SEARCHING IN THE UNIX DIRECTORY SPACE

by

O. Akin, C. Baykan, & D.R. Rao

December, 1983

DRC-48-03-83

Searching in the UNIX Directory Space

Omer Akin, Can Baykan, D. Radha Rao

Abstract

The structure of the directory space and subjects search behaviors in the UNIX operating system environment are examined. Protocol analysis with two subjects and survey of the contents of the directory space of all users of E-VAX and X-VAX systems in the Architecture and Computer Science systems at Carnegie-Mellon University were conducted. Depth first search characterized both the organization of the directories and the behavior of the subjects. Single-step as opposed to multiple-step traversal of the directory tree was also prevalent in the subject's behaviors. Recommendations for system friendliness, in terms of reusability, orientation, robustness, and consistency are developed.

Table of Contents

List of Figures

List of Tables

Searching in the UNIX Directory Space

1. Intent

One of the basic functionalities of most computer systems systems is filing. By filing we mean storing data created or obtained by the user in such a way that it can be conveniently retrieved when needed. This function in fact is necessary in most work environments, computerized or not. In an office setting clerical staff performs filing. Different filling systems are established in response to the quantity and the contents of the materials to be filed. These may range from a linear (alphabetical or chronological) ordering to complex hierarchical categorization systems.

Filing systems that are embedded in operating systems of computers are similar at least in concept to their manual counter parts. Users store away files they create and the operating systems fetch names of these files or their contents at the input of a command. Due to concerns of size, as in the case of manual filing systems, some automated systems create networks of sub-files organized in a node-and-link format. The difficulty often observed in such systems is related to the complexity these node-link structures acquire through continued usage and growth. Structures created for ease of categorization and access become so largo that search is complex and clumsy. The user finds himself wandering in large networks trying to find a file saved weeks ago with imperfect recollection of its location, title or contents.

This paper is concerned with the investigation of the principles that insure "friendly" interaction between users and operating system in terms of their filing functionalities.[1] Our assumptions underlying this study are:

1. Management of file systems of a certain size involve non-trivial search problems: such as, finding a file in ones own direclory(s).

2. Most users search in these spaces using imperfect knowledge of the directory and imperfect knowledge of system commands available to them. Usually to find a file in an unfamiliar portion of a directory space requires search methods almost as cumbersome as exhaustive search and most users are unaware of commands that can simplify such problems.

3. Users employ intuitive search methods (such as depth first) which may not be optimal. Most operating environments don't assist users to overcome this deficiency.

[1] **The context of this investigation is the UNIX opemting system as it exists in the two systems, XVAX and EVAX, at Computer Science Dcpm tment, Cnrncgie-Mcllon University.**

Our purpose is to document users methods of search as a function of, controlled observations. More specifically, we would like to know: what their knowledge of the search space is? what their knowledge of operations that allow search is? what search strategies are prefered or used? what universal conventions assist search? We would also like to evaluate system capabilities that assist especially in the UNIX system on Ihe X-VAX and E-VAX machines. More specifically: what operators are available and what kinds of search methods are supported? are optional search methods encouraged? Finally we are interested in developing principles that can make systems more friendly to users managing large file systems.

2. File Directory Environment in UNIX

In Unix, directories are organized as trees. Each directory belongs to another directory which is *above* it, and can contain other directories that are *below* it. The directories at the topmost level are called by the names of the disks that are used for storage. We shall call these *root directories.* At the level below this are the directories of the individual users and project groups. These we shall call *homo directories.* Every user who has a valid account on a Vax/Unix system has a *directory area,* an area where the files that he creates are stored. When the user logs in, he is automatically placed in his hoina directory. It is possible lo create new directories thai become a part of the user's directory area that branch out of the home directory. We shall call these *sub-directories.*

For the purposes of examining users' behavior in the directory area we must introduce another term to designate temporal order of search in the directory area. The *current* directory of a user is the directory that the user is in at any given moment. One's current directory can be changed by using the *cd* command, in UNIX. This command takes as argument a directory name or a *pathname,* which specifies how to get from the current directory to another directory by listing the directory names on the path.

Files are the bcisic units for storing information in a file directory area. Files can be written and stored in different formats, such as text files, binary data files, program files, run files, press files. Each file belongs to at least one directory, but can also belong to more than one directory. A file can be accessed directly from all of the directories that it belongs to. In order to access a file in another directory, it is necessary to change the current directory to the one that contains the file. This can be accomplished by specifying the path from the current directory to the directory containing the file in addition to the name of the file. Specifying the path to any directory from the root directory level is also acceptable, regardless of what the current directory is. The memory aids and keys for accessing contents of a file are: the name of a file, the extension of the filename which may specify the type of file,[2] the location of the file in a specific directory, and to a lesser extent, the creation date of the file and other information about the file, such as its size.

When an account for a new user is set up on a Unix system, the system supplies the user with the home directory and five sub-directories directly under it. The home directory is identified with the users name and the sub-directories are called bin, doc, include, lib, man. We call this level of sub-directories, level-1. Any other levels created below this are labelled numerically in ascending order, as 2, 3, 4 and so on. A user's directory when it is initially assigned has only one level. Subsequenlly the user can delete these or create new ones connected to the existing directories as he sees fit.

This nested organization of files and directories is intended to make it easier to find a file. Rather than having to look at all the files that are in his/her area, (some users have ~1000 files) if the user knows which sub-directory the file is in, he/she has to search only the particular sub-directory. On the other hand, changing the current directory and searching other directories requires additional effort and information. Thus a tradeoff is involved between the organization and search of files in directories.

3. Experimental Design

3.1. Transcription System

Since we are interested in studying user's search and file management behavior, a detailed and accurate documentation of the interaction between users and systems is needed. *One* way of collecting this data is to use video recordings, a method we used in our study of electronic mail systems (1981). Considering the large amounts of time involved in reviewing video taped data, we developed an alternative method using automatic transcription of user interactions. This transcription system, UISCHAT, is a is modified version of CHAT.[3] UISCHAT enabled us to: transcribe user-machine interaction automatically, put time-stamps onto the transcriptions at critical points: when a system prompt appears, at the begining and at the end of user input. This software transcribes user/system interactions accurately and with speed. To streamline this process all output from the system for any given command is truncated to 10 lines in the transcriptions.

[2] There are conventions for using specific extensions for types of files, i.e. *.mss* for manuscript files, *.press* for press files.

[v] Chat is a program that can establish an etlujrnet telenet connection with a given host. This is a standard means of allowing the user to login to the designated host, in UNIX. For moie details refer to UNIX User's Manual.

3.2. Subjects

We used two subjects in our experiments. Both our subjects are graduate students in the Department of Architecture. They are regular users of UNIX, and are also familiar with other computer environments at Carnegie-Mellon University. By the term "regular" we mean, user's who function as programers and who use the system on **a** daily basis in their work: this includes **text** editing, programming and electronic mail.

3.3. Experimental Task **and Data**

Our experimental method is protocol analysis. The experimental task we used has two parts. In the first part, which we shall call the *Work session,* a subject is asked to do a general cleanup of his area on the *X-VAX* or the *E-VAX.* This task is limited to a total of 30 minutes. Immediately after this a second task followed, which we shall call the *Recall session.* In this task, subjects were asked to recall the names and locations of all of their directories and sub-directories. Since the purpose of the study was never mentioned to the subjects prior to the Recall sessions, we assume that the subjects were unaware of the real goals of the experiment and the results of the Recall session are not confounded. Ttie only dependency we expected between the two sessions is the positive effect on recollection of file names by subjects in the Recall session due to the review of their directory spaces in the Work session.

The data consists of the transcriptions of the Work session, generated by UlSCHAT and the information gathered in the Recall session. A sample from UlSCHAT's transcription is included in table(3-1). The complicated text refers to the point in the work session where Subject moves to home directory with a 'cd[1] command and lists all the contents with 'Is'. He inadvertently types a press file at the terminal. Next his effort to type the manuscript file at the terminal ends up with an error message due to a typo. When he finally succeedes in reviewing the manuscript file 'mont.mss' he removes the press file and lists the contents of the directory once again.

We also gained access to each subjects actual directory tree and tabulated statistical information in order to to analyze the directory structure they have created for their own use.

Table 3-1: UISCHAT's Transcription Sample

```
$[4] |$[5] :01 Nov 82 12:13:25|
|1st:01 Nov 82 12:13:29[6]|
cd[7]
|CR:01 Nov 82 12:13:32[8])|
cd
$ |$ :01 Nov 82 12:13:32|
|1st:01 NOV 82 12:13:34|
ls
|cr:01 Nov 82 13:13:35|
Messages    dbmail       g          lib          old.mail    remcmd/
ayd         dbop         glide      man          orsa        remember
bin         dead.letter  graph      mbox         oyna        res
build8.dat  demo.bld     home.mss   mont.mss     paper       sigmod
c           demo.dta     io.dat     mont.press   papers      zzz.p
comp        fe.inc       io.gli     navy         perq
databases   figures      l.shp      nsf          ref

$ |$ :01 Nov 82 12:13:37|
|1st:01 Nov 82 12:13:52|
cat mont.press
|CR:01 Nov 82 12:13:59|
(mont.press file is printed on the terminal)
|CR:01 Nov 82 12:14:19|
$ |$ :01 Nov 82 12:14:19|
|1st:01 Nov 82 12:14:21|
cat mon.mss
|CR:01 Nov 82 12:14:24|
cat: can't open mon.mss
$ |$ :01 Nov 82 12:14:25|
|1st:01 Nov 82 12:14:28|
cat mont.press
|CR:01 Nov 82 12:14:31|
(mont.mss file is printed on the terminal)
```

[4] Sytem prompt

[5] Time stamp after prompt

[6] Time stamp at the beginning of user input

[7] Commands are in italics

[8] Time stamp at the end of user input

Table 3-1: UISCHAT's Transcription Sample (cont.)

```
$ |$ :01 Nov 82 12:14:38|
|1st:O1 Nov 82 12:14:39|
rmmont.press
ICR:01 Nov 82 12:14:44|
$|$:01 Nov 82 12:14:45|
|1st:OI Nov 82 12:14:46|
ls
|CR:01 Nov 82 12:14:46|
Messages     dbmail       g            lib          orsa         remember
ayd          dbop         glide        man          oyna         res
bin          dead.letter  graph        mbox         paper        sigmod
builcl8.dat  demo.bid     home.mss     mont.mss     papers       2ZZ.jp
comp         fe.inc       io.glinsf    ref
databases    figures      l.shp        old.mail     remand
$|$:01 Nov 82 12:14:49|
|lst:O1 Nov 82 I2:14:51|
cutdead.tetter
```

4. Analysis

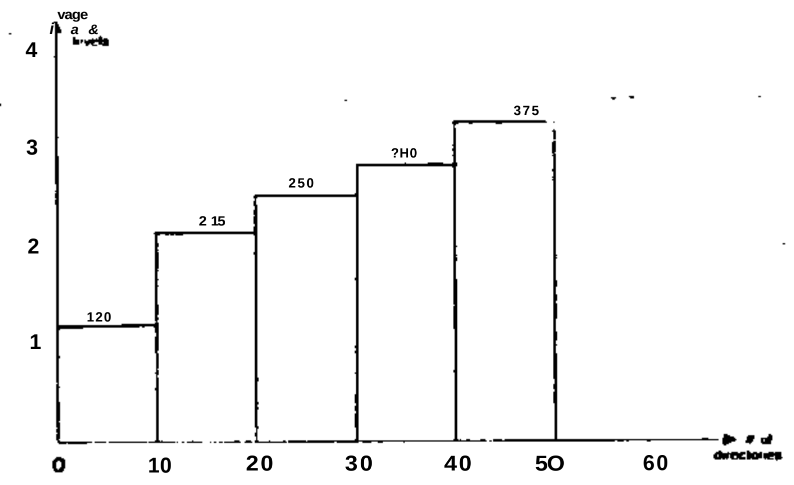
4.1. Directory Structure

We looked at the directories of two VAX machines: the X-VAX and E-VAX. Of the total 211 user accounts on the two machines, 171 belonged to individuals and 40 to projects shared by many individuals[9] A large variation in the number of files, sub-directories and directory levels exist in the X and E-VAXes.

Figures 4-1 and 4-2 show the relationship between the average number of levels and the number of directories. As the number of directories that are in a users area increase, so do the number of levels in his area. Also project directories contain slightly more levels than personal directories having the same number of subdirectories. This may be due to the fact that greater organization is needed to deal with the trafficing of a group of people using the same directory instead of a single user. Alternatively it may be due to the greater likelihood of having duplicate sub-directories and files in project directories.

Based on the way users organized their directories, we were able to distinguish two categories:

[9] All statistics related to users were collected during the fall 1983 semester.



Figu re 4-1: Average number of levels versus number of directories for project directories, those that contain files and those thai contain other directories. Some directories are for organizing other related directories under a common node, rather than for storing files. We call these *brunch -off* directories. Olhers contain mostly files and only a lew directories. We will call these *file directories.*

Moot user directories are shallow (figure 4 3). The median directory has only one level. Out of the 171 user level directories belonging to individuals, 86 contain one level. The maximum number of levels we encountered in a directory is 5.[10]

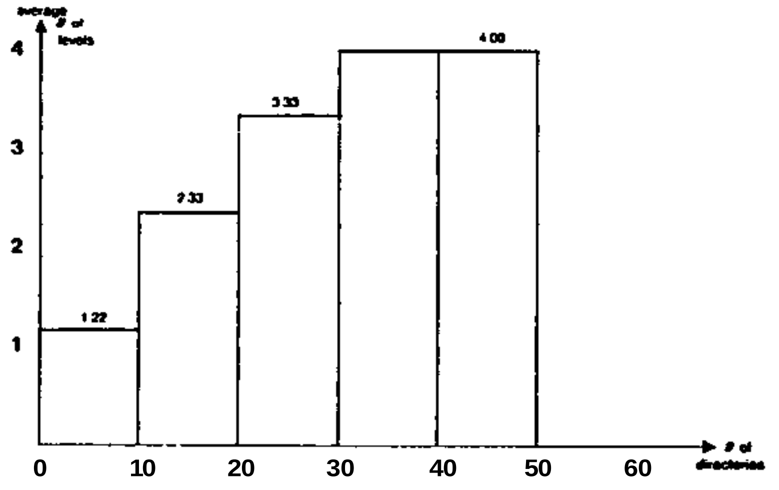
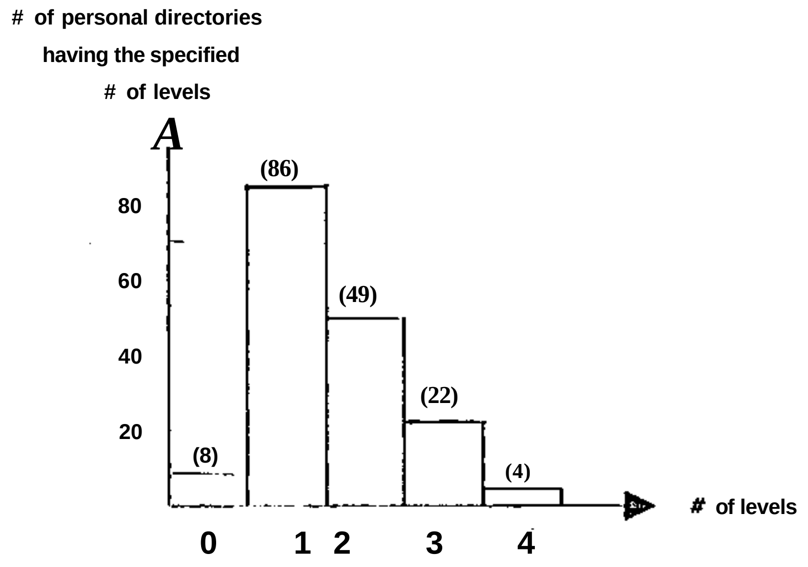


Figure 4-2: Average number of levels versus number of directories for personal directories

[10]There is only cwie cxnnifile of tlws cast;, which haf>pciuxJ to be n ixojcct directory.

We also looked at the overall directory structure of all users on the E-VAX and X-VAX to find out, if the general user habits in setting-up directories and sub-directories revealed any dominant trends. Table 4-1 shows that from a total of 211 accounts on X-VAX and E-VAX there are 108 sub-directories at level 1, 57 at level 2, 29 at level 3, 9 at level 4 and 1 at level 5. A total of 7 accounts exist without any sub-directories (1 project and 6 individual accounts) other than accounts containing only the standard directories assigned at the time the account was setup.[11]



Fiyure 4-3: Number of personal directories vs. levels

4.2. Work Session

Figures 4-4 shows Subject-1's directory tree. This area has 2 levels, contains 24 directories and approximately 350 files. This is almost identical to the average size user's area. The average number of levels for an individual directory containing 24 sub-directories in the X-VAX and E-VAX systems is 2.5 (figure 4-2).

it These accounts are not included in the tables 4-1 and 4-2.

Table 4-1: Directory Structure
X-VAX and E-VAX Directory Structure

	X-VAX		E-VAX		
Level	Project	Individual	Project	Individual	Total
1	9	48	11	40	108
2	4	24	4	25	57
3	3	13	3	10	29
4	2	3	2	2	9
5	0	0	1	0	1

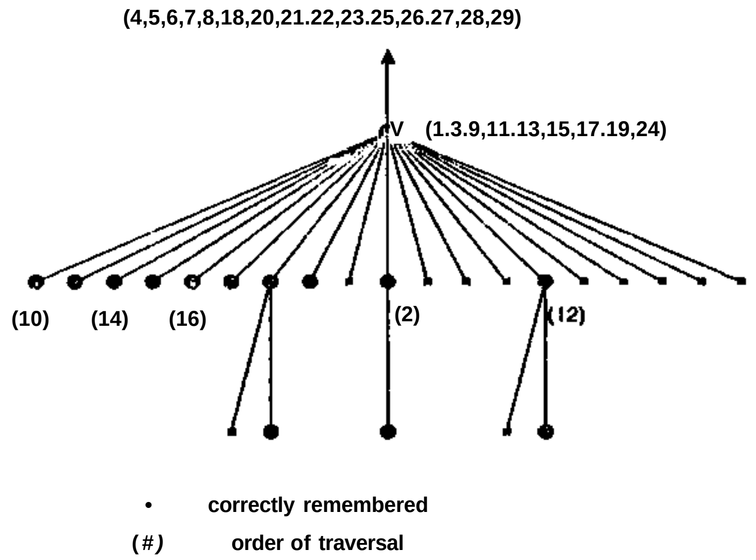
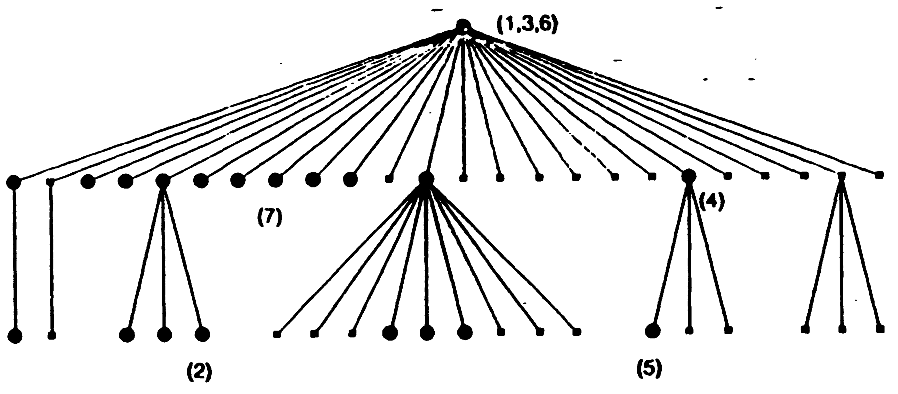


Figure 4-4: Directory of Subject-1's order of traversal and recall

During the work session the subjects were asked to clean their directories of files they did not need. Subject-1's order of traversal in his directory space is shown in the form of a graph in (figure 4-4). Each directory is indicated by a node in the graph. The number below each node specify the order in which the directories are visited by the subject. The directories that were correctly remembered by name and localion in the recall session are indicated by circular symbols as the nodes of the directory tree, and those that were not remembered by name or were simply misplaced are indicated by square nodes.

Subject-2 has 44 directories in two levels, which contain a total of -750 files. Subject-2's directory tree is given in figure 4-5. The subjects have a comparable number of directories at the first level.



● correctly remembered

(#) order of traversal

Figure 4-5: Directory of Subject-2's order of traversal and recall

However there is a significant difference between the number of directories at the second level: Subject-1 has 5 directories where as Subject-2 has 20.

Commands used by both subjects and their frequencies (table 4-3) indicate that subjects used the 'ls' command (to list names of all entries in a directory) more often than any other command. The 'cd <>' (change directory with an argument) command was the next most frequently used one.

4.3. Recall Session

Let us now review what happened in the Recall session. We shall primarily rely on observations about one of the subjects(Subject-1) to illustrate the general pattern of behavior. Subject-1 was able to recall 10 of the 20 directories that were at level 1, and 3 of the 5 directories at level 2 (table 4-2). On the whole he was able to recall 13 out of the total of 25, correctly. Of these 13, he visited 5 during the Work session. He was able to correctly remember the names and locations of all five directories he visited. Thus a recency effect in Recall behavior was present. Another plausible explanation for accuracy in recall is that the directories that Subject-1 has visited are likely to be only those of general interest to him. Hence, these may have been visited more frequently than others. It is interesting to note that although at the first level Subject-1 could remember only 5 of the 14 directories that he didn't visit, at the second level he was able to remember 3 of the 5 directories that he didn't visit. Also, he was able to recall correctly all of the directories at the first level that contain sub-directories or files at the second level.

Most of the errors Subject-1 made are typological errors or lapses of memory, such as trying to print a *.press* file at the CRT. There was only one instance in which the location of a sub-directory was remembered erroneously; and in this case Subject-1 tries to print at the terminal a file that is in another sub-directory.

Number of errors of both commission and omission are given in table 4-2. All numbers in parenthesis show the breakdown of observed frequencies for levels 1 and 2 of the directory, respectively.

Table 4-2: Actual Recall

	Correctly Recalled	Errors of Commission	Errors of Omission	Total
Subj: - 1	13(10,3)	1 (name)	11 (9,2)	24(19,5)
Subj: - 2	21 (13,8)	3 (location)	23(12, 11)	44(25, 19)

4.4. Comparison of Subjects

Both Subject's behaviors were by and large similar, yet there were some notable differences. Subject-2 recalled all directories fie visited during the Work session correctly in the Recall session. This can be attributed to a form of recency effect. However, the percentage of directories that were correctly recalled by Subject-1 is slightly higher than those correctly remembered by Subject-2. Correct recall is 54% for Subject-1 and 43% for Subject-2. On the other hand at the time of this study Subject-2 had almost twice as many directories in his area (44 vs. 24) and recalled many more directories correctly than Subject-I (21 vs. 13). By the same token Subject 2 visited proportionately a smaller part of his personal directory during the Work session than Subject-1.While Subject-2 visited 4 of his 44 directories Subject-1 visited 5 of his 24 directories.

On the other hand, there was no marked difference between the performance of the subjects in their use of the system commands available to them (table 4-3). Some general patterns that are observed in their behavior are: a) both subjects used the same kernel of commands, and b) the commands that were used most were *ls* and *cd.* This is probably due to the character of the task that the subjects were asked to do, i.e., to clean up their directories.

Both subjects used the commands in similar sequences. The command *ls* followed the command *cd* or the command *rm.* This we believe is due to the fact that UNIX commands are silent, and they execute without any feedback to the user. Thus the user needs to check the outcome of the execution of a command by another command. This other command case is /s, which lists the names of the files in the current directory. Most of the errors that the subjects made can be traced back to an erroneous assumption about the location of the current directory or the contents of that directory area. Not using the *ls* command frequently enough is the primary cause of these errors.

Both subjects used commands one at a time instead of specifying pathnames or executing multiple commands. We assume that this is the safeguard subjects have for avoiding errors and being assured of debugging those that are not avoided. In this way if they make an error, they know where on the path the error occured. Otherwise UNIX does *not* give error messages indicating that portion of the pathname which contains the error. Also, when directory change commands are used one at a time, they are interspersed with commands provide feedback to the users such as *ls.* Hence the subject can take a small step at a time by moving to a directory that is immediately above or below the current directory, and check the filenames and directory names at each step. Using these memory aids, (recognizing rather than remembering) subjects choose which directory to move to next with greater accuracy. Such a method of gradual focussing into one's target intuitively seems to be an appropriate search tactic since the recall behaviors of both subjects demonstrate significantly *incomplete knowledge* of their directory spaces (table 4-2).

Table 4-3: Commands and Frequency

	Commands	Frequency	Cmd. Sequence	Frequency
Subj: - 1				
	cd	7	cd ls	11
	ls	17		
	cd<>	17	mkdir	2
	cat	8		
	rm	6	rm ls	
	mkdir	2		
	mv	1	cd cd	
	cp	4		
Subj: - 2	cd	5	cd ls	6
	ls	9		
	cd<>	6	rm ls	1
	cd <>\<>	1		
	mkdir	0	cd cd	6
	mv	0		
	rm	5	lsrm	5
	cp	0		
	cat	5		

5. Discussion

At the onset of this paper we identified three objectives: a) documenting user's methods of search, b) documenting system functionalities in X-VAX and E-VAX that pertain to search behavior and c) identify principles of improving user friendliness. Let us now survey our observations in response to each objective.

5.1. Methods for search

The strategy of search that our subjects used is a straight forward one. There is a space of nodes that are linked strictly hierarchically with a unique "root" node, which initiates all access. Each node in this space has a name and the potential to contain multiple files. Access is strictly top down. Users have the ability to traverse a path upwards as well as downwards within the confines of this domain.

A search problem arises to the extent that the user has imperfect information about: a) what he is looking for and b) where he thinks he may find it. The first clear evidence we have in this area is the extent to which the subjects knew what they have in their own directories (figures 4-4,4-5). On the average they recalled only half of the names of their directory areas accurately 54% for Subject-1 and *47%* for Suhjoct-2. Most of the errors are omission and commission errors, accounting for 11% and 8%, of the total errors respectively. In other words both subjects failed to remember nearly half of their directory aieas; but almost all of the directories they could remember, were remembered correctly. Not only did they name thorn correctly but also placed them in the right location in the directory tree.

The rate of correct recall is no doubt positively influenced by the Woik task which preceded the Recnl! task. All directories visited in the Work task are recalled correctly and these constitute (23%) and (42%) of all correctly recalled directories for Subject-1 and Subject-2. Although we are not able to measure the magnitude of this effect in our analysis, it is clear that the 50% correct recall is at best an upwardly biased estimate.

On the other side of this argument is the relative inefficiency one must expect in recall, as compared to recognition of familiar items. That is, it is highly probable that while our subjects were rather inefficient in recalling directories they would have been much more efficient in recognizing the correct names and locations of the directories had they been presented with appropriate queues. In fact, during the Work session, the movement of both subjects in the directory space was accomplished by taking one step at a time and listing the contents of each new directory they were in before moving on to the next directory. Such a step by step pattern of movement suggest that they were searching for queues to ascertain that their path was indeed a correct one.

Alternatively some of this stop-and-go behavior may be in response to the directory clean up task that was given to the subjects at the onset; and many indicate the subject's need to survey their files at each directory node has the intention of *recognizing* the file that needs to be deleted, modified, renamed or relocated. In either case, the relative difficulty of recall is underscored.

The UNIX operating environment provides a variety of commands for searching directory areas. The standard command for moving in the directory space is cd, which stands for change directory. *Cd* can take a simple or string of directory name(s) as an argument. That single name must signify a directory adjacent to the current directory and indicate the directory to which the location of the user should be changed. The string of names specifies a directory path at the end node of which the user must be placed automatically. This version of cd is equivalent to issuing singular cd commands one after the other, as many times as there are directory names in the string. Thus it represents a significant short cut. Other more complex facilities for easy traversal of the directory tree also exist. User initiated commands that "jump" from the present directory to one at the end of a path which does not need to be explicitely spelled out by the user is one good example.

Throughout our observations both subjects failed to take advantage of the composite commands fnd thG *cd* commands specifying complex paths. Instead they consistently used singular *cd* commands to move around (table 4-3). The only explanation of this is their desire to survey the contents of each new directory they visit and to make sure that their direction of movement is correct. In fact, the majority of the cases where a *cd* command was used (47%) were followed by the *ls* command.

While the actual search behavior of our subjects provide the most direct evidence about how users deal with the problem of searching files in the UNIX environment, we also considered some indirect evidence. We assumed that the structure created by each subject to organize their directories implies their expectation vis a vis search. In other words, assuming that each subject has created an organization that assists him/her maximally in saving and finding files, this organization should tell us something about their search habits. In fact there is a dominant characteristic that is shared by both subjects directory trees: shallowness. The average number of files per level (not counting the root level) is 17. Furthermore, a great majority of directories (79% and 54% respectively) are connected at the level directly below the tree-root. This indicates a depth-first search strategy. Since the tree is very shallow, each excursion can traverse the tree in depth with little effort; making depth first search easy to achieve and attractive. In fact the movements of both subjects in their directory-tree during the Work session is characteristically "in depth" (figure 2-4, 2-5).

5.2. System Functionalities

The post-experimental interviews with both of our subjects (and others who commented on the issue, informally) indicate that UNIX provides a work environment favorable to the user in maintaining files. The features most favorably reviewed are: hierarchic structure of directories, flexibility of user commands and ability to structure special commands. In contrast to these we observed in our data a number of system functionalities which provide obstacles for friendly interactions with the user. These can be categorized under four headings: Conservation of time, Reusability, Orientation, Robustness and Consistency.

5.2.1. Conservation of time

Any filing system which is more complex than just a linear list of items has to be created with the consideration of a trade off between set-up time and search time. In the linear list, set up time is minimal. A new entry is inserted either at one end of the list or in between somewhere, based on a chronological or alphabetical ordering system. However, as the number of the entries increase so does the time it takes to locate an entry for retrieval. Furthermore, due to the limited information that can be conveyed through the file-name, when there is imperfect knowledge about the file being searched, the search may be expanded into the contents of each file or a subset (based on chronological sequence, for example) of the complete list of files.

More complex organizations, say hierarchical ones, simplify these problems. Files can be grouped according to multiple sets of criteria, or nested within larger groups of files as these criteria may dictate. This assists access to each file and reduces ambiguity about the location of a file being searched. On the other hand to maintain such an organization obviously takes more time and effort. Consistency in the way criteria are used to define sub-categories (or directories) is essential to accomplish unambiguous categories.

The UNIX directory system theoritically allows for either kind of organization. In principle this is a good idea and allows for a combination of the two. This is where the problem lies. While a hybrid system may potentially incorporate the advantages of both principles of organization, in practice we observed that they instead (or as well) assume the disadvantages of both.

First of all, we observe in table 4-1 that there is a high degree of unevenness in the distribution of the files in the various levels of the directory hierarchy. A large percentage of the files are concentrated at higher levels (level 1 and 2) especially in the case of individual's file directories (81% and 85%, respectively). In order to realize the kinds of economies in search time we mentioned above, the directory tree has to have its files distributed evenly. An efficient model for search the EPAM

(Feigenbaum 1961) network, for example, dedicates all upper level nodes in the search tree to criteria to be utilized in the search and deposites all contents to the terminal nodes. While in a tree structure which is constantly growing from a root node, accommodation of all files exclusively in the terminal nodes is an unrealistic goal to accomplish; the lesson to be learned from EPAM model apparent. Greater efficiency in search will be achieved in trees where the contents are more evenly distributed over all of their levels. Search will become progressively inefficient when the files are concentrated in only a few levels of the directory tree.

The subject's behaviors support this observalion. Searching for any file is tantamount to scanning long lists of files or directory names contained in the upper levels, one by one. The number of files actually accessed as a ratio of the total number of files and directory names scanned, during the whole experiment, are 7% and 3%, respectively, for Subject-1 and Subject-2. It is clear that the tradeoff between the set-up and search time in user directories is the factor which determines the actual form of each users directory. Without using system functionalities that improve each aspect of the tradeoff independently of the other, substantial improvement can not be achieved. We shall discuss this further in the next section.

Another aspect of user's efficient use of his time is the time he takes to accomplish a standard opeiation. In ihexase of search the basic operation is going from the present node in the directory tree to a desired node. As described earlier the operating environment of UNIX permits this to take various forms: by traversing a path (consisting of many nodes) by using a single command or by using many commands. We discovered that both of our subjects prefered to use many commands (i.e., take many steps going from A to B) rather than single commands specifying complete paths, in spite of the fact that the former took much longer to execute in UNIX (12.06 sec.) than the latter (0.857 sec.) on the average. The only plausible explanation of this seems to be the cost associated with making errors in specifying complex path commands. This introduces the next principle of user friendliness we would like to discuss: reusability.

5.2.2. Reusability

The problem can best be illustrated by a hypothetical example. The subject is at node A and wishes to move to node B. He thinks that the path connecting A and B goes through X, Y, Z. He directs the system to move on the path A-> X-> Y-> Z-> B. If the assumption about the path is correct, UNIX will execute this command and he will accomplish this task. If on the other hand his knowledge about the path is inaccurate (which was the case about half of the time in the data we examined) then he is back to square one. UNIX simply tells the user that the path does not exist and gives him neither a clue nor an opportunity to correct the error without having to retype the whole command over again.[12] Even if this is what the user may chose to do, he runs the risk of making the same error and facing the "all or nothing" choice once more. This is why we believe that both of our subjects have chosen to take one step at a time while traversing the tree and to scan all possible paths at each node. This in fact is the surest way to circumvent the problem created by the "all or nothing" proposition.

5.2.3. Orientation

One of the critical information in conducting search in any nontrivial search space is to know where one is in relation to important points of reference. For example, in hierarchical spaces it is important to know where the root of the tree is and where the terminal nodes of the tree are in relation to ones present position. In all search spaces it is critical to know where the goal node is in relation to the present position.

In UNIX there is a facility to provide information about the relative location of the root node. The command *pwd* gives the complete path between the *homo* directory and the current directory. Yet, UNIX lacks a general "map" facility which shows the current directory in relation to all outer boundaries of the directory space.

5.2.4. Robustness and Consistency

A common difficulty for both of our subjects was to have the system distinguish files from directories when names were included as arguments with system commands. For example, a command which contains in its argument list a filename causes an error in execution. The system is unable to appropriately interpret the user's intention to access the file named in the list. Most of the errors committed during the work sessions resulted from the subjects' confusing file names with directory names and the system's inability to assist in recovering from these errors quickly and effortlessly.

5.3. Improving Friendliness of the UNIX directory space

Above we outlined some observations regarding the user's behavior in the UNIX environment and the functionalities which are problematic in achieving a level of friendliness. Here we shall summarize a number of concrete recommendations which we believe will improve the operating environment of UNIX, especially in terms of file/directory maintenance.

1. It is clear that users neither structure nor search their directory space optionally. Greater efficiencies in this area can be obtained if the system environment provides standard tools for automatic filing and retrieval. Ideally the user should only be bothered with

[12] A recently developed version of the operating environment in **UNIX, COUSIN remedies this problem among others that** relates to system friendliness.

remembering something or just any thing (name, time of creation, subject area, source obtained from, contents, size) about a file and the system should be able to have the user access it automatically. If such a functionality was available the organization of the directory could also be undertaken by the system, automatically.

2. Efficiency in storage and retrieval tasks can be significantly improved without totally automating these functionalities. This requires that special commands be developed in both categories. A system of file organization can be developed for each user's area as a function of user profiles or based on a list of user defined criteria generated specially for each file or directory. In such a system, as users create new files, they would be prompted by the system for criteria and other information under which the file should be saved. When retrieving the file the user simply provides some or all of the criteria and the system conducts an EPAM-like search and comes up with a unique file or a list of files and directories. If multiple files are returned the user reiterates the process or searches manually from that point on.

3. For greater efficiency users have to be encouraged to use complex commands or lists of arguments in traversing their search space. 1 his requires that the system exhibit a degree of robustness in interpreting the users intentions correctly and allow the reusability of parts of erroneously typed commands or arguments. More specifically the system should provide a corrected command for approval by the user or at least diagnose the error and allow the user to modify earlier commands without retyping them completely.

4. More complete aids to *user,* showing his/her position in the search space must be available.

5. The system must make the distinctions between file and directory names automatically. Users should be consulted only when ambiguous information can not be disambiguated by the system.

6. A complete and easily accessible help system is necessary to insure an adequate level of communication. This, while has not been a major focus of our investigation, is a general prerequisite for friendly interaction.

References

Akin O., D. R. Rao (1901) Working papers in User-Computer Interface. Technical Publications, Dept. of Computer Science, Carnegie-Mellon University. CMU-CS-81-140.

Card, S. K., Moran, T. P. & Newell, A. (1930) The keystroke-level model for user performance time with interactive systems. Communications of the Association of Computing Machinery. 23, 396-410.

Codd, E. F. (1974) Seven steps to rendezvous with the casual user. In Data Base Management. KLIMBIE, J. W. & Koffeman, K. L, Eds. Amsterdam: North-Holland.

Cuff, R. N. (1980). On casual users. International Journal of Man-Machine studies. 12,163-187.

Feigenbaum, A. M. (1961) The simulation of Verbal Learning Behavior, Proceedings of the Western Joint Computer Conference. 121-132.

Kernighan, B. W. (1978) UNIX For Beginers - Second edition. Bell Laboratories, Murray Hill, New Jersey 07974.

Rich E. (1983) Users are Individuals: individualizing user modes. International Journal of Man-Machine Studies. 18, 199 - 214, 1983.

Rich E. A. (1979a). Building and exploiting user models. Ph.D. thesis, Carnegie-Mellon University.

Rich E. A. (1979b). User modelling via steriotypes. Cognitive Science. 3, 329-354.

Thompson, K. L and D. M. Ritchie, (1978). The UNIX Programmer's Manual, Bell Laboratories.

File Management Techniques. (Nov 5 1980). EDN USA Vol. 25 No.20 pp.289 - 91.