STUDYING LONG TERM SYSTEM USE

Judy Kay and Richard Thomas

TECHNICAL REPORT 497
MARCH 1995

BASSER DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF SYDNEY

Basser Department of Computer Science
University of Sydney
Sydney N.S.W. 2006
Australia
Studying long term system use

Judy Kay
Basser Department of Computer Science
University of Sydney, Australia

Richard Thomas
Department of Computer Science
University of Western Australia, Australia

CR categories and subject descriptors:
D.2.2, H.1.2, H.5.2, I.7
text editors, user modelling, training, human-computer interaction

General Terms: field study, education

Additional Terms: exploratory learning, long term learning

to appear: ACM Communications of the ACM
INTRODUCTION

There have been many studies of computer users but few that extend over very long periods. There are various reasons for this such as costs, the quality of data that can be obtained and satisfactorily analysed, and the requirement to obtain results in the short term.

We were interested to build long term user models for teaching systems and wished to track the total development of a guru as well as other types of users.

Such goals require tracking large numbers of people over a very long period because guru level experience takes years to acquire, and it cannot be predicted in advance who will turn out to develop this level of skill. Therefore even to find one guru may require the tracking of around 20 to 100 users for a year or two. Ten gurus might require 1000 users.

It is only possible to observe such numbers over years using a method that is very careful about costs. The objective of this paper is to present a methodology for monitoring many users in their natural settings over the long term and give an example of its application to the domain of students learning an editor for three years. We show how data obtained from this methodology permits the construction of individual user models.

MONITORING METHODOLOGY

We developed a monitoring system to study the use of the sam (13) text editor. It is essentially a modeless editor with most basic commands invoked using the mouse. It also has powerful keyboard commands, issued through the command window, including the usual Unix regular expressions and an elegant interface to the Unix shell.

We had a number of design goals for our system which strongly influenced our methods. First, we were restricted to low system building costs followed by very low operational
costs. Second the system needed to be highly reliable; it would not be possible to make changes once the monitoring was underway, nor could the immediate availability of people to fix problems during operation be guaranteed. Third, the impact on the computing resources had to be negligible - a degradation in performance during the day was not an option. Fourth, we wanted to give the users some control over their own data, they should be able to read it and delete it if they really wanted to. Finally, and most demandingly, we wanted to measure the time of first use of each command, but this required continuous monitoring of all editor use.

We had to study a large population to stand some chance of tracking a few power users. We took the decision not to collect details of actual edits, only commands invoked. To store a copy of every file and all the characters entered during editing would have been prohibitively expensive, and it could have violated privacy. We believed that this low level, large quantity of data would be sufficiently rich to obtain data on trends and changes over the long term.

We modified the source of sam as simply as possible in order to make it output an identifier every time a command or operator was invoked. The generation of data could be inhibited by the use of a special switch, but the default was to output every time sam was used by anyone.

Table 1 shows a simple Pascal program before and after editing. There are two errors in the first line of the original program, which the user will correct. Table 2 shows the monitor file, and a description of what the user did. A sam session is started, reading in the original file "demo.p". After correcting the major error, the file is rewritten to disk and re-compiled (not shown in the monitoring data). The user then inserted the missing semicolon on the first line, added a couple of extra lines to the program, and put a semicolon after the second "writeln" statement. An attempt to quit sam results in a warning that the file has been changed since the last write, so the user reconfirms the quit.
This data gives an incomplete picture of what the user was doing. For example, there is no indication of the size of the window created in line 2 - it could have been an unsuitable size. There is no mention of where the cursor is placed, for example in line 4. We have no notion from lines 5 to 7 that the user had to correct the typing of the word "printing". Again, in lines 12 to 15 we do not appreciate that the user went back to type the semicolon after the "lazy dog" line. And in line 18, we cannot infer that there was no net action because we do not know that the backspace was typed after the character. On the other hand, we know quite a bit about patterns of invocation and frequencies and these are employed in our user modelling.

In general, there are a few issues about the quality of this data. First, we cannot ensure that all sam work by an individual has been monitored. Nor can it be certain that another user didn’t create some of the commands, perhaps while demonstrating how to perform a task. To check this, methods of "fingerprinting" a pattern of use would need to be developed.

Our experience has helped us formulate a set of design guidelines though. It will be noted from Table 2 that none of the data is generated from sources other than the immediate sam session. Thus it is fine to include filenames and timestamps, because these are known by the system. But nothing depends upon the context of a user’s work during any arbitrary period, eg "the third session today by this user". The entire context is within the file. This has been a major contributor to reliability and scale of operation. If the system crashes, we can recover immediately without doing anything special for sam.

Second, the relative frequencies of commands are quite well behaved (see Figure 2). So it would be possible to work out the volume of data that needs to be collected given the command set size and the minimum number of observations of each command.

Third, we have found that redundancy in the data helps when things appear to go wrong. For example, we add an independent timestamp to each monitor file and these have been
useful for cross checking data.

<table>
<thead>
<tr>
<th>Original Version</th>
<th>Final Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>program (input, output)</td>
<td>program printing (input, output);</td>
</tr>
<tr>
<td>begin</td>
<td>begin</td>
</tr>
<tr>
<td>writeln('The quick brown fox');</td>
<td>writeln('The quick brown fox');</td>
</tr>
<tr>
<td>writeln('jumped over the lazy dog.'</td>
<td>writeln('jumped over the lazy dog.'</td>
</tr>
<tr>
<td>end.</td>
<td>writeln;</td>
</tr>
<tr>
<td></td>
<td>writeln('Goodbye world.');</td>
</tr>
<tr>
<td></td>
<td>end.</td>
</tr>
</tbody>
</table>

Table 1: Original and final versions of "demo.p"
<table>
<thead>
<tr>
<th>Line #</th>
<th>Monitor Data Collected</th>
<th>What the user did</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>H rct 793271236</td>
<td>Invoked by user &quot;rct&quot; at time 793271236</td>
</tr>
<tr>
<td>2</td>
<td>MrMy</td>
<td>Select demo.p from menu and sweep out window</td>
</tr>
<tr>
<td>3</td>
<td>G107 demo.p 793271270</td>
<td>file &quot;demo.p&quot;, size 107 bytes, read at 793271270</td>
</tr>
<tr>
<td>4</td>
<td>Md</td>
<td>Place the cursor after the word program</td>
</tr>
<tr>
<td>5</td>
<td>18 0 15</td>
<td>Type &quot;pinting &quot;, taking 15 seconds until the next action</td>
</tr>
<tr>
<td>6</td>
<td>Md</td>
<td>Place the cursor after the &quot;p&quot; in &quot;pinting&quot;</td>
</tr>
<tr>
<td>7</td>
<td>11 0 25</td>
<td>Type the character &quot;r&quot; and spend time (25 seconds) thinking</td>
</tr>
<tr>
<td>8</td>
<td>Mv</td>
<td>Use the walk though menu to write out the file</td>
</tr>
<tr>
<td>9</td>
<td>X116 demo.p 793271327</td>
<td>demo.p rewritten to disk, now 116 bytes</td>
</tr>
<tr>
<td>10</td>
<td>Md</td>
<td>Position cursor at end of first line</td>
</tr>
<tr>
<td>11</td>
<td>11 0 10</td>
<td>Type &quot;;&quot;, taking 10 seconds</td>
</tr>
<tr>
<td>12</td>
<td>Md</td>
<td>Position cursor after the last writeln statement</td>
</tr>
<tr>
<td>13</td>
<td>138 1 36</td>
<td>Type in 2 writeln statements in 36 seconds using backspace once</td>
</tr>
<tr>
<td>14</td>
<td>Md</td>
<td>Place cursor at end of previous writeln statement</td>
</tr>
<tr>
<td>15</td>
<td>11 0 4</td>
<td>Type &quot;;&quot;, taking 4 seconds in all</td>
</tr>
<tr>
<td>16</td>
<td>Mv</td>
<td>Use the walk though menu to write out the file again</td>
</tr>
<tr>
<td>17</td>
<td>X155 demo.p 793271506</td>
<td>demo.p rewritten to disk, now 155 bytes</td>
</tr>
<tr>
<td>18</td>
<td>11 1 9</td>
<td>Type a character into &quot;demo.p&quot; and then delete it</td>
</tr>
<tr>
<td>19</td>
<td>T793271554</td>
<td>Timestamp showing 5 or more elapsed minutes</td>
</tr>
<tr>
<td>20</td>
<td>MaKqFnKq</td>
<td>Select the command window, quit after warning</td>
</tr>
<tr>
<td>21</td>
<td>S793271558</td>
<td>Sam shutdown with time stamp</td>
</tr>
</tbody>
</table>

Table 2: Monitor file and associated user actions

The data generated by every sam session is stored in separate hidden files in the user's home directory. Thus the interested user is free to look at the data and even to delete or modify it, although in practice they rarely do so. This open policy has also been extremely easy to implement, and avoids any complications that could arise from remote hosts being down or daemons stopping. Overnight, the contents of the hidden files are
transferred to a central location where there is one file per user per year. The hidden files are deleted while the data is being accumulated centrally. The nightly collection runs with superuser privileges, but otherwise no special permissions are required. Every account across the hosts is checked for data, so there is no need to store and update tables of sam users. This design made it easy to extend monitoring to anyone who obtains an account.

We believe the following to be important for practical system operation. First, it is in the nature of long term monitoring to expect system crashes, changes to machine configurations and the distribution of users among hosts, and upgrades to hardware and software. It has to be easy to deal with these because support staff are always under pressure at these times. We found that minor processing overnight and year end housekeeping solve most of the difficulties.

Second, we have a fail-soft design. If the nightly collection fails, the data just sits in the users' directories for another day. If the system crashes, only the current sam sessions are affected. Actual data generation takes place in real time, locally to the user.

Third, all the expensive processing is performed off peak. Merging and compressing files takes time, but the system is lightly loaded when we do it.

There have been few problems since monitoring started in 1991. We have found the system fairly easy to maintain, and because it is cheap and unobtrusive, there has been little pressure to turn it off. Disk space used is a few hundred Mbytes, most of which can be archived as necessary.

Generally, we have found it necessary to expect the unexpected. Almost all limits get breached eventually, even if it is only by individuals putting a book on the mouse and holding the cursor down! But on the positive side, the monitoring helped find at least two obscure bugs.
Overall guidelines include: keep all the original data, even if archived away; build team commitment amongst systems programmers, operations staff and researchers; try a dummy run on the data to see how well it can be analysed; and finally, plan for autonomous operation.

EXAMPLES OF TREND DATA

We used the monitoring method to study the use of the sam text editor. There have been many studies of users of text editors and their learning (1, 6, 9, 11, 14, 15, 10), which build up a picture of users learning a great deal early and then stopping learning. In his survey Bosser (2) states (page 119) that ‘performance approaches the asymptotic range after about 50 hours’. We wanted to study if this trend held as we had reasons to believe it did not. For example some editors such as vi are believed to take considerable time to learn well, and the data from Mason’s adaptive command prompting system (12) suggested to us that some users went on developing over a long period. We were just curious to know how long. Additionally we wanted to build long term user models.

The study involved all 2273 undergraduate computer science students enrolled at the University of Sydney for the three years 1991 to 1993. We give sample analyses for some 63 students who began university in February 1991 and studied at least until November 1993. These analyses should help the reader obtain an understanding of the scale and rarity of some user actions.

Students were not compelled to use sam, so our sample included both committed and discretionary users.

The 63 three-year users issued over 4 million commands, collectively experimenting with 77 types at an average of 42, the maximum for a single user being 55.
Many of the calculations were performed using "work periods". Each user's data was split into 20 buckets of equal size, representing 5% slices of experience. This approach avoids some of the difficulties with vacations and deadlines.

We use the notion of command-rank: the most-used command has rank 1, the least-used 77 (for this group). All commands are listed by rank in the Appendix.

Command Frequencies

Simple summary statistics can inform about the patterns of editor use. For example Figure 1 is easily generated by aggregating. It shows the relative use of each command by time period. The very popular commands, ranks 1 and 2, are "position cursor" and "insert" - together accounting for about 40% of all use.

There are less than 10 consistently heavily-used commands: these are the leftmost lines. They perform primitive operations as listed in the Appendix for ranks 1 to 10. It can be seen that frequencies of these highly popular commands do vary across the time periods, although their ranking from period to period is fairly constant.

Another set of commands, with ranks 11 to 20, appear as small humps. These are still quite simple commands, and tended to be phased in or out of use during the three years. Substantial data collection is needed to observe trends here.

The rest of the lines are flat. Such low use cannot be reflected at all in the scaling we have used for this figure.

These observations help to explain the widespread belief that all the important learning happens early. Such a conclusion is not surprising given that the highest ranked commands tend to be learnt early.
Informally, we find that command frequencies tend to follow an exponential distribution, in contrast to the Zipf law found in the Unix command domain by others (7, 8). Figure 2 depicts the relative command frequencies, on a log scale, plotted against the rank for the tenth time period. The 15th ranked command is used about 1% in all, whereas the 35th ranked occurs only once in 10,000 invocations. Only 54 commands were used during period 10, compared with 77 over the three years.

Changes in Command Use

Figure 3 shows how the number of users of selected commands changes quite significantly over time. These trends can help establish where to do more detailed analysis of the monitor data by hand. It also establishes norms of usage so we can better identify “interesting users” who deviate markedly from this pattern. Successive time periods are represented along the horizontal axis, while the percentage of users making any use of the command in each time period is shown vertically. The four curves need explaining individually. They were selected to illustrate various points about trends and monitoring.

The early rise in the users of the "line number" command is quite steep. Students were recommended to use it as soon as they began writing programs - errors were reported by line number - and the command is directly suitable for this task. This pattern shows how our actions, as teachers, may have affected overall trends.

The "load files" command is issued through the command window. Its strength is that it can load many files into the menu in one action. The curve reflects a shift to those keyboard commands which achieve a lot of work in one hit: many files could be loaded at once.

The bottom curve in the figure is for the "undo" command. It shows a steady but slow increase. We found this a very surprising result, having expected ‘undo’ to be in the
essential set of commands and having told students many times about it. Were we to adopt data like this for measuring some effects of teaching, we would need to check out carefully why students chose not to bother much with "undo".

A very different profile applies for some commands. One interesting case is the "reshape_sweep" command which allows the user to sweep out a window exactly where it is wanted and with appropriate dimensions. Its use declined steadily: initially, this may have been due to a drop in accidental selections of the menu options but in the long term, there was a shift to use of the default window position and size values. The average use of "reshape_sweep" per person, was fairly constant, suggesting people slowly developed a preferred style and stuck to it. Asymptotically, about half the students used only the defaults. Data like this enables us to establish models of the long term command use for different groups of users. These can serve as a basis for teaching.

There is no doubt learning took place over the period of the study because the students could not use sam initially. But we cannot tell whether the learning that took place was appropriate. Were the best commands deployed for various tasks? Was the low use of, say "undo", a result of the workload.

We suspect people were still learning at the end of the study. We have evidence that some users underwent dramatic changes in style towards the end of their degrees when the workload was probably at its peak (3).

INDIVIDUAL USER MODELS

An important part of the power of our approach is that we can construct useful models of individual users at modest cost. A user model or student model represents the system's model of aspects such as the user's knowledge. In our sam work, we have built tools that perform quite simple analyses of the monitor data to create models similar to the one displayed in Figure 4. In this section we discuss the ways that our monitoring methodology
supports low cost construction of such user models. We also indicate some of the ways we have used these models to facilitate learning.

Figure 4 shows the aspects the user appears to know as dark nodes. For example the top square is the go-command (go_k). This is invoked by clicking on mouse button-1 to go to a new editing point. Our display indicates the user appears to know this command.

Nodes in the display that are diamond shaped represent user misconceptions. For example, the one labelled quit_b represents the common misconception that it is better to exit the editor destructively, by killing its window, than using the proper quit command. In our displays of the model this is dark, indicating that the user does not have this misconception. This means that the model for an expert who knew every aspect of the editor would be composed entirely of dark nodes.

The other node shape in the figure is a cross. This is for user attributes other than knowledge. The node labelled pascal_c indicates this user programs in Pascal.

There are many ways that the monitor data can be used to build user models. Here, we discuss in detail those we actually used in a series of experiments involving a simple CAI system. We constructed these user models from the monitor data at two stages: at the end of the first year of editor use and again in the middle of the second year.

We had initially expected to need quite sophisticated analyses of the monitor logs to establish user’s knowledge. However, we decided, in the first instance, to try very simple approaches and evaluate them before proceeding to more complex analyses.

The simplest approach seemed to be to count uses of the each command over a period of time and then to use a threshold count to define whether the user is judged to know that command. We set these thresholds heuristically. First, we used summary data on overall usage of commands to classify commands according to their frequency. Then, we studied
selected user logs by hand to establish the threshold number of command uses needed for
the analysis program to classify a user as knowing it. We were able to define a small set
of different threshold counts that seemed to classify user’s knowledge consistently with
our own hand analysis of the monitor log. Once the counted uses for a command
exceeded the required threshold a piece of ‘evidence’ was added to that person’s user
model.

It was clear that this approach could not be applied for commands that are easily invoked
accidentally. This includes mouse commands. For these, we do more complex analyses
to exclude at least some of the most blatantly unlikely selections. For example, the xerox
command enables the user to create multiple windows on one file. To filter out unintended
selections, we analysed the monitor data around the selection to identify those
invocations of xerox that were unlikely to be useful. For example, we determined
whether the file was too big to fit in one window and whether there were any effective
editing actions after the xerox. We also decided that xerox should be regarded as a relatively
sophisticated command and ignored uses of such commands for the very early,
highly chaotic period.

At the end of the first year of monitoring, we conducted our first evaluation of the user
models constructed by this process. Details of the actual threshold values and the analysis
programs are reported in Cook and Kay (4). Here we provide a brief summary. We
selected seven students with a wide range of academic performance and interviewed
them.

The interview began by asking how the student how they judged themselves to know
something in computing. The answers of all seven emphasised the view that one only
really knew something if one could apply it. There was also a strong sentiment that really
knowing something required a range of uses in different situations.

We then asked the students to examine the display of their user model, to explore it and
talk aloud about its accuracy as they examined each part. At the same time, the user model viewer program logged their actions for later analysis.

The students all agreed that the model accurately reflected their knowledge of the editor. We judged this to be an indication that these analyses of our monitor data might be adequate to build individual user models.

After this encouraging early evaluation, we decided to run another experiment that both employed the user models and served as additional validation of the accuracy of the modelling process. At the mid-point of our study, eighteen months from the beginning, we again constructed user models, this time for a larger, randomly selected sample of students.

These students were mailed information about how to run the user model viewer program. Those who actually did so, were able to see a display similar to that of Figure 4, but with facilities for users to interact with their user model. The program offered users three menu options for interacting with each node in their user models.

The first is an explanation: this describes the meaning of this part of the user model and can be seen as a form of sam documentation. The explanations are customised to match the user's knowledge: so the novice sees explanations that use only terms that are likely to be familiar.

The second option is to be given a justification for the system's assessment of the user's knowledge of this component. Since the user model operates by keeping the collections of evidence about the user's knowledge of each aspect, the justification simply presents this. The last option is to change the value. The viewer and the design of the justifiable user modelling system are described in greater depth in Cook and Kay (5).

With the viewer interface, the user model itself becomes a learning tool and a novel
medium for providing documentation. Models like this are also an invaluable basis for conventional CAI systems. Indeed, we have used them to drive a coaching system. This used the model's definition of what the user knew, along with a series of desirable teaching goals, to generate advice to each user.

One of the goals of our work was to explore the usefulness of the low cost but plentiful monitor data for building models of what each individual knows. We were surprised at the effectiveness of simple analyses of the monitor log.

We attribute this in part to the nature of learning computer tools like an editor where it is important to automate basic editing actions so that they become 'ready-to-hand' (16). Only then can users focus on and remain absorbed by their real task, such as constructing a program. This is also consistent with our interviewees' judgement that they knew something only if they had repeatedly applied it in ways they judged satisfactory. This, combined with the very large body of data generated by our monitoring methodology seem to provide a simple but effective means for creating models of user's knowledge.

CONCLUSIONS

We have presented a methodology for tracking the use of a system in a longitudinal time frame by large numbers of people. An example has been given of its use in the domain of students using the sam editor.

We attribute some of the success of the methodology to the almost autonomous mode of operation, with new users being tracked by default. The low cost, fail-safe design also helped to ensure that there was little pressure to stop monitoring. We found that giving users some access to their own data actually simplified the implementation.

We have shown that it is hard to determine user intentions from our data design. Research is needed to develop powerful methods to infer intentionality from monitor data, as was
also noted by Bosser (2). There is scope to benchmark the output from our data against more traditional measures of learning and usability. In addition, the design of the monitoring languages has research potential. The development of methods to identify users from their data “fingerprints” can also help to determine the integrity of any data collected.

We tracked over 2000 users of the sam editor for up to three years. Overall command frequencies have been presented, and their implication for the size and duration of studies has been discussed. In essence, the very low frequencies of some types of command dictate large scale, low level data collection. We have also shown some of the types of changes in command use over years that can be observed using our methodology.

The trends illustrated in Figure 3 indicate that the profile of command use is continuing to change steadily for some commands. It seems likely that these changes correspond to users learning. Moreover, this seems to be important learning. The commands which are being adopted late, for example "load files" and "undo", tend to be very powerful. Our analyses illustrate the growing numbers of users making use of a command like undo and that, even at the end of three years, the trend has not reached an asymptote. Our methodology opens the possibility of developing a better understanding of such very long term learning in the workplace.

We would expect the method to be applicable to other studies of applications with window and mouse or simple command interfaces. In situations with networked users, in which data can be accessed routinely for periodic collection, it may be possible to adopt our ideas. The method is particularly applicable when there are many users, all within a defined system area.

Our methodology offers the means for both low cost tracking of use profiles for a system and a practical mechanism for constructing detailed individual user models. Together, these can improve our understanding of the development in use of a tool by large user
populations, over a long period in a natural setting.

Acknowledgements

This study was supported by Telecom Grant Y05/04/34. We are grateful to Kathryn Crawford for invaluable discussions about learning tasks, to David Benyon in helping set up the study, Ronny Cook who built most of the programming tools and aided in the analyses and Greg Ryan for his support throughout. Last, our gratitude to the guest editors of this issue for helpful comments and extraordinary patience and perseverance.

References


### Appendix

<table>
<thead>
<tr>
<th>Rank</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>position</td>
</tr>
<tr>
<td>2</td>
<td>Insertion of Text</td>
</tr>
<tr>
<td>3</td>
<td>scroll</td>
</tr>
<tr>
<td>4</td>
<td>scroll</td>
</tr>
<tr>
<td>5</td>
<td>drag</td>
</tr>
<tr>
<td>6</td>
<td>scroll</td>
</tr>
<tr>
<td>7</td>
<td>write (walk thru menu)</td>
</tr>
<tr>
<td>8</td>
<td>cut</td>
</tr>
<tr>
<td>9</td>
<td>select file from menu</td>
</tr>
<tr>
<td>10</td>
<td>select window</td>
</tr>
<tr>
<td>11</td>
<td>paste</td>
</tr>
<tr>
<td>12</td>
<td>double click</td>
</tr>
<tr>
<td>13</td>
<td>reshape default size</td>
</tr>
<tr>
<td>14</td>
<td>snarf</td>
</tr>
<tr>
<td>15</td>
<td>write</td>
</tr>
<tr>
<td>16</td>
<td>Quit</td>
</tr>
<tr>
<td>17</td>
<td>Set dot to given address</td>
</tr>
<tr>
<td>18</td>
<td>line number</td>
</tr>
<tr>
<td>19</td>
<td>look</td>
</tr>
<tr>
<td>20</td>
<td>Esc, newline or tag</td>
</tr>
<tr>
<td>21</td>
<td>Write buffer to given file</td>
</tr>
<tr>
<td>22</td>
<td>Set current file, loading file(s)</td>
</tr>
<tr>
<td>23</td>
<td>reshape sweep size</td>
</tr>
<tr>
<td>24</td>
<td>search</td>
</tr>
<tr>
<td>25</td>
<td>new</td>
</tr>
<tr>
<td>26</td>
<td>exchange</td>
</tr>
<tr>
<td>27</td>
<td>&quot;sam&quot;</td>
</tr>
<tr>
<td>28</td>
<td>close (walk thru menu)</td>
</tr>
</tbody>
</table>
Undo
close
forward search
reshape cmd
xerox
Replace & edit file
backward search
Read in named Unix file
xerox (walk thru menu)
select text just inserted
send
Substitute
Change directory
sever
Set current file
Rename file
Open grouping
no scroll
Any char except newline
substring from 1st to 2nd address
dot (current position)
Print menu of files
Delete file
Run Unix command
scroll
end of line
Zero or more
pre-set mark
Close grouping
2nd address eval back from 1st
substring from 1st to 2nd address
Print address of range
Search, run command
Print text range
2nd address eval at start of 1st
Replace (pipe)
Character class
Start of line
Character position
Search, run command when not found
One or more
Set position mark
Send range to named Unix command
Search, run command between matches
Search, run command on matches
Zero or One
End of line
Replace text
filename
Figure 1. Relative Command Frequencies for all Users
Figure 1. Relative Command Frequencies for all Users
Figure 3. Proportion of Users Employing Selected Commands

- reshape_sweep
- line number
- load file
- undo

Users % vs. Time Period