
to.

RESPONSES TO THE '?' PROMPT IN THE MAIL UTILITY
```

The following are valid responses to the '?' prompt that is printed after each piece of mail that the mail utility shows you.

```
?         A question mark causes mail to display a summary of valid responses.
!<cmd>    This response causes mail to exit to the Shell, execute whatever
          command you typed, and return to the mail program when the command
          has finished executing.
CONTROL-D This response causes the mail program to stop and leave unexamined
          mail in the mailbox so that you can look at it the next time you
          run mail.
d         A d causes the mail program to delete the piece of mail it just
          displayed, and proceed to the next one.
m[name]   This response causes the mail program to remail the piece of mail to
          the specified person or people.  If you do not specify a person,
          the mail is sent back to you again.
RETURN    Press the RETURN key to proceed to the next piece of mail.
p         A p causes the mail program to redisplay the previous piece of mail.
q         A q has the same effect as CONTROL-D; you exit the mail program and
          unexamined mail is saved until next time.
s[file]   This response causes the mail program to save the piece of mail in
          the named file or "mbox" if you do not specify a filename.
w[file]   This response causes mail to save the piece of mail, without a
          header, in the named file or "mbox" if you do not specify a filename.
x         This response causes the mail program to exit and not change the
          status of your mail.
```

EXAMPLES OF THE MAIL COMMAND

The first example below shows how to send a message to several users. In this case, mail sends the message to users with the login names of hls, alex, and jenny.

```
$ mail hls alex jenny
(message)
(message)
.
```

You can also compose a message in a file and then send it by redirecting the input to mail. The command below sends the file today to barbara.

```
$ mail barbara < today
```

D.7. The rm command: standard UNIX manual

```
RM(1)                UNIX Programmer's Manual                RM(1)


NAME
     rm, rmdir  - remove (unlink) files

SYNOPSIS
     rm [ -e ] [ -f ] [ -r ] [ -i ] [ - ] file ...

     rmdir dir ...

DESCRIPTION
     rm removes the entries for one or more files from a direc-
     tory.  If an entry was the last link to the file, the file
     is destroyed.  Removal of a file requires write permission
     in its directory, but neither read nor write permission on
     the file itself.

     If a file has no write permission and the standard input is
     a terminal, its permissions are printed and a line is read
     from the standard input.  If that line begins with 'y' the
     file is deleted, otherwise the file remains.  No questions
     are asked and no errors are reported when the -f (force)
     option is given.

     If a designated file is a directory, an error comment is
     printed unless the optional argument -r has been used.  In
     that case, rm recursively deletes the entire contents of the
     specified directory, and the directory itself.

     If the -i (interactive) option is in effect, rm asks whether
     to delete each file, and, under -r, whether to examine each
     directory.

     If the -f option is not in effect, rm checks the extensions
     in the environment variable RMEXTS (see below) against the
     final extensions of each file, and asks whether to delete
     each file whose extension matches.  This is to prevent
     accidental deletion of source code and other important
     files.

     The null option - indicates that all the arguments following
     it are to be treated as file names.  This allows the specif-
     ication of file names starting with a minus.

     The option -e will print out the path of each file or direc-
     tory as it is removed.  This is useful in shell scripts.

     rmdir removes entries for the named directories, which must
     be empty.

ENVIRONMENT
     RMEXTS is a list of extensions that you would like to pro-
     tect, using commas and/or spaces as separators.

EXAMPLES
     Each of the following examples would protect files with any
     of the extensions ".c", ".h", ".mss", ".p", ".pas":
          RMEXTS="c,h,mss,p,pas"; export RMEXTS
          RMEXTS="c h mss p pas"; export RMEXTS
          RMEXTS="c , h , mss , p , pas"; export RMEXTS

SEE ALSO
     unlink(2)

DIAGNOSTICS
     Generally self-explanatory.  It is forbidden to remove the
     file '..' merely to avoid the antisocial consequences of
     inadvertently doing something like 'rm -r .*'.

HISTORY
     12-Nov-83  Neal Friedman (naf) at Carnegie-Mellon University
          Added description of -e option, which was provided by
          John Schlag.

     20-Jan-82  John Wicks (jrw) at Carnegie-Mellon University
          Modified to describe check for protected file exten-
          sions set in the environment.
```

D.8. The rm command: Sobell/ACRONYM Version

```
rm: Delete a file (remove a link)
Format: rm [options] file-list
Options: -f force
         -i interactive
         -r recursive
```

SUMMARY OF THE RM COMMAND

rm removes links to one or more file. When you remove the last link, you can no longer access the file and the system releases the space the file occupied on the disk for use by another file (i.e. the file is deleted).

To delete a file, you must have execute and write access permission to the parent directory of the file, but you do not need read or write access permission to the file itself. If you are running rm from a terminal (i.e. rm's standard input is coming from a terminal) and you do not have write access permission to the file, rm displays your access permission and waits for you to respond. If you enter "y" or "yes", rm deletes the file; otherwise it does not. If the standard input is not coming from the terminal, rm deletes the file without question.

OPTIONS FOR THE RM COMMAND

There are three options to the rm command:

-f (force) This option causes rm to remove files for which you do not have write access permission, without asking for your consent. It can also be used to override protections given by the RMEXTS variable, which is described in the section "Notes on the rm command".

-i (interactive) This option causes rm to ask you before removing each file. If you use the -r option with this option, rm also asks you before examining each directory. When you use the -i option with the * wildcard, rm can delete files with characters in their filenames that prevent you from deleting the files by other means.

-r (recursive) This option causes rm to delete the contents of the specified directory and the directory itself. Use this option cautiously.

ARGUMENTS FOR THE RM COMMAND

The arguments to rm are a file-list that contains the list of files that rm deletes. The list can include wildcards. Because you can remove a large number of files with a single command, use rm with wildcards cautiously. If you are in doubt as to the effect of an rm command with a wildcard, use the echo utility with the same wildcard first. echo displays the list of files that rm will delete.

ADDITIONAL NOTES ON THE RM COMMAND

Because rm will happily delete your most important files without a complaint, the local version of rm has a feature that allows you to safeguard certain types of files. If, in your .login or .profile, you set the environment variable "RMEXTS" to be a list of filename extensions, then rm will always ask for confirmation before deleting filenames with those extensions (unless the -f option is specified). Thus, if RMEXTS is set to "c p a l h mss", then no file whose name ends in ".c", ".p", ".a", ".l", ".h", or ".mss" will be deleted without confirmation.

EXAMPLES OF THE RM COMMAND

The following command lines delete files, both in the current working directory and in another directory.

```
$ rm memo
$ rm letter memo1 memo2
$ rm /usr/jenny/temp
```

The following command deletes the directory "useless" and all its contents, including the contents of any subdirectories. It should only be used if you are absolutely positive you want to delete all those files.

```
$ rm -r useless
```

D.9. The sort command: standard UNIX manual

```
SORT(1)              UNIX Programmer's Manual              SORT(1)


NAME
     sort - sort or merge files

SYNOPSIS
     sort [ -mubdfinrtx ] [ +pos1  [ -pos2 ] ] ...  [ -o name ] [
     -T directory ] [ name ] ...

DESCRIPTION
     Sort sorts lines of all the named files together and writes
     the result on the standard output.  The name '-' means the
     standard input.  If no input files are named, the standard
     input is sorted.

     The default sort key is an entire line.  Default ordering is
     lexicographic by bytes in machine collating sequence.  The
     ordering is affected globally by the following options, one
     or more of which may appear.

     b    Ignore leading blanks (spaces and tabs) in field com-
          parisons.

     d    'Dictionary' order: only letters, digits and blanks are
          significant in comparisons.

     f    Fold upper case letters onto lower case.

     i    Ignore characters outside the ASCII range 040-0176 in
          nonnumeric comparisons.

     n    An initial numeric string, consisting of optional
          blanks, optional minus sign, and zero or more digits
          with optional decimal point, is sorted by arithmetic
          value.  Option n implies option b.

     r    Reverse the sense of comparisons.

     tx   'Tab character' separating fields is x.

     The notation +pos1 -pos2 restricts a sort key to a field
     beginning at pos1 and ending just before pos2.  Pos1 and
     pos2 each have the form m.n, optionally followed by one or
     more of the flags bdfinr, where m tells a number of fields
     to skip from the beginning of the line and n tells a number
     of characters to skip further.  If any flags are present
     they override all the global ordering options for this key.
     If the b option is in effect n is counted from the first
     nonblank in the field; b is attached independently to pos2.
     A missing .n means .0; a missing -pos2 means the end of the
     line.  Under the -tx option, fields are strings separated by
     x; otherwise fields are nonempty nonblank strings separated
     by blanks.

     When there are multiple sort keys, later keys are compared
     only after all earlier keys compare equal.  Lines that oth-
     erwise compare equal are ordered with all bytes significant.
     These option arguments are also understood:

     c    Check that the input file is sorted according to the
          ordering rules; give no output unless the file is out
          of sort.

     m    Merge only, the input files are already sorted.

     o    The next argument is the name of an output file to use
          instead of the standard output.  This file may be the
          same as one of the inputs.

     T    The next argument is the name of a directory in which
          temporary files should be made.

     u    Suppress all but one in each set of equal lines.
          Ignored bytes and bytes outside keys do not participate
          in this comparison.

     Examples. Print in alphabetical order all the unique spel-
     lings in a list of words.  Capitalized words differ from
     uncapitalized.

             sort -u +0f +0 list

     Print the password file (passwd(5)) sorted by user id number
     (the 3rd colon-separated field).

             sort -t: +2n /etc/passwd

     Print the first instance of each month in an already sorted
     file of (month day) entries.  The options -um with just one
     input file make the choice of a unique representative from a
     set of equal lines predictable.

             sort -um +0 -1 dates

FILES
     /usr/tmp/stm*, /tmp/*   first and second tries for temporary
     files

SEE ALSO
     uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS
     Comments and exits with nonzero status for various trouble
     conditions and for disorder discovered under option -c.

BUGS
     Very long lines are silently truncated.
```

D.10. The sort command: Sobell/ACRONYM Version

```
sort: Sort and/or merge files
Format: sort [options] [field-specifier-list] [file-list]
Options: -b      (blanks) ignore leading blanks
         -c      (check) check for proper sorting only
         -d      (dictionary) ignore nonalphanumerical and blank characters
         -f      (fold) sort uppercase letters as though they were lowercase
         -m      (merge) merge only, assume sorted order
         -n      (numeric) minus signs and decimal points take on their
                         arithmetic value (implies -b)
         -o      (output) must be followed by output filename
         -r      (reverse) reverse sense of sort
         -tx     (tab) x is input field delimiter
         -u      (unique) do not display a repeated line more than once

Arguments: field-specifier-list specifies input fields by pairs of pointers
        as +f.c -f.c where f is the number of fields to skip and c is the
        number of characters to skip.

SUMMARY OF THE SORT COMMAND

The sort utility sorts and/or merges one or more text files in sequence.
When you use the -n option, sort performs a numeric sort.

Sort takes its input from files specified on the command line or from the
standard input.  Unless you use the -o option, output from sort goes to the
standard output.

OPTIONS FOR THE SORT COMMAND

If you do not specify an option, sort orders the file in the machine
collating (ASCII) sequence.  You can embed options within the
field-specifier-list.  The options are:

-b      (ignore leading blanks) Blanks (TAB and SPACE characters) are
        normally field delimiters in the input file.  Unless you use
        this option, sort also considers leading blanks to be part of the
        field they precede.  This option causes sort to consider multiple
        blanks as field delimiters, with no intrinsic value, so that sort
        does not consider these characters in sort comparisons.
-c      (check only) This option causes the sort to check to see that the
        file is properly sorted.  sort does not display anything if
        everything is in order.  sort displays a message if the file is not
        in sorted order.
-d      (sort in dictionary order) This option causes sort to ignore all
        characters that are not alphanumeric characters or blanks.
        Specifically, sort does not consider punctuation and CONTROL
        characters.
-f      (fold uppercase into lowercase) This option causes sort to consider
        all uppercase letters to be lowercase letters.  use this option when
        you are sorting a file that contains both uppercase and lowercase
        text.
-m      (merge) This option causes the sort to assume that multiple input
        files are in sorted order.  sort merges these files without verifying
        that they are sorted.
-n      (numeric sort) When you use this option, minus signs and decimal
        points take on their arithmetic meaning and the -b option is implied.
        sort does not order lines or sort fields in the machine collating
        sequence, but rather in arithmetic order.
-o      (specify output file) You must place a filename after this option on
        the command line.  sort sends its output to this file instead of to
        the standard output.
-r      (reverse sense) This option reverses the sense of the sort (e.g., z
```

	precedes a).
-tx	(set tab character) When you use this option, replace the x with the character that is the field delimiter in the input file. This new character replaces blanks, which become regular (nondelimiting) characters.
-u	(unique lines) This option causes sort to output repeated lines only once. sort outputs lines that are not repeated as it would without this option.

ARGUMENTS FOR THE SORT COMMAND

The arguments to the sort program are all optional; they can include a field-specifier list and/or a file-list. The field-specifier-list selects one or more sort-fields within each line to be sorted. The sort utility uses the sort-fields to sort the lines. (See "Detailed Description of the sort Command" for more details.) The file-list contains pathnames of one or more plain files that contain the text to be sorted. sort sorts and merges the files unless you use the -m option, in which case, sort only merges the files.

DETAILED DESCRIPTION OF THE SORT COMMAND

In the following description, a line-field is a sequence of characters on a line in the input field. These sequences are bounded by blanks and the beginning and end of the line. Line-fields are used to define a sort field.

A sort-field is a sequence of characters that sort uses to put lines in order. The description of a sort-field is based on line-fields. A sort-field can contain part or all of one or more line-fields. In the following line,

```
Toni Barnett          95020
```

the three line-fields are "Toni", " Barnett", and "   95020". Possible sort-fields in this line might be "Barnett", "020", or any other portion of a line.

The field-specifier-list contains pairs of pointers that define subsections of each line (sort-fields) for comparison. If you omit the second pointer from a pair, sort assumes the end of the line. A pointer is in the form "+f.c" or "-f.c". the first of each pair of pointers begins with a plus sign, while the second begins with a hyphen.

You can make a pointer (f.c) point to any character on a line. The f is the number of line-fields you want to skip, counting from the beginning of the line. The c is the number of characters you want to skip, counting from the end of the last line-field you skipped with the f.

The -b option causes sort to count multiple leading blanks as a single line-field delimiter character. If you do not use this option, sort considers each leading blank to be a character in the sort-field, and includes it in the sort comparison.

You can specify options that pertain only to a given sort-field by immediately following the field specifier by one of the options b, d, f, i, n, or r. In this case, you must NOT precede the options with a hyphen.

If you specify more than one sort-field, sort examines them in the order that you specify them on the command line. If the first sort-field of two-lines is the same, sort examines the second sort-field. If these are again the same, sort looks at the third field. This process continues for all the sort-fields you specify. If all the sort-fields are the same, sort examines the entire line.

If you do not use any options or arguments, the sort is based on entire lines.

EXAMPLES OF THE SORT COMMAND

The following examples assume that a file named list is in the working directory. All the blanks are SPACEs, not TABs.

```
$ cat list
Tom Winstrom        94201
Janet Dempsey       94111
Alice MacLeod       94114
David Mack          94114
Toni Barnett        95020
Jack Cooper         94072
Richard MacDonald   95510
$
```

The first example demonstrates sort without any options or arguments, other than a filename. sort sorts the file on a line-by-line basis. If the first characters on two lines are the same, sort looks at the second characters to determine the proper sorted order. If the second characters are the same, sort looks at the third characters. This process continues until sort finds a character that differs between the lines. If the lines are identical, it doesn't matter which one sort puts first. The sort command in this example only needs to examine the first three letters (at most) of each line. Sort displays a list that is in alphabetical order by first name.

```
$ sort list
Alice MacLeod       94114
David Mack          94114
Jack Cooper         94072
Janet Dempsey       94111
Richard MacDonald   95510
Tom Winstrom        94201
Toni Barnett        95020
```

sort can skip any number of line-fields and characters on a line before beginning its comparison. Blanks normally separate one line-field from another. The next example sorts the same list by last name, the second line-field. The +1 argument indicates that sort is to skip one line-field before beginning its comparison. It skips the first-name field. Because there is no second pointer, the sort-field extends to the end of the line. Now the list is almost in last name order, but there is a problem with "Mac".

```
$ sort +1 list
Toni Barnett        95020
Jack Cooper         94072
Janet Dempsey       94111
Richard MacDonald   95510
Alice MacLeod       94114
David Mack          94114
Tom Winstrom        94201
```

In the example above, MacLeod comes before Mack. sort found the sort-fields of these two files the same through the third letter ("Mac"). Then it put L before k because it arranges lines in the order of ASCII (or other) character codes. In this ordering, uppercase letters come before lowercase ones and therefore L comes before k.

The -f option makes sort treat uppercase and lowercase letters as equals, and thus fixes the problem with MacLeod and Mack.

```
$ sort -f +1 list
Toni Barnett        95020
Jack Cooper         94072
```

```
Janet Dempsey        94111
Richard MacDonald    95510
David Mack           94114
Alice MacLeod        94114
Tom Winstrom         94201
```

The next example attempts to sort list on the third line-field, the zip code. sort does not put the numbers in order, but puts the shortest name first in the sorted list and the longest name last. With the argument of +2, sort skips two line-fields and counts the SPACEs after the second line-field (last name) as part of the sort-field. The ASCII value of a SPACE character is less than that of any other printable character, so sort puts the zip code that is preceded by the greatest number of SPACEs first, and the zip code that is preceded by the fewest SPACEs last.

```
$ sort +2 list
David Mack           94114
Jack Cooper          94072
Tom Winstrom         94201
Toni Barnett         95020
Janet Dempsey        94111
Alice MacLeod        94114
Richard MacDonald    95510
```

The -b option causes sort to ignore leading SPACEs. With the -b option, the zip codes come out in the proper order (see following example).

When sort determines that MacLeod and Mack have the same zip code, it compares the entire lines. The Mack/MacLeod problem crops up again because the -f option is not used.

```
$ sort -b +2 list
Jack Cooper          94072
Janet Dempsey        94111
Alice MacLeod        94114
David Mack           94114
Tom Winstrom         94201
Toni Barnett         95020
Richard MacDonald    95510
```

The next example shows a sort command that not only skips line-fields, but skips characters as well. The +2.3 causes sort to skip two line-fields and then skip three characters before starting its comparisons. The sort-field is, and the list below is sorted in order of, the last two digits in the zip code. (The -f option is included to take care of MacLeod and Mack.)

```
$ sort -f -b +2.3 list
Tom Winstrom         94201
Richard MacDonald    95510
Janet Dempsey        94111
Alice MacLeod        94114
David Mack           94114
Toni Barnett         95020
Jack Cooper          94072
```

The next example uses a different file, list2, to demonstrate the -n option. This file contains text that represents numbers and includes minus signs and decimal points.

```
$ cat list2
.7
1.1
-11
10.0
0.5
```

```
-1.1
$
```

When sort processes this list of numbers, it sorts them according to the machine collating sequence. They are not put in arithmetic order.

```
$ sort list2
-1.1
-11
.7
0.5
1.1
10.0
```

The -n option causes sort to put the list of numbers -- including symbols -- in its proper, arithmetic sequence.

```
$ sort -n list2
-11
-1.1
0.5
.7
1.1
10.0
```

The final sort example demonstrates a more complex use of options and arguments. cat displays the "words" file used in this example.

```
$ cat words
apple
pear
peach
apple
Apple
Pear
prune
Plum
peach
orange
pear
plum
pumpkin
$
```

The following sort command sorts "words" and displays only one copy of each line (-u option). The arguments cause sort to evaluate each line twice.

```
$ sort -u +0f +0 words
Apple
apple
orange
peach
Pear
pear
Plum
plum
prune
pumpkin
```

Just as the argument +1 causes sort to skip one line-field, +0 causes sort to skip zero fields; sort examines the first line-field.

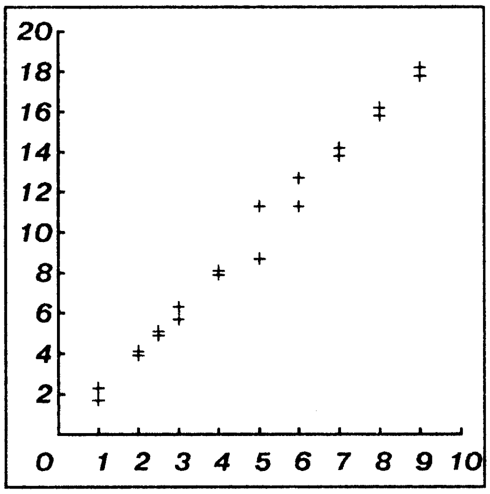
The +0f argument causes sort to evaluate the first line-field through the end of the line (the entire word) as though it were lowercase. The -f option is used to fold uppercase into lowercase and is not preceded by a hyphen because it follows a field specifier. The second argument, +0, evaluates each word, differentiating between uppercase and lowercase letters. The result is a list of all the words, in alphabetical order, differentiating between upper- and lowercase letters, displaying only one copy of each word.

Appendix E
A Brief Introduction to Regression Analysis

In this appendix, I will outline the basic ideas of regression analysis as they are relevant to interpreting the data presented in this thesis. This is not intended to serve as a general introductory text on regression analysis; for that, any good introductory statistics textbook should suffice. Rather, the idea here is to present an intuitive picture of regression as it is used in this thesis.

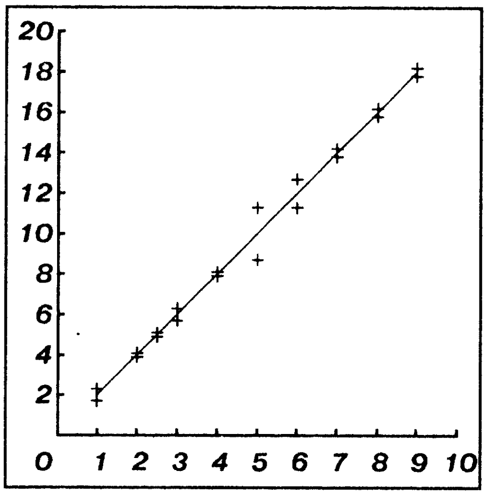
E.1. Simple regression

Often, in collecting data regarding a simple phenomenon, the data will suggest an underlying trend that is hard to define directly from the data. In this picture, for example, there appears to be a clear relationship between the x and y values graphed:



It seems likely, looking at this data, that there is a direct relationship between the two variables graphed. It is generally useful to express this relationship as a function Y = f(x), but it is not always obvious what that function is. Regression is a technique which, given a certain class of functions, finds the one which "best fits" the data. In particular, linear regression, which is most commonly used, only examines those functions of the form $Y = c + kx$, so that the purpose of the regression process is to select the best values for the constants c and k.

The process by which this selection is made is called the *method of least squares*, which is explained in any standard statistics textbook. Suffice it to say that the technique finds the best *linear* equation for the data, where "best" means that the sum of the squares of the differences on the y axis are minimized. In the following picture, the data shown previously is shown along with a graph of its regression equation:



Here, the equation of the regression line is simply $y=2x$.

The idea behind this method is that the regression equation approximates the "real" underlying relationship between the variables, which can be obscured by experimental error and variation in the data. The danger of using this technique is that regression equations can be arbitrarily poor fits for the data; even if there is no real relationship, the regression equation will find a "best" relationship. However, the standard deviation of a coefficient in the regression equation is a simple and reliable measure of the significance of the effect of the associated independent variable. In trying to understand whether or not a given coefficient's value indicates a significant effect, we must compare the size of the coefficient with its standard deviation. The ratio of the coefficient to the standard deviation is know as the *T-ratio*. The larger the T-ratio, the more confident we can be about the significance of the effect observed. For example, a T-ratio of 1.95 or more indicates a confidence level of 95% or more; that is, the chances of the observed effect being due merely to chance (coincidence) is less than 1 in 20.

E.2. Multivariate Regression

In genuinely interesting applications, researchers rarely study phenomena so simple that a single dependent variable and a single independent variable are the only ones involved. Instead, they will typically have several independent variables and one or more dependent variables. In the discussion that follows, we will assume only one dependent variable, y, and n independent variables, x_1, x_2, ... x_n. The goal of the linear regression analysis is to produce an equation of the form:

$$y = c_0 + c_1x_1 + c_2x_2 + ... + c_nx_n$$

The method used here is completely analogous to the two-dimensional method outlined in the previous section, except that the computation is more difficult. Again, the values of the constants c_i are selected to best model the data, and their standard deviations provide a measure of their significance.

E.3. Indicator Variables

The situation becomes slightly further complicated when one of the variables observed does not take on continuous numeric values. For example, in the experiments described in this thesis, one of the independent variables was the subject; obviously, different subjects performed differently, and it was important to be able to differentiate the effect of subject variation from the effect of help system variation. Unfortunately, simply giving each subject a numeric value such as "1" or "17" is unlikely to lead to a meaningful regression equation, given that the numbers assigned are arbitrary.

Instead, we can create *indicator variables* for each of the different subjects. Thus, in an experiment with four subjects, we might create four indicator variables S_1, S_2, S_3, and S_4. For each subject, one indicator variable will be 1 and the rest will be 0. Thus, for example, for the third subject, S_1, S_2, and S_4 would be 0, while S_3 would be 1. Given that only one of these four variables is non-zero, the coefficient of each of these variables in the final regression equation is an indication of the effect of the relevant discrete condition -- in this case, the effect of a certain subject -- on the dependent variable.

For a very simple example, imagine an experiment in which we observe three people, John, Mary, and Bill, each eating two scoops of ice cream, chocolate and vanilla. If we let x_1 be an indicator variable for John, x_2 for Mary, x_3 for Bill, x_4 for chocolate, and x_5 for vanilla, we might imagine that an equation of the following form might predict the time t that it takes an individual to eat a scoop of ice cream:

$$t = c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 + c_5x_5$$

However, for technical reasons, this is not precisely the way such analyses are conducted. It turns out that, to obtain significant and meaningful results, not all of the indicator variables should be included in the regression equation; rather, one of each set of indicator variables should be omitted to allow the analysis a sufficient number of degrees of freedom. Thus, more realistically, a regression equation for the experiment just described might look more like this:

$$t = c_0 + c_1x_1 + c_2x_2 + c_3x_3$$

Here, the indicator variable x_1 is 1 if the subject is John, and x_2 is 1 if the subject is Mary; if both x_1 and x_2 are zero, the subject is Mary. Similarly, x_3 is 1 for chocolate and 0 for vanilla. In general, in such an equation, the constant c_0 reflects the predicted value of the dependent variable when all of the *omitted* indicator variables are 1. In this case, c_0 is the predicted time it would take for Bill to eat vanilla ice cream. A complete table of the possible values of the variables and their meaning is found in the Table E-1. Note that certain indicator variables are mutually exclusive; it is impossible for a subject to be both John and Mary, for example, regardless of what flavor the ice cream is.

Table E-1: Meanings of Indicator Variables in Toy Regression Example

x_1	x_2	x_3	Meaning
0	0	0	Bill eats vanilla.
0	0	1	Bill eats chocolate.
0	1	0	Mary eats vanilla.
0	1	1	Mary eats chocolate.
1	0	0	John eats vanilla.
1	0	1	John eats chocolate.
1	1	0	Impossible situation.
1	1	1	Impossible situation.

E.4. Reading the regression results from Chapter 7

In several different tables in Chapter 7, the results of a multivariate regression analysis are displayed. The first part of each of these tables is the regression equation, which should be interpreted as described above. For each of these equations, the baseline help system, the first subject, and the first task do not have indicator variables; thus the constant in the equation predicts the log time for the first subject using the first task and the first help system. (Log time is used instead of actual time to reduce the effect of minor variations.)

After this equation, which summarizes the regression results, comes a table of values with one line for each of the regression variables. Each such line gives the variable, its meaning as an indicator variable, its coefficient from the regression equation, and the "T ratio". The T ratio is simply the regression coefficient divided by its standard deviation. This ratio gfives a measure of the signficance of the coefficient; anything with absolute value over 1.95 is significant with a confidence level p<.05 (95%). Thus, if the T ratio is over 1.95 it is reasonably certain that the condition indicated by that variable significantly increases the dependent variable -- in this case, the time to execute the task. (A T ratio over 1.65 indicates a confidence level p<.1 (90%), a rather less reliable indicator.) Similarly, a T ratio less than -1.95 indicates a significant decrease in task execution time.

The regression tables presented in this thesis show only the comparisons in which the indicator variable omitted is the man/key (baseline) help system. This raises the question of the comparability of the other help systems. Table 7-3, on page 89, for example, shows the following T-ratios for the coefficients corresponding to the other help systems:

System	T-ratio	Coeficient
Hybrid	-1.53	-0.2675
ACRONYM	-2.15	-0.3668
Tutor	-3.52	-0.6095
English	-1.65	-0.3010

This suggests that all of the systems are significantly better than the baseline system. However, the much larger T-ratio for the human tutor, -3.52, suggests that the tutor is significantly better than the other systems as well. The regression analysis as presented does NOT, however, prove that this is the case, but merely suggests it. However, it turns out that the trends suggested in this manner are generally borne out (for reasonable data) by further analysis. In this case, for example, running the regression with the human tutor as the omitted indicator variable yields the following T-ratios:

System	T-ratio	Coeficient
Baseline	3.52	0.6095

Hybrid	1.39	0.3420
ACRONYM	1.00	0.2427
English	1.27	0.3085

These T-ratios indicate a likely difference, though not as strongly as is suggested by the differences in the first table. The difference in significance is entirely an artifact of the experimental design. Remember that the T-ratio is simply the ratio of the coefficient to the standard deviation. Although the coefficients interact strictly linearly -- that is, the coefficient for ACRONYM, 0.2427, in the second table, is the difference between the ACRONYM and Tutor coefficients in the previous table -- the standard deviations do not. This is because there is more data for the baseline system (man/key) than for any other system, given its use as a baseline in the experiments. Since there is more data available for man/key than for other systems, it is natural that confidence levels be somewhat higher for those comparisons that involve it than for those that do not. Also, comparing, for example, ACRONYM to the tutor involves an extra bit of indirection in the data; no single subject actually used both of those two systems, so the comparison is inherently somewhat riskier.

Nonetheless, the different forms of the regression analysis tend to back up the conclusions of the initial analysis, albeit at slightly reduced levels of significance. For example, the most shaky finding, in this analysis, is that the human tutor is better than ACRONYM. This conclusion has a probability of just over 1 chance in 4 of being incorrect. It would very likely be made firmer by increasing the number of subjects studied or by running subjects with a more direct comparison between the two systems.

The regression analysis reported in this thesis was conducted using the MINITAB statistical analysis program [104]. For more information on regression analysis in general, consult an appropriate textbook.

Bibliography

[1] Ambrozy, Denise.
On Man-Computer Dialogue.
International Journal of Man-Machine Studies 3(4):375-383, 1971.

[2] Anderson, John R.
Learning to Program.
year unknown.

[3] Anderson, John R., Robert Farrell, and Ron Sauers.
Learning to Plan in LISP.
Technical Report, Carnegie-Mellon University Department of Psychology, 1982.

[4] Ball, E. and P. Hayes.
A Test-Bed for User Interface Designs.
In *Proceedings, Human Factors In Computer Systems.* March, 1982.

[5] Bannon, Liam and Claire O'Malley.
Problems in Evaluation of Human-Computer Interfaces: A Case Study.
HMI Project, University of California at San Diego, March, 1984.

[6] Bates, Madeleine, and John Vittal.
Tools for the Development of Systems for Human Factors Experiments: An Example for the SSA.
IEEE Transactions on Systems, Man, and Cybernetics SMC-12(2):133-148, March/April, 1982.

[7] Black, J. and T. Moran.
Learning and Remembering Command Names.
In *Proceedings, Human Factors In Computer Systems.* March, 1982.

[8] Borenstein, Nathaniel.
The Evaluation of Text Editors: A Critical Review of the Roberts and Moran Methodology Based on New Experiments.
In *Procedings of CHI '85.* 1985.

[9] Borenstein, Nathaniel and James Gosling.
UNIX Emacs as a Test-bed for User Interface Design.
1985.
in preparation.

[10] Bott, Ross A.
A Study in Complex Learning: Theory and Methodologies.
PhD thesis, U. C. San Diego, March, 1979.

[11] Bramwell, Bob.
BROWSE: An On-line Manual and System Without an Acronym.
SIGDOC Newsletter, 1984.

[12] Bramwell, Bob.
Browsing Around A Manual.
In *Canadian Information Processing Society Session '84 Proceedings*, pages 438-451. 1984.

[13] Campbell, Donald T., and Julian C. Stanley.
Experimental and Quasi-Experimental Designs for Research.
Houghton Mifflin Company, 1966.

[14] Card, Stuart K., Thomas P. Moran, and Allen Newell.
The Psychology of Human-Computer Interaction.
Lawwrence Erlbaum Associates, Hillsdale, NJ, 1983.

[15] Cherry, Lorinda L.
Computer Aids for Writers.
SIGPLAN Notices :62-67, June, 1981.

[16] Cherry, L. L. and W. Vesterman.
Writing Tools -- The STYLE and DICTION Programs.
year unknown.

[17] Christensen, Margaret.
Background for the Design of an Expert Consulting System for On-line Help.
October, 1984.
Thesis proposal, Temple University.

[18] *TOPS-10 On-Line Help System*
year unknown.
HLP:HELP.HLP on CMU-CS-A.ARPA.

[19] *CMU LISP On-line help*
year unknown.
on CMU-CS-A.ARPA.

[20] *Personal Computing on the Vic-20: A Friendly Computer Guide*
Commodore Electronics, Ltd., 1982.

[21] Coutaz, Joelle.
A Framework for the SPICE Help System.
1985.

[22] *TOPS-20 User's Guide*
Seventh edition, Digital Equipment Corporation, Marlboro, Massachusetts, 1980.

[23] Doherty, Walter J.
System Performance and user Behaviour.
IBM Thomas J. Watson Research Center. Yorktown Heights, NY, 1981.

[24] Doherty, Walter J. and R. P. Kelisky.
Managing VM/CMS Systems for User Effectiveness.
IBM Systems Journal 18(1), 1979.

[25] Doherty, Walter J., and Arvind J. Thandhani.
The Economic Value of Rapid Response Time.
IBM Thomas J. Watson Research Center. Yorktown Heights, NY, year unknown.

[26] Donner, Marc D. and David Notkin.
Flexible Systems: Customization and Extension.
1985.

[27] Draper, Stephen W.
The Nature of Expertise in UNIX.
HMI Project, University of California at San Diego, March, 1984.

[28] Duffy, Thomas M., and Paula Kabance.
Testing a Readable Writing Approach to Text Revision.
Journal of Educational Psychology 74(5):733-748, 1982.

[29] Dunsmore, H. E.
Designing an Interactive Facility for Non-Programmers.
In *Proceedings of ACM-80*, pages 475-483. 1980.

[30] Durham, Ivor.
The CMU Plot Manual
Carnegie-Mellon University Computer Science Department, 1981.

[31] Durham, Ivor, David A. Lamb, and James B. Saxe.
Spelling Correction in User Interfaces.
Communications of the ACM 26:764-773, 1983.

[32] Dzida, W., S. Herda, and W. D. Itzfeldt.
User-Perceived Quality of Interactive Systems.
IEEE Transactions on Software Engineering SE-4(4), 1978.

[33] Felker, Daniel B., Frances Pickering, Veda R. Charrow, V. Melissa Holland, and Janice C. Redish.
Guidelines for Document Designers.
American Institutes for Research, Washington, DC, 1981.

[34] Fenchel, Robert S.
Integral Help for Interactive Systems.
PhD thesis, UCLA, 1980.

[35] Fenchel, Robert S. and Gerald Estrin.
Self-Describing Systems Using Integral Help.
IEEE Transactions on Systems, Man, and Cybernetics SMC-12(2):162-167, March/April, 1982.

[36] Feyock, Stefan.
Transition Diagram-based CAI/HELP Systems.
International Journal of Man-Machine Studies 9:399-413, 1977.

[37] Finin, Timothy W.
Providing Help and Advice in Task-Oriented Systems.
In *IJCAI 83 Proceedings*, pages 176-178. 1983.

[38] Fischer, Gerhard, Andreas Lemke, and Thomas Schwab.
Knowledge-based Help Systems.
In *Proceedings of CHI '85*, pages 161-167. 1985.

[39] Foster, Mike.
private communication.
1981.

[40] Genesereth, Michael.
An Automated Consultant for MACSYMA An Automated Consultant for MACSYMA.
In *IJCAI 77 Proceedings*, pages 789. 1977.

[41] Girill, T. R. and Clement H. Luk.
DOCUMENT: An Interactive, Online Solution to Four Documentation Problems.
Communications of the ACM 26(5):328-337, May, 1983.

[42] Glasner, Ingrid D., and Philip J. Hayes.
Automatic Construction of Explanation Networks for a Cooperative User Interface.
Technical Report CMU-CS-81-146, Carnegie-Mellon University Department of Computer Science, November, 1981.

[43] Glushko, R. J., and M. H. Bianchi.
On-line Documentation: Mechanizing Development, Delivery, and Use.
The Bell System Technical Journal 61(6):1313-1323, July-August, 1982.

[44] Gosling, James.
UNIX Emacs Manual
1983.

[45] Gould, J. and N. Grischkowsky.
Doing the Same Work with Hardcopy and with CRT Terminals.
Human Factors 26(3), 1984.

[46] Haas, Christina, and John R. Hayes.
Reading on the Computer: A Comparison of Standard and Advanced Computer Display and Hard Copy.
Technical Report CDC Tech Report #7, Carnegie-Mellon University Communications Design Center, February, 1985.

[47] Haas, Christina, and John R. Hayes.
Effects of Text Display Variables in Reading Tasks: Computer Screens vs. Hard Copy.
Technical Report CDC Tech Report #3, Carnegie-Mellon University Communications Design Center, March, 1985.

[48] Halasz, F., and T. Moran.
Analogy Considered Harmful.
In *Proceedings, Human Factors In Computer Systems.* March, 1982.

[49] Hanson, Stephen Jose, Robert E. Kraut, and James M. Farber.
Interface Design and Multivariate Analysis of UNIX Command Use.
ACM Transactions on Office Information Systems 2(1):42-57, March, 1984.

[50] Hayes, Philip J.
Uniform Help Facilities for a Cooperative User Interface.
In *National Computer Conference Proceedings*, pages 469-474. AFIPS, 1982.

[51] Hayes, Phillip J.
Executable Interface Definitions Using Form-Based Interface Abstractions.
Technical Report CMU-CS-84-110, Carnegie-Mellon University Computer Science Department, March, 1984.

[52] Hayes, Phil, Rick Lerner, and Pedro Szekely.
Cousin Manual for End Users.
1983.

[53] Hayes, Phillip J., Pedro A. Szekely, and Richard A. Lerner.
Design ALternatives for User Interface Management Systems Based on Experience with COUSIN.
In *CHI '85 Proceedings.* April, 1985.

[54] Heckel, Paul.
The Elements of Friendly Software Design.
Warner Books, 1984.

[55] Houghton, Raymond C., Jr.
Online Help Systems: A Conspectus.
CACM 27(2):126-133, February, 1984.

[56] Howe, Adele.
HOW? A Customizable, Associative Network Based Help Facility.
1983.
Senior Design Project.

[57] Huck, Schuyler W. and Howard M. Sandler.
Rival Hypotheses: Alternative Interpretations of Data Based Conclusions.
Harper & Row, New York, 1979.

[58] *IBM Virtual Machine/System Product: CMS Primer*
First edition, IBM, 1982.

[59] *System Productivity Facility Dialog Management Services*
IBM, year unknown.

[60] Jacob, Robert J. K.
Using Formal Specifications in the Design of a Human-Computer Interface.
CACM 26(3), April, 1983.

[61] Kelley, J. F.
An Interface Design Methodology for User-friendly Natural Language Office Information Applications.
ACM Transactions on Office Information Systems 2(1), March, 1984.

[62] Kennedy, T. C. S.
The Design of Interactive Procedures for Man-Machine Communication.
International Journal of Man-Machine Studies 6(3):309-334, 1974.

[63] Kennedy, T. C. S.
Some Behavioral Factors Affecting the Training of Naive Users of an Interactive Computer System.
International Journal of Man-Machine Studies 7(6):817-834, 1975.

[64] Kernighan, Brian W. and John R. Mashey.
The UNIX Programming Environment.
Computer :12-22, April, 1981.

[65] Kernighan, Brian W. and P. J. Plauger.
Software Tools.
Addison-Wesley Publishing Co., Reading, Massachusetts, 1976.

[66] Kunze, John.
The Berkeley UNIX Help System.
1984.
on-line manual entry.

[67] Lamb, David Alex.
RdMail Message Management System: User's Guide and Reference
Seventh edition, CMU Computer Science Department, Pittsburgh, 1982.

[68] Lampson, Butler.
Hints for Computer System Design.
IEEE Software , January, 1984.

[69] Lang, Kathy, Robin Auld, and Terry Lang.
The Goals and Methods of Computer Users.
International Journal of Man-Machine Studies 17(4):375-399, 1982.

[70] Loo, Robert.
Individual Differences and the Perception of Traffic Signs.
Human Factors 20(1):65-74, 1978.

[71] Lowerre, B. T.
The HARPY Speech Recognition System.
Technical Report, Carnegie-Mellon University Computer Science Department, April, 1976.

[72] Mack, Robert L., Clayton H. Lewis, and John M. Carroll.
Learning to Use Word Processors: Problems and Prospects.
ACM Transactions on Office Information Systems 1(3):254-271, July, 1983.

[73] Magers, Celeste S.
An Experimental Evaluation of On-line Help for Non-Programmers.
In *CHI '83 Proceedings*, pages 277-281. 1983.

[74] Mantei, Marilyn and Nancy Haskell.
Autobiography of a First-Time Discretionary Microcomputer User.
In *CHI '83 Proceedings*, pages 286-290. 1983.

[75] Moran, Thomas P.
The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems.
International Journal of Man-Machine Studies 15(1):3-50, 1981.

[76] Moran, Thomas P.
An Applied Psychology of the User.
ACM Computing Surveys 13(1), March, 1981.

[77] Mudge, J. C..
Human Factors in the Design of a Computer-Aided Instruction System.
PhD thesis, University of North Carolina at Chapel Hill, June, 1973.

[78] Nicholson, Raymond S.
Why Interactive Computing Systems are Sometimes not Used by People who Might Benefit from Them.
International Journal of Man-Machine Studies 15:469-483, 1981.

[79] Norcio, A.
Indentation, Documentation, and Programmer Comprehension.
In *Proceedings, Human Factors In Computer Systems.* March, 1982.

[80] Norman, D.
The Truth About UNIX: The User Interface is Horrid!
year unknown.

[81] O'Malley, C., P. Smolensky, L. Bannon, E. Conway, J. Graham, J. Sokolov, and M. L. Monty.
A Proposal for User Centered System Documentation.
In *CHI '83 Proceedings*, pages 282-285. 1983.

[82] Palay, Andrew J., and Mark S. Fox.
Browsing Through Databases.
Information Retrieval Research.
Butterworth and Co., Ltd., London, 1981.

[83] Peters, Tom.
private communication.
1984.

[84] Posner, John, Jeff Hill, Steven G. Miller, Ezra Gottheil, and Mary Lynn Davis.
Lotus 123 User's Manual
Lotus Development Corporation, 1983.

[85] Price, Lynne A.
Thumb: An Interactive Tool for Accessing and Maintaining Text.
IEEE Transactions on Systems, Man, and Cybernetics SMC-12(2):155-161, March/April, 1982.

[86] Rashid, R. F.
An Inter-process Communication Facility for UNIX.
Technical Report CMU-CS-80-124, Carnegie-Mellon University Computer Science Department, February, 1980.

[87] Reddy, D. R. and the Computer Science Department Speech Group.
Working Papers in Speech Recognition IV -- The Hearsay-II System.
Technical Report, Carnegie-Mellon University Computer Science Department, February, 1976.

[88] Reisner, Phyllis.
Human Factors Studies of Database Query Languages: A Survey and Assessment.
ACM Computing Surveys 13(1), March, 1981.

[89] Relles, Nathan and Lynne A. Price.
A User Interface for Online Assistance.
In *Procedings of the Fifth Conference on Software Engineering*, pages 400-408. 1981.

[90] Relles, Nathan, and Norman K. Sondheimer.
A Unified Apporach to Online Assistance.
In *National Computer Conference Proceedings*, pages 383-387. AFIPS, 1981.

[91] Rich, Elaine.
Programs as Data for their Help Systems.
In *National Computer Conference Proceedings*, pages 481-485. AFIPS, 1982.

[92] Rich, Elaine.
Users Are Individuals: Individualizing User Models.
International Journal of Man-Machine Studies 18(3):199-214, 1983.

[93] Roberts, Teresa L.
Evaluation of Computer Text Editors.
PhD thesis, Stanford University, 1979.

[94] Roberts, T. and T. Moran.
Evaluation of Text Editors.
In *Proceedings, Human Factors In Computer Systems.* March, 1982.

[95] Roberts, Teresa L. and Thomas P. Moran.
The Evaluation of Text Editors: Methodology and Empirical Results.
CACM , April, 1983.

[96] Robertson, C. Kamila.
Experimental Evaluation of an Interactive Information Processing Aid for an Emergency Poison Center.
Behavioral Science 26, 1981.

[97] Robertson, C. Kamila, and Robert Akscyn.
Experimental Evaluation of Tools for Teaching the ZOG Frame Editor.
Technical Report CMU-CS-82-122, Carnegie-Mellon University Department of Computer Science, May, 1982.

[98] Robertson, C. Kamila, and Allen Newell.
Experimental Evaluation of Five Techniques for Teaching for the ZOG Frame Editor.
1983.

[99] Robertson, C. Kamila, Donald L. McCracken, and Allen Newell.
Experimental Evaluation of the ZOG Frame Editor.
Technical Report CMU TR 81-112, Carnegie-Mellon University, 1981.

[100] Robertson, G., D. McCracken, and A. Newell.
The ZOG Approach to Man-Machine Communication.
Technical Report CMU-CS-79-148, Carnegie-Mellon University, October, 1979.

[101] Robertson, G., D. McCracken, and A. Newell.
The ZOG Approach to Man-Machine Communication.
International Journal of Man-Machine Studies 14(4):461-488, 1981.

[102] Rosenberg, J.
Evaluating the Suggestiveness of Command Names.
In *Proceedings, Human Factors In Computer Systems.* March, 1982.

[103] Rothenberg, Jeff.
An Intelligent Tutor: On-line Documentation and Help for a Military Message Service.
Technical Report ISI/RR-74-26, USC-ISI, May, 1975.

[104] Ryan, Thomas Arthur.
Minitab Student Handbook
1976.

[105] Scelza, Donald A.
The Shepherd File Management and Documentation System User Manual
CMU Computer Science Department, Pittsburgh, 1979.

[106] Shafer, Steve.
Ci UNIX manual entry
1983.

[107] Shneiderman, Ben.
Software Psychology.
Winthrop Publishers, Cambridge, Massachusetts, 1979.

[108] Shneiderman, Ben.
Human Factors Issues of Manuals, Online Help, and Tutorials.
Technical Report CS-TR-1446, Department of Computer Science, University of Maryland, September, 1984.

[109] Shrager, Jeff, and Tim Finin.
An Expert System that Volunteers Advice.
In *AAAI-82*, pages 339-340. 1982.

[110] Smith, Huston.
The Religions of Man.
Harper & Row, New York, 1958.

[111] Smith, David Canfield, Charles Irby, Ralph Kimball, and Eric Harslem.
The STAR User Interface: An Overview.
In *National Computer Conference Proceedings*, pages 515-528. AFIPS, 1982.

[112] Sobell, Mark.
A Practical Guide to the UNIX System.
The Benjamin/Cummings Publishing Company, Menlo Park, California, 1984.

[113] Sondheimer, Norman K. and Nathan Relles.
Human Factors and User Assistance in Interactive Computing Systems: An Introduction.
IEEE Transactions on Systems, Man, and Cybernetics SMC-12(2):102-107, March-April, 1982.

[114] Sproull, Lee S., Sara Kiesler, and David Zubrow.
Encountering an Alien Culture.
Journal of Social Issues, 1984.
in press.

[115] Stallman, Richard M.
EMACS Manual for TOPS-20 Users
MIT AI Laboratory, 1981.

[116] Teitelman, Warren.
Interlisp Reference Manual
Xerox Palo Alto Research Center, year unknown.

[117] Temin, Aaron Lehman.
The Question Answering Module in an Automated Natural Language Help System for the Text-Formatter Scribe.
1982.
Thesis Proposal, University of Texas at Austin.

[118] *UNIX Programmer's Manual*
Seventh Virtual Vax-11 edition, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1981.

[119] Walker, Janet.
Symbolics Sage: A Documentation Support System.
In *Proceedings IEEE Spring CompCom 84.* 1984.

[120] Walker, Janet.
Implementing Documentation and Help Online.
1985.
CHI '85 Tutorial Notes.

[121] Weinberg, Gerald M.
The Psychology of Computer Programming.
Van Nostrand Reinhold Co., New York, 1971.

[122] Wilensky, Robert.
Talking to UNIX in English: An Overview of an On-line UNIX Consultant.
AI Magazine 5(1):29-39, Spring, 1984.

[123] Wilson, E. Bright, Jr.
An Introduction to Scientific Research.
McGraw-Hill, New York, 1952.

[124] Witten, Ian H. and Bob Bramwell.
A System for Interactive Viewing of Structured Documents.
CACM 28(3), March, 1985.

[125] Wood, W. G., and D. G. Martin.
Experimental Method.
The Athlone Press, London, 1974.