MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 338 May 1992

Capture It, Name It, Own It:

How to capture re-occurring patterns, name them and turn them into reusable functions via Emacs kbd-macros

Stefan N. Kozlowski

(**Note: This text was delivered as a lecture to the
AI Lab Support Staff and still appears as such.**)

Abstract

The purpose of this talk is not to teach you about Emacs or Emacs kbd-macros, though we will use both as examples. I can teach you everything there is to know about Emacs kbd-macros in 5 minutes. There are literally only about six commands which govern the majority of the Emacs kbd-macro universe but just knowing the commands is not going to help you much. To borrow an analogy from the introductory 6.001 lecture, I can teach you all the rules of chess in ten minutes but that does not mean that you will be a good chess player in ten minutes. The purpose of this talk is to get you to think about many of the methods and processes we perform each day in our jobs. Hopefully, such an examination will make you realize that we often repeat the same processes over and over. If we can isolate a repeated process, we can often capture it and transform it into a reusable function.

Today we will be looking at capturing such processes via Emacs kbd-macros though, you should be aware that many these of the methods can also be applied to UNIX, other operating systems, editors and languages. The reason we will be examining this topic via Emacs kbd-macros is that it is the easiest and most user-friendly way to approach the subject. We are going to start by looking at very simple examples and progress in complexity. I have written these macros and use some of them on a daily basis. Hopefully some these examples will directly correlate to duties you perform each day at work and you will be able to use some of them.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-89-J-3202 and by the National Science Foundation under grant number MIP-9001651.

Capture It, Name It, Own It:

How to capture re-occurring patterns, name them and
turn them into reusable functions via Emacs kbd-macros

Stefan Kozlowski
MIT AI LAB
March 20, 1992

The purpose of this talk is not to teach you about Emacs or Emacs kbd-macros, though we will use both as examples. I can teach you everything there is to know about Emacs kbd-macros in 5 minutes. There are literally only about six commands which govern the majority of the Emacs kbd-macro universe but just knowing the commands is not going to help you much. To borrow an analogy from the introductory 6.001 lecture, I can teach you all the rules of chess in ten minutes but that does not mean that you will be a good chess player in ten minutes. The purpose of this talk is to get you to think about many of the methods and processes we perform each day in our jobs. Hopefully, such an examination will make you realize that we often repeat the same processes over and over. If we can isolate a repeated process, we can often capture it and transform it into a reusable function.

Today we will be looking at capturing such processes via Emacs kbd-macros though, you should be aware that many these of the methods can also be applied to UNIX, other operating systems, editors and languages. The reason we will be examining this topic via Emacs kbd-macros is that it is the easiest and most user-friendly way to approach the subject. We are going to start by looking at very simple examples and progress in complexity. I have written these macros and use some of them on a daily basis. Hopefully some these examples will directly correlate to duties you perform each day at work and you will be able to use some of them.

1 The Basics:

I am not going to spend much time discussing the basics of kbd-macros for two reasons. The Gnu-Emacs manual already does a fine job of this and the basics are so simple that it would be a waste of our time. By showing examples, you will get a feel for them instantly.

1.1 What is a kbd-macro?

The simplest definition for an Emacs kbd-macro can be found in the Gnu-Emacs manual:

> A "keyboard macro" is a command defined by the user to abbreviate a sequence of keys. For example, if you discover that you are about to type 'C-n C-d' forty times, you can speed your work by defining a keyboard macro to do 'C-n C-d' and calling it with a repeat count of forty.

1.2 How do we make a kbd-macro and use it?

As noted in the introduction, there are really only about six major commands in the Emacs kbd-macro universe but there are really only 3 that you need to start writing macros.

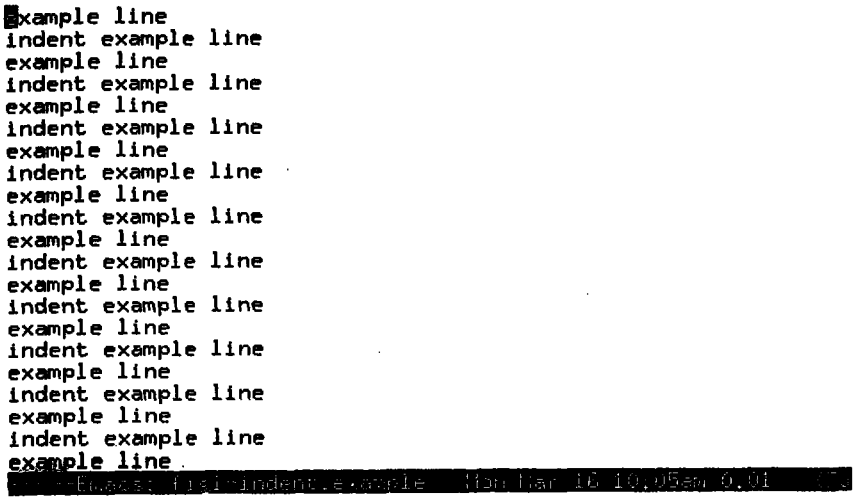

Fig-1: Untouched and unindented example file.

1.2.1 Basic kbd-macro Functions

Command	Explanation
'C-x ('	Start defining a keyboard macro ('start-kbd-macro').
'C-x)'	End the definition of a keyboard macro ('end-kbd-macro').
'C-x e'	Execute the most recent keyboard macro ('call-last-kbd-macro').

Appendix A lists all kbd-macro functions

2 Simple Example Macros:

Knowing only these three commands, we can start to automate many of the repetitive tasks we encounter every day. The following are two simple examples which will demonstrate this.

2.1 Indentation Example

We will start with a very simple example. Figure 1 has "example lines" and "indent example lines" down the screen. We are going to write a very simple macro which will take each "indent example line" and indent it 5 spaces.

The first question we want to ask ourselves is; If we were going to indent these lines by hand, how would we do it? This seems obviously simple. We would:

1. move the cursor down one line or go to the beginning of the first "indent example line"
2. press the <SPC> five times
3. move the cursor down one line

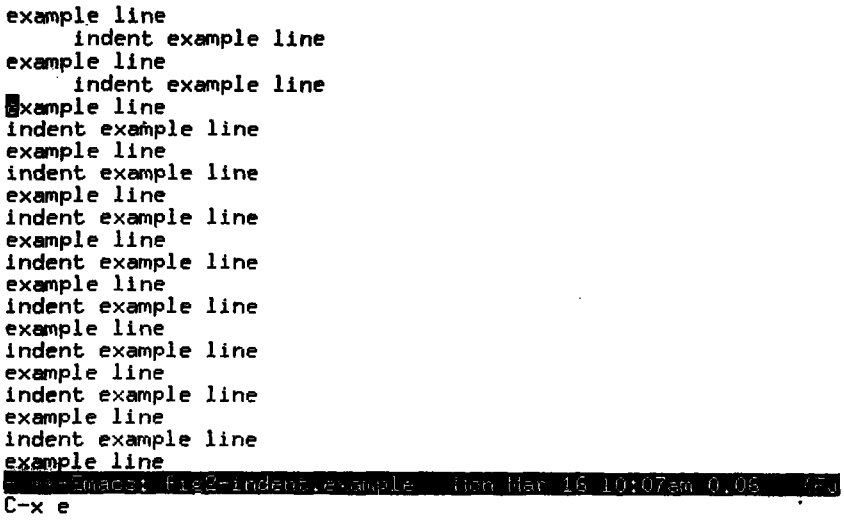

Fig-2: The state of our test file after our new macro has been defined and executed once.

4. position the cursor back at the beginning of the line
5. repeat steps 1-4 until the end of the file

Steps two through four are the heart of this our task. The fact that this is a very trivial example makes it easy to isolate the repeating process. Since we are able to isolate the process, we can capture the method and bind it to a kbd-macro. Using a macro means that we will only have to repeat the task once in order to define the macro itself. Once the macro has been defined, it will do all of the remaining work for us. Therefore, lets write a simple macro that captures steps 1-4.

2.1.1 Indentation Macro:

Step	Command	Explanation
1	'C-x ('	Start defining a kbd-macro. (*note: The message "defining kbd-macro" will appear in the emacs command line after this is executed.*)
2	'C-n'	Move the cursor down one line so that we are at the begining of our first "indent example line."
3	'<SPC>'*5	Indent the line by pressing the space bar 5 times.
4	'C-n'	Move the cursor down one line
5	'C-a'	Go to the begining of the new line
8	'C-x)'	Close the kbd-macro. (*note: The message "kbd-macro-defined" will appear in the command line after this is executed*)

Our kbd-macro is now defined and we are free to use it. To do so, we need to issue the execute command 'C-x e'. The macro repeats the same key strokes that we just executed and our cursor magically jumps down 2 lines and indents the appropriate intermediate line along the way.

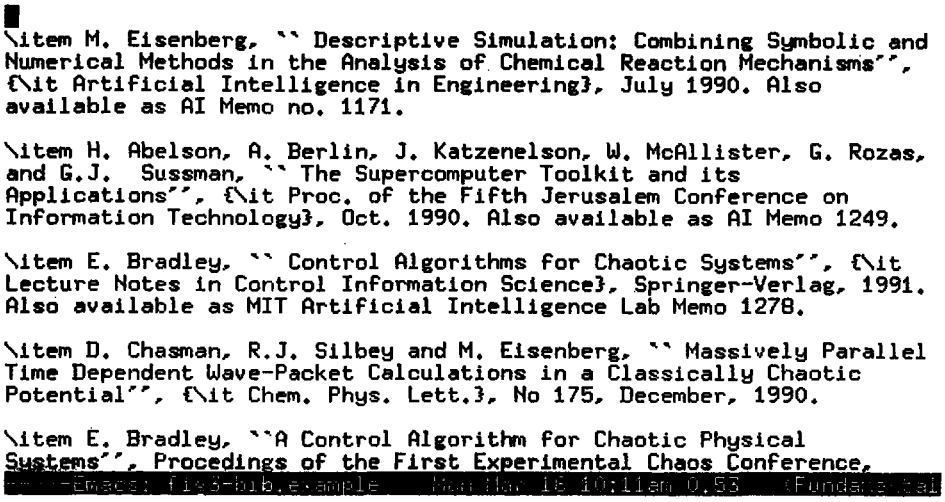

Fig-3: The untouched bibliography file.

So far we have optimized a process which would have taken 9 key strokes [1] down to a single command 'C-x e'. By giving the execute command the prefix argument 'C-u C-u C-u', this macro can apply itself to the entire file.[2]

2.2 Bibliography Example

Our second example of a simple kbd-macro is slightly more complex. Figure 3 displays a file that contains bibliographic citations.

For this example, lets say that one of the members of our research group (Liz Bradley) is graduating and going on to teach elsewhere and we need to remove all of her citations from our group bibliography. We could do this by moving down the file and deleting every citation which bears Liz's name but once again we are repeating a single task over and over. There is no good reason not to write a simple kbd-macro to accomplish the work for us.

2.2.1 Simple Bibliography macro:

Step	Command	Explanation
1	'C-x ('	Begin composing a kbd-macro.
2	'C-s Bradley'	Begin a search on "Bradley".
3	'C-b'	Move back one space to exit the search mode but to keep the cursor in the appropriate paragraph.
4	'M-h'	Mark the current paragraph.
5	'C-w'	Kill the marked region.
6	'C-x)'	Close the kbd-macro.

[1] ('C-n, C-n, <SPC>, <SPC>, <SPC>, <SPC>, <SPC>, C-n, C-a')

[2] Actually, the 'C-u C-u C-u' command will execute our macro 4x4x4 or 64 times which will therefore deal with 128 line pairs. We could have given a single 'C-u' a numeric value to tell it exactly how many times we wished to repeat the procedure. Check your Emacs manual for info about this.

We have now captured the process we wish to repeat in a kbd-macro and the computer will do all of our future searching and cutting for us. When given the execute command ('C-x e'), the editor's cursor instantly jumps forward and removes Liz's next bibliographic entry. Once again, by adding a prefix argument to the execute command ('C-u C-u C-u C-x e') we can ask the kbd-macro to apply itself to the entire file.

3 Macros to Functions: naming and permanently saving kbd-macros

I have called the preceding examples "simple kbd-macros" because they were temporal or meant to be written and used only in a specific instance. Odds are that we will not have to repeat these tasks in the future. For example, Liz Bradley is not likely to become a member of our research group and then leave again. The next several macros we will examine will be of a different type. They will be macros which we would like to keep around permanently in order to reuse them. To keep kbd-macros for the long term, we need to name and save them. The following two commands help us perform these functions.

3.1 Commands to Name and Save Macros

Command	Explanation
'M-x name-last-kbd-macro'	Give a command name (for the duration of the session) to the most recently defined keyboard macro.
'M-x insert-kbd-macro'	Insert in the buffer a keyboard macro's definition, as Lisp code.

As before, we will explore these new commands via specific examples. The following macros should be more relavent to our job duties. I have written them and use many of them every day. The first example we will examine is technically very simple. What is important about this macro is not necessarily what it does but what we are going to do with it after it has been composed.

3.2 Library Example

One of our re-occurring job duties here at the lab is to look up and retrieve reference material from the library. As most of you know, the MIT Library Catalog is online and accessible through various sources. The easiest and quickest way to access the catalog is by telnetting to the AI/LCS reading room. The following macro will automatically connect us to the reading room.

3.2.1 Library Macro

Step	Command	Explanation
1	**'C-x ('**	Begin composing a macro.
2	**'M-x Telnet'**	Invoke a Telnet via emacs.
3	**'reading-room.lcs.mit.edu'**	Give the telnet the address of the reading room at the prompt
4	**'C-x)'**	Close the kbd-macro.

You might notice a difference about this macro when compared to the others. We can invoke it via the execute command **'C-x e** but why should we? By defining it, we have accomplished our task (e.g. we are already logged into the reading room library system). Eventually we will want to connect to the library again but that might not be for a week and the macro we have just defined will go away to the great bit-cemetery in the sky if we either define another kbd-macro or close down our Emacs editor.

What we need is a way to save the macro we have just defined so that we can use it later. As has been alluded to earlier, we can do this by naming the macro and saving it into a file.

3.2.2 Naming Our Macro

Step	Command	Explanation
1	**'M-x name-last-kbd-macro'**	This command enables us to give our new macro a suitable name.
2	**'library'**	At the prompt, we give it an appropriate name

This kbd-macro has been bound to the name **'library'**. By doing so, we have turned it into a **'M-x'** command. This means that we can invoke it like any other command by typing **'M-x library'**. While this is useful because it frees us up to define another kbd-macro, we still have a problem. Our "library" macro will only continue to exist as long as the current Emacs environment exists. Therefore, if we were to close our editor, **'M-x library'** will still go away to the great bit-cemetery in the sky. We need a way to save our macro for later use. The command **'M-x insert-kbd-macro'** serves this function.

3.3 Saving Our Macro: Where & How

There is an issue we must address before we try to save our macro. Where should we save it? We have two options. We could save it into our "~/.emacs" file. If we insert our macro and save it here, we will be assured that the command **'M-x library'** is always available. This is because our "~/.emacs" file is automatically loaded each time we bring up an Emacs editor.

Another option would be to create a new file named something like "macros.el" where all of our user-defined macros would reside. This file would not automatically load when we opened a new editor. In order to use our macros, we would have to explicitly tell Emacs to load this file either via the command **'M-x load-file'** or by placing some emacs-lisp code in a file calling the function 'autoload' with a pointer to our "macros.el" file.[3]

Most of the time, it is easier and simpler to just insert our macros directly into our "~/.emacs" file. This way we don't have to worry about them and they are always available

[3]See your Gnu-Emacs manual for more information on these processes.

```
;;KBD Macros

(fset '\six
   "^X^Fhal/6001-grad.^?e.tex^M^N^N^N^F^F^F^F^F^F^F")

(fset 'dl
   "^X^S^X^B^Xo^S.tex^@\342\342\367\370shell-command^Mlatex ^Y^M^Xk^M\
^Xo^X1^Xq\370shell-command^Mslpr ^Y^?^?^?dvi^M^Xk^M\370shell-command^\
Mslpq ^Y^_^L^M")

(fset 'hlet
   "\274^@\276^W^Xi"/rec/hal/let^M^S,3^N^Y^R[]^?^?{3^F^@")

(fset 'lib-adds
   "^X^F^A^K~/lib/adds^M")

(fset 'lode
   "\370shell-command^Mload^M")

(fset 'new-student-scm  ;;start a new student-scheme
   "^U\370run schem ^M -constant 1100 -heap^?p 1500 -band 6001.com^M"\
-:**-Emacs: .emacs                  Mon Mar 16 10:15am 0.04    (Emacs-Lisp)
```

Fig-4: Unaltered "~/.emacs" file.

for use. However there may be times when we will want to create distinct macro files that only load when asked but these issues are beyond the scope of this talk. We will proceed by demonstrating the method for saving our **'M-x library'** function into our "~/.emacs" file.

3.3.1 Inserting and Saving a Macro

In figure 4, you will see a copy of the macro section of my "~/.emacs" file. The cursor is positioned at the place where I would like to insert our "library" macro. (I like to keep them all in alphabetical order. This makes them easy to find when you have defined a large bunch.) By executing the following commands, our library macro will become a permanent part of my Emacs editor.

Step	Command	Explanation
1	**'M-x insert-kbd-macro'**	Insert a kbd-macro's definition at the current cursor position
2	**'library'**	Give the name at the prompt of the macro to be inserted
3	**'C-x C-s'**	Save our altered "~/.emacs" file.

Figure 5 shows us the result of the result of executing step number two. The macro is not very readable to us but it makes perfect sense to our Emacs editor. Now that we have inserted and saved our "~/.emacs" file we are finished. **"M-x library"** will exist until we explicitly remove it.

While our library macro is useful, it is neither terribly complex nor is it a command we will use every day. I used a very simple example because I really wanted to demonstrate the idea of transforming a macro into a permanent function. The next example should be a macro we can use frequently.

```
;;KBD Macros

(fset '\six
   "^X^Fhal/6001-grad.^?e.tex^M^N^N^N^F^F^F^F^F^F^F^F")

(fset 'dl
   "^X^S^X^B^Xo^S.tex^@\342\342\367\370shell-command^Mlatex ^Y^M^Xk^M\
^Xo^X1^Xq\370shell-command^Mslpr ^Y^?^?^?dvi^M^Xk^M\370shell-command^\
Mslpq ^Y^_^L^M")

(fset 'hlet
   "\274^@\276^W^Xi"/rec/hal/let^M^S,3^N^Y^R[]^?^?{}^F^@")

(fset 'lib-adds
   "^X^F^A^K"/lib/adds^M")

(fset 'library
   "\370telnt^?et^Mreading-room.lcs.mit.edu^M")

(fset 'lode
   "\370shell-command^Mload^M")
-Emacs: .emacs                    Mon Mar 15 10:19am 0.04    (Emacs-Lisp)
```

Fig-5: The contents of the "~/.emacs" file after inserting our new macro.

3.4 Letter Example

Several times a week, I receive email from Hal asking me to produce a letter for him (See figure 6). I quickly realized that there was a pattern to performing this task but writing a macro for it created a problem. The process I was repeating was easy to spot.

1. mark the region of the email message containing the text of the letter
2. write the text out to an appropriately named file
3. insert a LaTeX letter template around the text of the letter
4. address the letter

The problem arose during step two of this list. Each time I repeated this process the file name would be different and therefore it seemed that writing a macro to complete the entire task would not be possible. But as is always the case when dealing with Emacs; if you can think of a useful function, it already exists. I needed some way to pause the macro at step two to enable me to give the appropriate file name and then let the macro resume. The command **'C-x q'** or (kbd-macro-query) does exactly this. The official definition of this command is:

Command	Explanation
'C-x q'	When this point is reached during macro execution, ask for confirmation ('kbd-macro-query').

What this means for us is, we now have a way to make kbd-macros interactive. We can therefore write macros that pause for instructions before resuming their execution. This ability can solve a multitude of problems and enable us to write much more complex and powerful macros. Lets write a macro to help us produce Hal's extremely IMPORTANT letter.

```
Return-Path: <hal@altdorf.ai.mit.edu>
Date: Wed, 11 Mar 92 17:59:21 -0500
From: "Hal Abelson" <hal@altdorf.ai.mit.edu>
To: stefan@altdorf.ai.mit.edu
Subject: Make and send a letter to Eileen.
Reply-To: hal@altdorf.ai.mit.edu


Please format this, put it on AI Lab stationary, sign it for me and
send it to Eileen.

****************

Please make sure Stefan gets a $10k a year raise from our DARPA
account before he leaves the lab this summer. He certainly deserves
it and I have been meaning to get around to it for a while.

<formal signature>
```

Fig-6: E-mail message to be transformed into a letter.

3.4.1 Letter Macro

There are several steps of this process that we can not bind into our kbd-macro. They are : the marking of the region (each piece of e-mail will be different), the naming of the ".tex" file (all letters will have a different name) and the addressing of our letter. These are the only unique aspects of this process. Our new **'kbd-macro-query'** command will enable us to bind everything but these steps into a macro.

Step	Command	Explanation
1	'C-<SPC>' 'C-n'	Mark the region which contains the text of our letter
2	'C-x ('	Begin composing our macro.
3	'M-x write-region'	Write the region containing the text to a file.
4	'~/hal/.tex'	Give the path and ".tex" extension
5	'M-b' 'C-b'	Move back near the begining of our new files name, using the meta-command to skip over the "tex" extension and the control-command to place the cursor on the dot.
6	'C-x q'	Enter the pause command so a macro will stop here during execution.

Our macro is not finished yet but I would like to stop for a moment and discuss what has occurred in steps 4-6. Figure 7 shows the state of our screen at this moment. The premise of step 4 comes from the fact that I keep all of the letters I create for Hal in the directory "~/hal" therefore I know that whatever filename I give to a particular letter, it will always have the same path. Since the path is always the same, I add it to my macro definition. Similarly, I know that any letter I produce for Hal will always be a ".tex" file and therefore I will also include this extension in my macro. The two commands in step 5 are used to

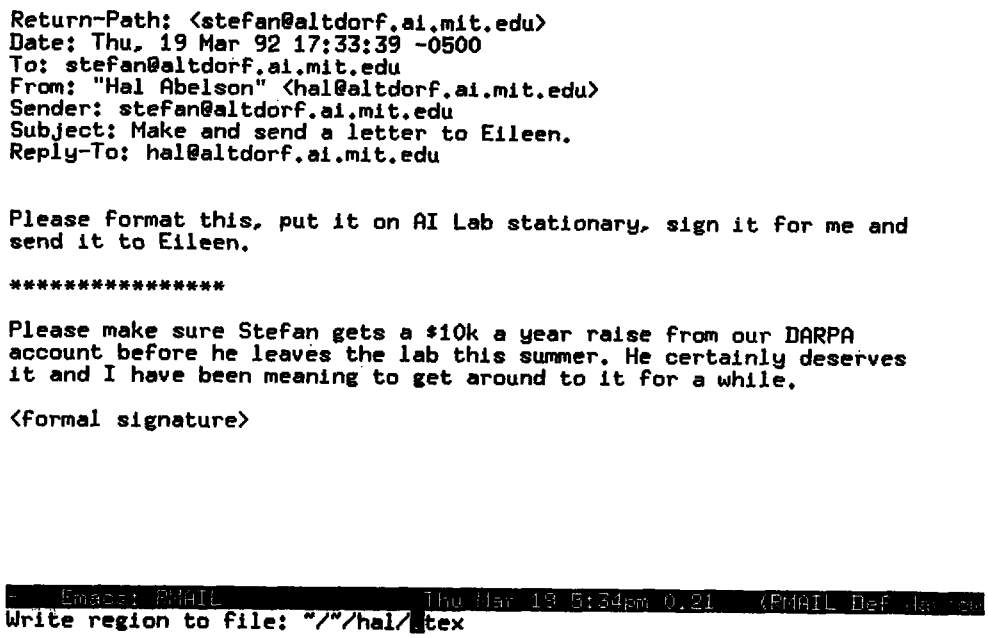

Fig-7: The position of our cursor after step 6 of our definition.

position my cursor so that it is sitting on the dot directly before the "tex" extension when the macro pauses. This way I only have to type the unique filename (in this case something like "10k-raise") and nothing more before telling the macro to resume executing.

Adding such specific information to the macro increases it's complexity and therefore it's specificity. I could have just told the macro to pause after I had invoked the **'M-x write-region'** command in step 3 and omitted steps 4 and 5 altogether. During execution, I would have to explicitly write out the path and the file name. Such an approach would have generalized my macro to the point where I could use it to produce letters for not only Hal but for Gerry as well. This is a stylistic choice I am making. I prefer to squeeze every bit of optimization I can from my kbd-macros. I view it as a challenge. If I want a macro to create letters for Gerry, I will write one. You may not wish to be this fanatical when attempting to use this macro. If so, cut commands 4 and 5 out altogether.

Step	Command	Explanation
7	'C-a'	Go to the begining of the line
8	'C-k'	Place the path and file name into the emacs-kill ring
9	'C-y' <RTN>	Return the path and file name back to the 'M-x write-region command and execute
10	'C-x C-f'	Find File
11	'C-y' <RTN>	Give the current contents of the kill ring to the 'find-file' command and execute
12	'C-<SPC>'	Set the mark at the begining of the file
13	'M->'	Goto the end of the file
14	'C-w'	Remove the enire contents of the file from the current buffer and place it in the kill ring.
15	'C-x i'	Insert a file into the current buffer
16	"/hal/let" <RTN>	Give the name of my LaTeX letter template file and insert it into the current buffer. I am assuming that we all have and use a LaTeX letter template for composing letters. See Appendix C or Fig-9 for an example of such a template.
17	'C-s ,}'	Search for the appropriate place to insert the text of our letter which is still sitting in the kill ring.
18	'C-n'	Get out of the search mode and place the cursor at the correct position
19	'C-y'	Insert the text of our letter
20	'C-r {}'	Search backward to find the address section of our template
21	'C-f'	Exit the search mode with cursor at the desired position
22	'C-x)'	Finish composing our kbd-macro
23	'M-x name-last-kbd-macro'	Name something this complex immediately
24	'hal-let'	Choose an appropriate name you will remember.

Our first complex macro is now defined and named ('M-x hal-let') but even if you managed to keep straight everything we did in the last 24 steps, their is still an issue I have failed to address. We still have not seen what the mysterious 'C-x q' or ('kbd-macro-query') command is actually going to do during execution. We will turn our attention to this subject now.

3.5 Executing A Macros Which Contain Pause Commands

We are going to use 'M-x hal-let' to create a letter. We begin by returning to the original email message and executing the following steps.

```
Return-Path: <stefan@altdorf.ai.mit.edu>
Date: Thu, 19 Mar 92 17:33:39 -0500
To: stefan@altdorf.ai.mit.edu
From: "Hal Abelson" <hal@altdorf.ai.mit.edu>
Sender: stefan@altdorf.ai.mit.edu
Subject: Make and send a letter to Eileen.
Reply-To: hal@altdorf.ai.mit.edu


Please format this, put it on AI Lab stationary, sign it for me and
send it to Eileen.

****************

Please make sure Stefan gets a $10k a year raise from our DARPA
account before he leaves the lab this summer. He certainly deserves
it and I have been meaning to get around to it for a while.

<Formal signature>
```

```
-- Emacs: RMAIL                    Thu Mar 19 5:38pm 0.18 Mail    (RMAIL [w] --
Proceed with macro? (Space, DEL, C-d, C-r or C-l)
```

Fig-8: The state of our screen after the **'M-x hal-let'** macro executes step 6 ('kbd-macro-query').

Step	Command	Explanation
1	**'C-<SPC>'** **'C-n'**	Set the mark at the beginning of the letters text and move downward eventually positioning the cursor at the end of the text.
2	**'M-x hal-let'**	Invoke our new macro/command

' M-x hal-let' is now going to execute steps 2-6 described in the formal definition of the macro. When the function reaches step 6, it will pause and ask us for instructions. Figure 8 displays the state of our screen at this moment. We are asked if we want to proceed with our macro and are given a number of potential options. What all of these options mean are listed in Appendix B of the accompanying paper to this talk. For our purposes at the moment, only the following commands are relavent.

Special 'kbd-macro-query' Commands

Command	Explanation
'C-r'	Enter a recursive editing level, in which you can perform editing which is not part of the macro.
'C-M-c'	Exit macro editting mode. Note that this command is not displayed as an option in our command line at this time. We will use it to finish editting (in this case giving the file name) and want to resume our macro.
<space>	Continue executing current macro.

In this case, the first command will enable us to enter our file name and the second command will tell our macro to resume after we have entered it. This will become clearer after we see it happen.

```
\documentstyle[11pt]{letter}

\address{Harold Abelson\\
545 Technology Square\\
Cambridge, MA  02139\\
NE43-410}

\signature{Harold Abelson\\
Professor of Computer Science\\
and Engineering\\
MIT}

\begin{document}
\begin{letter}{}

\opening{Dear ,}
Please make sure Stefan gets a $10k a year raise from our DARPA
account before he leaves the lab this summer.  He certainly deserves
it and I have been meaning to get around to it for a while.


\closing{Sincerely,}

\end{letter}
\end{document}

-- -Emacs: 10k-raise.tex       Thu Mar 19 5:48pm 0.08 Mail    (TeX Fill)-
Auto-saving...done
```

Fig-9: Screen display after our macro has finished executing.

Step	Command	Explanation
3	**'C-r'**	Enter editing mode so we can give our macro a filename
4	**"10k-raise"**	Enter an appropriate file name. (Note that our cursor is positioned exactly where would ideally want it and that our path is already defined.)
5	**'C-M-c'**	Exit macro editing mode. This command returns us back to the state exhibited in figure 7.
6	**<SPC>**	Continue executing current macro

'M-x hal-let' will now resume it's function and instantly our new letter file appears before us with the cursor sitting at the exact position to edit the recipients address. Figure 9 shows our screen display after the macro has finished executing. This macro has reduced what was a complex multi-step process down to 6 steps and many of them (steps 3,5-6) are extremely trivial.

4 Advanced Macros That Call UNIX

Until this point, all of our kbd-macros have manipulated pieces of text. The only way we have used our operating system was through Emacs commands which hid most of the detail from us. For example, when we copy one file to another in Emacs, the editor is invoking the UNIX command "cp" for us. We just never see the shell command being executed. This next example will show us that we can easily deal directly with our operating system.

4.1 LaTeX Example

The last macro we examined performs a good deal of work for us but there is no reason to stop optimizing now. **'M-x hal-let'** completes it's execution by leaving us in position to insert unique addressee information. After we have addressed the letter, we will once again

begin repeating a process.

1. save the current buffer/file
2. switch to a UNIX shell and LaTeX the file
3. print the created ".dvi" file
4. check the print queue

There are no universal Emacs commands which can perform steps 2-4 of this process for us. Therefore, if we are going to write a macro to automate this task, we will need to talk directly to our operating system (UNIX). Emacs makes this simple via **'M-x Shell'** commands.

Once again we will return to our "10k-raise.tex" file and use it as an example to define our "do latex and print" macro. I am going to move through the creation of this macro with less explanation than I have offered for previous examples. Watching me do it can give you a feel for the methods involved but nothing can replace the pedagogic value of trying it yourself. This macro will generally work on any Emacs editor. The only changes you will need to make are regarding the name of your local printer.

4.1.1 LaTeX Macro

Step	Command	Explanation
1	'C-x ('	Begin defining kbd-macro
2	'C-x C-s'	Save the current buffer
3	'C-x C-b'	List current buffers
4	'C-x o'	Switch Buffer to the other buffer
5	'C-s .tex'	Position the cursor past the ".tex" of the filename
6	'C-<space>'	Set the mark
7	'M-a'	Backward one sentence
8	'M-f'	Forward one word
9	'M-b'	Backward one word (Commands 5-9 are used to retrieve the current filename. This will make more sense when you watching the cursor movements on a screen.
10	'M-w'	Copy region as a kill (grab the file name)
11	'C-x o'	Switch back to the file buffer
12	'M-x Shell-command'	Invoke a shell command
13	'latex C-y'	Give the latex command on the filename you have just retrieved (steps 3-10) at the shell prompt
14	'C-x o'	Change buffers to the shell-command-output buffer
15	'M->'	Goto the end of the buffer
16	'C-x q'	kbd-macro-query, pause to make sure that the LaTeX worked
17	'C-x o'	Switch back to our ".tex" file/buffer
18	'C-x 1'	Remove our shell-command-output buffer
19	'M-x shell-command'	Invoke another shell command prompt
20	'lpr -P'<foo> '-d C-y'	Give the printer name and empty the filename from the kill ring into the shell command line
21	'<backspace>'*3	Remove the ".tex" extension.
22	'.dvi' '<RTN>'	Replace it with ".dvi" and execute the shell command.
23	'M-x shell-command'	Invoke another shell command
24	'lpq -P'<foo>	Give the appropriate lpq command
25	'C-x)'	Close the kbd-macro
26	'M-x name-last-kbd-macro' 'do-latex'	Name macro immediately and appropriately

Much like our 'library' macro, we have completed our intended task by defining the macro 'do-latex'. The generated ".dvi" file is now printing at the designated printer. We should immediately save this macro into our "~/.emacs" file just to be safe. For demonstration purposes, I will now execute 'M-x do-latex'.

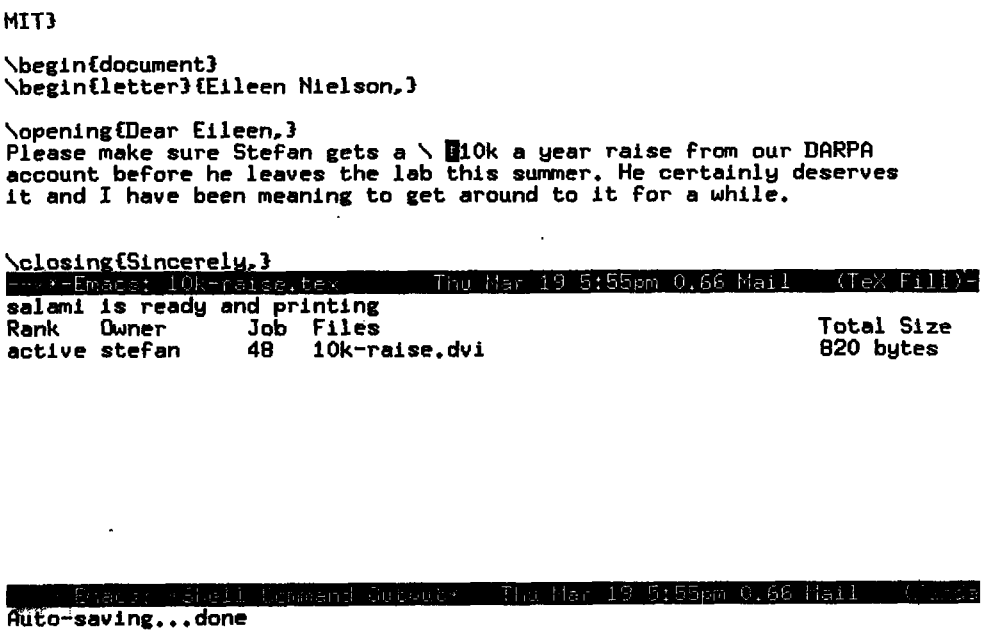

Fig-10: The screen display after **'M-x do-latex'** has finished executing.

4.1.2 Executing LaTeX Macro

First we must switch back to our "~/hal/10k-raise.tex" buffer that was created by the command **'M-x hal-let.**

Step	Command	Explanation
1	**'M-x do-latex'**	Invoke our new command.
2	**<SPC>**	Our macro paused and asked me if I wished to continue. Seeing that LaTeX created a ".dvi" file I give the resume command to our macro. If the shell-command buffer had displayed that an error had occurred during the LaTeX process, I would have issued the command **'C-d'** which would have halted the macros execution. See appendix C for more information.

Figure 10 shows the state of our screen after our macro has completed it's task. The print queue is visible in the second window and I can easily gauge when our I should stop by the printer and retrieve my finished document. Once again, we have reduced a multi-stage process down to a single command name.

Some of you now might realize something about the last two examples I have demonstrated today. There is no reason why I couldn't have combined **'M-x hal-let'** and **'M-x do-latex'** into one big "super-macro" and called it **'M-x hal-let-do-latex'**. This macro would have executed all the tasks of both macros at once. I didn't do this because defining something so massive might have been confusing to you. Nonetheless, I just happen to have such a macro defined and I will now demonstrate it for you.

We will now return to our original e-mail message, mark the region and issue the command **'M-x hal-let-do-latex'**. The only difference you will see between executing this "super-macro" as opposed to invoking **'hal-let'** and **'do-latex'** concurrently is one additional pause command. The state of our editor after **'M-x hal-let'** has finished, leaves

us at the point where we have to address the "10k-raise.tex" letter. By inserting a pause command here instead of ending our macro, we can still accomplish this task and have our "super-macro" continue onward and complete all the steps listed in **'M-x do-latex'**.

5 Procedural Abstraction in Macros

Some of you might be cringing at the thought of defining a macro which has 50 or so steps but the power of procedural abstraction enables us to compartmentalize these steps to control the complexity of the macro. Defining a "super-macro" like **'M-x hal-let-do-latex'** is actually one of the simpler macros we will see today. I will demonstrate how we could compose such a macro.

5.1 "Super-Macro" Example

Three of the macros we have seen to day ("library, hal-let and do-latex") have been transformed from macros into permanent functions. Therefore, we should not view them any differently than all other Emacs 'M-x' commands. Since this is the case, we can use these macros/functions in the definition of other macros.

5.1.1 'hal-let-do-latex' Macro

Step	Command	Explanation
1	**'C-x ('**	Begin defining a kbd-macro.
2	**'M-x hal-let'**	Invoke our first macro
3	**'C-r'**	Enter edit mode to give our filename
4	**'10k-raise.tex'**	Enter file name
5	**'C-M-c'**	Exit edit mode
6	**<SPC>**	Resume our kbd-macro.
7	**'C-x q'**	Insert a pause command where **'M-x hal-let'** finishes.
8	**'M-x do-latex'**	Call our LaTeX macro.
9	**<SPC>**	Check to see our LaTeX command worked and resume our macro
10	**'C-x)'**	Close our kbd-macro.
11	**'M-x name-last-kbd-macro' 'hal-let-do-latex'**	Name our kbd-macro appropriately.

The fact that we were able to define and name our 'letter' and 'latex' macros, means that we are able to use these commands like any other. Therefore the formal definition of **'M-x hal-let-do-latex'** is not terribly complicated. We have used procedural abstraction to reduce the complexity of this macros definition. Instead of containing 50+ steps, it is only 10 steps in length. I could now launch into an entirely new lecture about the use of procedural abstraction in kbd-macros but this is well beyond the scope of this talk. I would like to stop myself here by saying that we can create many of our macros in a generic way that enables us to easily string them together.

6 Advice & Conclusion

I hope that all of you who haven't experimented with kbd-macros before will now start trying to write them. You might wish to simply follow the steps that have been laid out in this talk explicitly and produce your own versions of the 3 macros we have seen today but it is probably better if you don't start with something a complicated as **'M-x do-latex'**. Start with the 'library' example we saw earlier. It is simple and very useful. I have an entire file loaded with very simple 5 to 10 step macros which I use each day. With a little bit of thought, you can quickly find some tasks which just cry out to be automated.

6.1 Help With Emacs

We might have used some Emacs commands today which you never knew existed before. Do not be discouraged by this. Nobody, except for Stallman, probably knows all if not most of the commands which the editor has available. There are several help options online which will help you find out more about Emacs commands.

6.1.1 Online Help

Command	Explanation
'M-x info'	This will display a HUGE, menu-driven data base about your local system. One of the options will have the entire gnu-emacs manual beneath it.
'M-x apropos'	This function asks the user for a keyword and then displays every function which has that keyword in it's name. For example, if you gave this function the keyword "column", it would display a list of every function that has "column" in it's name.
'M-x describe-function'	This function will give the user a prompt, at which you type any command name. A written explanation of what the command does is then returned
'M-x describe-key'	This function returns a prompt at which the user types any control command. The function then returns all information about the control command. For example, if you gave this function the argument **'C-x C-s'**, **'M-x describe-key'** will return a buffer containing all information about saving files.

Another simple way to improve your knowledge of Emacs is to type **'M-x ?'**. This will produce a list of almost all Emacs commands. You can then print this list out and look it over from time to time. For the first several years I was here, I posted such a list on my wall next to my computer. Every now and then, I would look for an interesting name on the list and experiment with it. I probably learned more about Emacs this way than I would have ever learned reading the manual. Also, the command **'M-x help-for-help'** is a function which packages together a large bunch of help commands together.

6.2 Good Luck

What I hope you take away from this talk is not a cook book list of macros to make your job easier. What I hope today's talk will do, is to make you start to think about the patterns we repeat when we perform our job duties. We have seen how some tasks can be viewed as repeating processes. If these processes can be isolated enough, we are able to convert them into permanent, reusable functions. We have used Emacs kbd-macros as the vehicle to examine such optimizations but these ideas are not only for use with Emacs. The ability to control processes by naming them is a powerful idea which appears in many realms of computing. For example, we can produce similar optimizations in UNIX by defining shell scripts and aliases. After using kbd-macros for a while, you might find that your appetite has been wetted for more complex projects. If this becomes true, the ideas and methods you have employed while working with kbd-macros will help you considerably.

Appendix

[A]List of All Major Emacs kbd-macro Commands

Command	Explanation	M-x Equivelant
'C-x ('	Start defining a keyboard macro.	**'start-kbd-macro'**
'C-x)'	End the definition of a keyboard macro	**'end-kbd-macro'**
'C-x e'	Execute the most recent keyboard macro	**'call-last-kbd-macro'**
'C-u C-x ('	Re-execute last keyboard macro, then add more keys to its definition	*none*
'C-x q'	When this point is reached during macro execution, ask for confirmation	**'kbd-macro-query'**
'M-x name-last-kbd-macro'	Give a command name (for the duration of the session) to the most recently defined keyboard macro	*NA*
'M-x insert-kbd-macro'	Insert in the buffer a keyboard macro's definition, as Lisp code	*NA*

[B]All kbd-macro-query Commands

Command	Explanation
'C-x q'	When this point is reached during macro execution, ask for confirmation ('kbd-macro-query').
'C-r'	Enter a recursive editing level, in which you can perform editing which is not part of the macro.
'C-M-c'	Exit macro editting mode. Note that this command is not displayed as an option in our command line at this time. We will use it to finish editting (in this case giving the file name) and want to resume our macro.
'<SPC>'	Continue executing current macro.
'<DEL>'	Skip the remainder of this repetition of the macro, starting again from the beginning in the next repetition.
'C-d'	Skip the remainder of this repetition and cancel further repetition.
'C-l'	Redraws the screen and asks you again for a character to say what to do.

[C]Example LaTeX Template

```
\documentstyle[11pt]{letter}

\address{Harold Abelson\\
545 Technology Square\\
Cambridge, MA  02139\\
NE43-410}

\signature{Harold Abelson\\
Professor of Computer Science\\
and Engineering\\
MIT}

\begin{document}
\begin{letter}{}

\opening{Dear ,}


\closing{Sincerely,}

\end{letter}
\end{document}
```