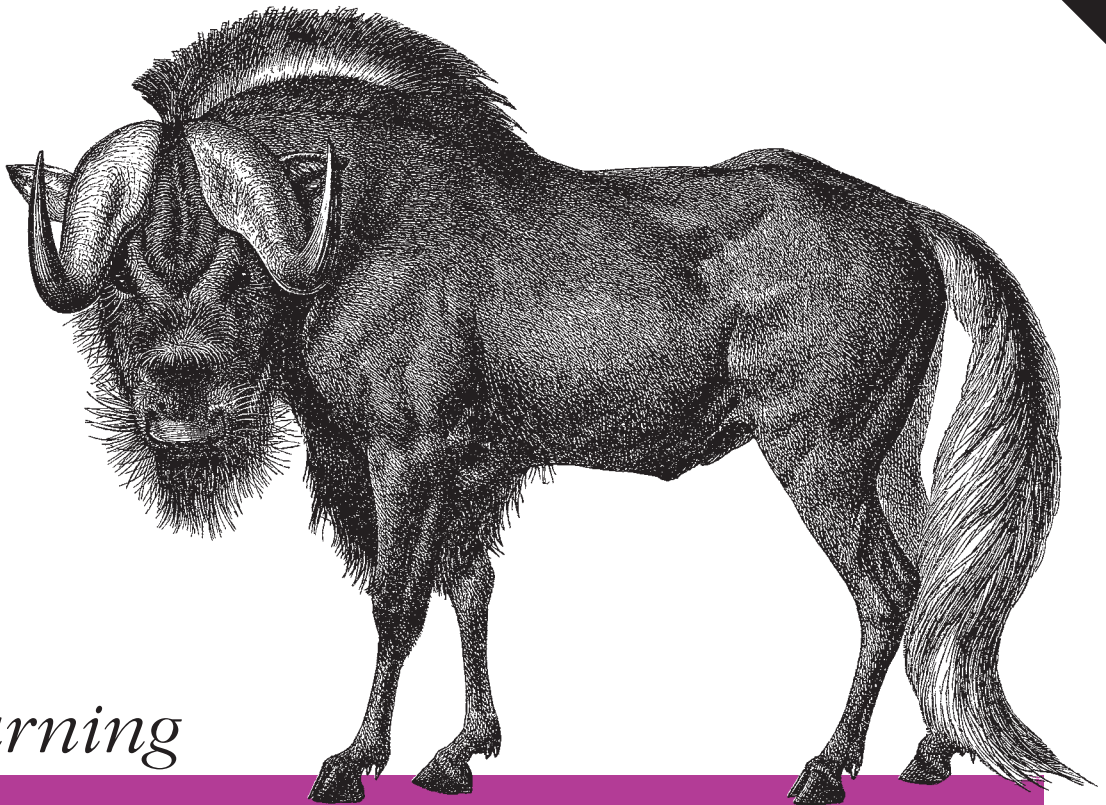


*A Guide to the World's Most Extensible,
Customizable Editor*

3rd Edition



Learning

GNU Emacs

O'REILLY®

*Debra Cameron, James Elliott,
Marc Loy, Eric Raymond & Bill Rosenblatt*

Writing Macros

What is a macro? In Emacs, a *macro* is simply a group of recorded keystrokes you can play back over and over again. Macros are a great way to save yourself repetitive work. For example, let's say you want to delete the third column of a table. Normally, you would go to the first line; move over to the third column; delete it; then go to the second line; give the same set of commands; and so on, until you finish, your fingers wear out, or you get too bored. Emacs lets you record the keystrokes you used to work on the first line of the table, and then “play these back” repeatedly until the job is done.*

Any command or action you do within Emacs, from typing text to editing to switching buffers, can be done within a macro. The key to using macros well is, not too surprisingly, recognizing when you're doing repetitive work: sensing that you have pressed more or less the same sequence of keys several times in a row. Once you learn to recognize repetitious work, you have a good feel for when to use macros. The next talent that you'll need is, given that you've recognized a cycle of “almost identical” keystrokes, figuring out how to make that cycle *precisely identical*—that is, figuring out a set of keystrokes that, if repeated, will do exactly what you want. Neither of these skills is particularly difficult; with a little practice, you'll be using macros all the time.

If this sounds like lazy man's programming, it is: macros give you a simple way to do very complicated things without learning Lisp and without learning any customization tricks. If the task you build the macro for is something you have to do frequently, you can save macros and load them when you want to use them. In this way, you can build up a set of convenient macros that become your own editing commands. Even if you don't write Lisp, you're not limited to the commands Emacs gives you; you can make your own!

* You could delete the third column of a table by marking it as a rectangle, as described in Chapter 7. But bear with us for the sake of making this point: when you find yourself doing repetitive work, macros are the tool to remember.

What you use macros for will depend on the kind of work you do in Emacs. We've used macros to:

- Mark up text for formatting.
- Copy headings from one buffer to another to create an outline.
- Perform complex search-and-replace type operations that query-replace can't quite handle.
- Create index entries.
- Reformat files that were imported from another application.
- Edit tables.
- Compile, run, and test the output from a program with a single command.
- Manipulate and clean large datasets.

You'll be able to think of many more things to do with macros after you learn the few basic commands you need to use them.

A Macro Revolution

In this book, we almost never emphasize which version of Emacs we're talking about. Macros, specifically changes to macros in Emacs 21.3.5, have forced our hand. Macros underwent a major overhaul in 21.3.5. Although some of the core key bindings still work the same way, the keyboard macro functionality was radically expanded. If you are running an earlier version of Emacs, we encourage you to install the latest version (see Chapter 13) or go to the web site for this book, <http://www.oreilly.com/catalog/gnu3/>, which includes a link to an earlier version of this chapter.

Defining a Macro

To start defining a macro, press **F3** or **C-x** (* The abbreviation **Def** appears on the mode line, showing that you are in macro definition mode. In this mode, Emacs records all the keystrokes that you type, whether they are commands or literal text, so that you can replay them later. To end the macro, press **F4** or **C-x**); you leave macro definition mode, and Emacs stops recording your keystrokes. Emacs also stops recording your keystrokes automatically if an error occurs or if you press **C-g**.

* Mac OS X users may have bound **F3** and **F4**, used in defining and executing macros, to another key. These users should press **Option-F3** and **Option-F4** to get the same functionality.

While you're defining a macro, Emacs acts on your keystrokes as well as recording them: that is, anything you type while in macro definition mode is treated as a regular command and executed. While you're defining a macro, you're doing completely normal editing. That way you can see that the macro does exactly what you want it to, and you can cancel it (with **C-g**) if you notice that the macro isn't really quite what you want.

To execute your macro, press **F4** or **C-x e**. Emacs then replays your keystrokes exactly. (You can see that **F4** has two different functions relating to macros: to end a macro definition and, after it's defined, to execute the macro.)

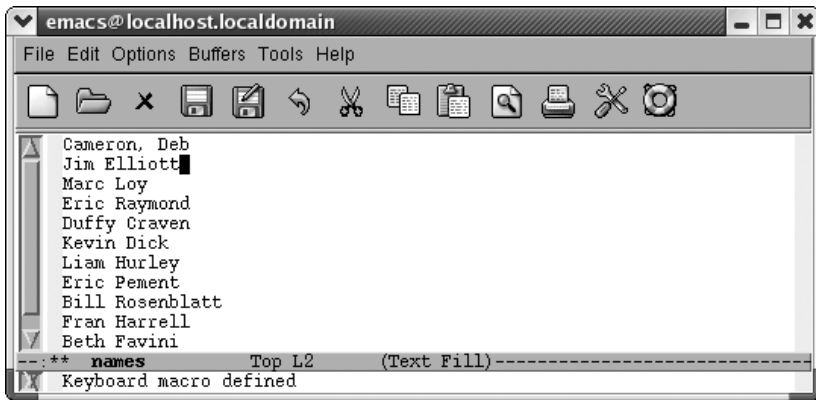
This macro is referred to as the "last" keyboard macro, with last here meaning most recent. Only one macro is the last keyboard macro. A macro ring, much like the kill ring, allows you to access a number of macros during an Emacs session.

Table 6-1 shows the steps required to define and execute a macro. This macro takes a list of names in the ordinary First Name Last Name order and changes it to the frequently needed Last Name, First Name order.

Table 6-1. Steps for creating name transposition macro

| Keystrokes | Action |
|---------------------------|---|
| F3 or C-x (| Start the macro; <code>Def</code> appears on the mode line. |
| C-a | Move to the beginning of the current line. |
| M-f | Move forward a word. |
| , | Type a comma. |
| M-t | Transpose first and last. |
| C-n | Move to the next line. |
| F4 or C-x) | End the macro definition. |

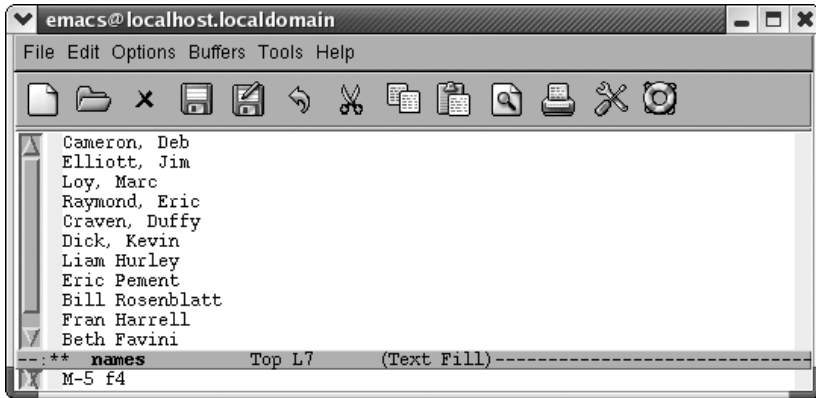
Define the macro using the keystrokes given in Table 6-1.



In defining the macro, you transposed the names on the first line, leaving the cursor on the second line.

Now let's be brave and assume the macro works; we'll try repeating it five times by prefacing the command to execute a macro with **M-5**. Of course, in real life, you'd be better off trying it once before doing anything so bold.

Type **M-5 F4** or **M-5 C-x e**



Now we've done the first six lines: one by defining the macro and five more by executing it.

The macro works well, so we can finish the rest of the buffer with confidence: type **M-100**, then **C-x e** or **F4**. Emacs stops automatically when you reach the end of the buffer, so it doesn't matter if you repeat the macro more times than necessary.

Here are a few points to remember:

- Don't forget to press **F4** or **C-x**) when you've finished the macro. If you try to execute a macro before it has been defined, Emacs complains and forgets the macro's definition.
- **C-g** terminates a macro, causing Emacs to forget its definition.
- Virtually any error automatically terminates a macro. If Emacs beeps at you, you have to start over.
- Emacs executes the keystrokes *exactly* as you type them, with no intelligence whatsoever. Avoid making assumptions like, "Of course I'll be at the beginning (or end) of the line when I execute the macro."

If you invoke a macro and it does the wrong thing, you can use **C-_** to undo it. Emacs is smart enough to realize that "undo the last command" means "undo the entire macro" rather than "undo the last command within the macro." However, if you repeat a macro multiple times using **M-n**, **C-_** undoes only the last instance of the macro, not all the instances.

Tips for Creating Good Macros

It's easy to learn how to record and reuse your keystrokes. However, when you're starting out, you make a few mistakes: you create a macro, use it, and then find out that it doesn't do exactly what you thought. With a little care, it's easy to make your macros more useful and less vulnerable to mistakes.

Good macros work in all situations. Therefore, within a macro, you should use commands that are absolute rather than relative. For example, if you write a macro that puts a formatting string around the word the cursor is on, you want the macro to work no matter how long the word is. Therefore, you would use an absolute command such as **M-f** (for **forward-word**) rather than a few **C-f**s to move forward one character at a time. Similarly, commands such as **C-e** and **C-a** are good for finding the beginning or end of a line rather than moving the cursor forward or backward.

Often, macros start with a search command that brings you to the place in the file you want the macro to start. It's a good idea to type the search argument (as in **C-s searchstring**) rather than using the command to repeat the last search (**C-s C-s**). You may have changed the search string between the time you define the macro and the time you execute it, and **C-s C-s** remembers only what the last search string was.

It is often a good idea to add extra commands (typically **C-a** and **C-e**) that aren't strictly necessary, just to make sure that you're positioned correctly on the line. The fewer assumptions that a macro makes, the better it works. So, if a sequence of commands works correctly only if you start at the end of the line, start the macro with **C-e**, even if you already "know" that you want to give the command only when you're at the end of the line.

Finally, while we're reciting rules and cautions, here's one more: keep in mind that you probably want to execute macros repeatedly. With a little foresight, you'll be able to create macros that can be executed in long chains without problems.

In general, good macros have three parts:

- They find the place you want the macro to start working (often using search).
- They do the work that needs to be done on the text.
- They prepare themselves to repeat.

How can a macro prepare itself to repeat? For example, assume that you're writing a macro to delete the third column of a table. After deleting the column, the macro should position itself at the beginning of the next line (or wherever it needs to be) so you don't have to reposition the cursor before reusing it.

Here's a slightly more complex example. If you start a macro with a search, you have to make sure that the end of the macro moves the cursor past the last spot you searched for. If you don't, the macro will keep finding the same place in the file and never go on to the next occurrence of what you're searching for. As a general rule, if your macro operates on a line of text, it should end by moving to the beginning of the next line. Remember that your goal is to create a sequence of keystrokes that can be executed many times in a row, with no interruption.

A More Complicated Macro Example

Sometimes you may want to find all the references to a particular topic in a file. Table 6-2 lists steps for creating a macro that takes every sentence in the buffer that contains the word *Emacs* and copies it to another buffer. If you try this macro, you'll need to type some text about Emacs into a buffer. You can also get a test file to work with by opening the Emacs *NEWS* file (using **C-h n**), then writing it to a file (**C-x C-w NEWS**). This buffer is in view mode by default; change to text mode by typing **M-x text-mode Enter**.

Table 6-2. Steps for macro that creates a buffer of Emacs references

| Keystrokes | Action |
|---------------------------|---|
| F3 or C-x (| Start macro definition; Def appears on the mode line. |
| C-s emacs | Find the word Emacs. |
| Enter | Stop the search after it is successful; if the search is unsuccessful, it rings the bell and stops the macro. |
| M-a | Move to the beginning of the sentence. ^a |
| C-Space | Set the mark. |
| M-e | Move to the end of the sentence. |
| M-w | Copy the sentence to the kill ring. |

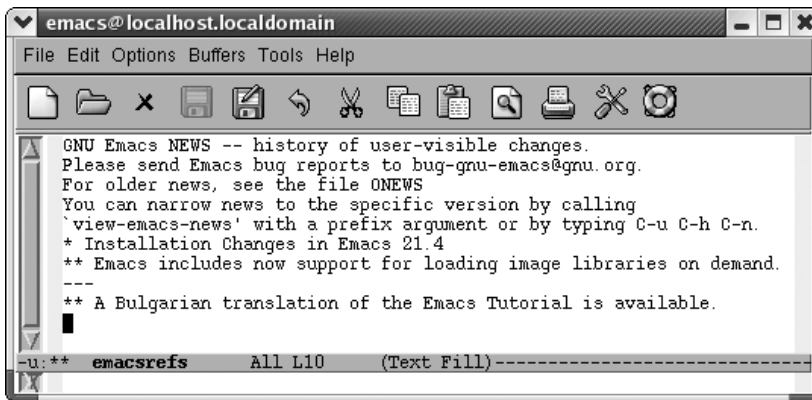
Table 6-2. Steps for macro that creates a buffer of Emacs references (continued)

| Keystrokes | Action |
|------------------------------|--|
| C-x b emacsrefs Enter | Move to a buffer called emacsrefs. |
| C-y | Insert the sentence. |
| Enter | Start the next sentence on a new line. |
| C-x b Enter | Move back to the original buffer. |
| F4 or C-x) | End the macro definition; Def is removed from the mode line. |

^a M-a's definition of a "sentence" is controlled by the variable `sentence-end`, which is a fairly complex regular expression. By default, a sentence ends with a period, question mark, or exclamation mark, optionally followed by a quotation mark or parenthesis (including brackets or braces), and followed by two or more spaces or a newline.

Now, assume that you've already constructed the macro outlined in Table 6-2 and that you can invoke it with F4. The following screen shows what happens when you run it five times and then display the `emacsrefs` buffer.

Type: **M-5 F4** or **M-5 C-x e**, followed by **C-x b Enter**



By executing the macro repeatedly, we've created a buffer that contains references to the Emacs editor.

As in the previous example, you can jump back and forth between an unlimited number of buffers while defining a macro. Macros don't need to be confined to one buffer. Macros that work with several buffers are more difficult to debug; when several buffers are involved, it becomes harder for you to keep track of where the cursor and the mark are. It is also easy to make mistaken assumptions about what buffer you're visiting; hence, it's a good idea to specify the buffer name explicitly. However, after you get accustomed to working with macros and multiple buffers, you'll be amazed at how much work you can do with almost no effort.

Windows are sometimes useful in macros, but, again, you have to watch out. It's better to start a macro with one window on the screen, have the macro open other windows, and finally close all but one window (**C-x 1**). If you write a macro with two windows on the screen and later try to execute it with four windows on the screen, the results will be unpredictable at best! In general, moving to a named buffer, **C-x b *buffername***, is preferable to moving to the "other" window using **C-x o** (too vague to be generally useful). The other window could be anything—a **Help** buffer, **Completion** buffer, **shell** buffer, and so on. Moving to a named buffer always gets you to the right place, no matter how (or whether) the buffer is displayed.

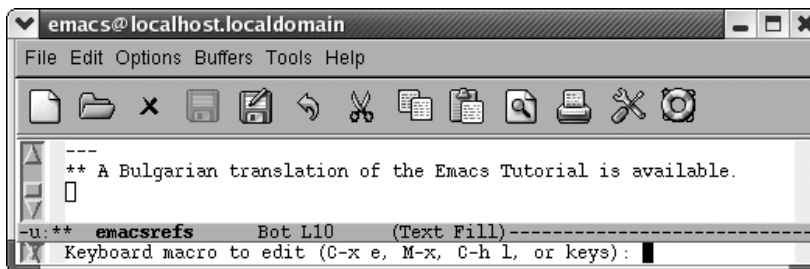
Editing a Macro

You can edit a macro and make changes to it in a few different ways. For this example, we chose an all-purpose editing command, **edit-kbd-macro**, which is bound to **C-x C-k e**. Several macro editing commands are available, but this one works for all types of macros, so it's good to learn.

Our macro could use a bit of tweaking. First of all, finding references to Emacs in our copy of the Emacs *NEWS* file is pretty lame. Perhaps we're interested in using a mouse more frequently with Emacs and would like to know about changes to that part of the interface. We'll edit the macro to search for the word *mouse*. We'll also modify it so it marks a paragraph rather than a sentence since a sentence doesn't really provide enough context to be helpful.

Let's start editing the macro.

Type: **C-x C-k e**

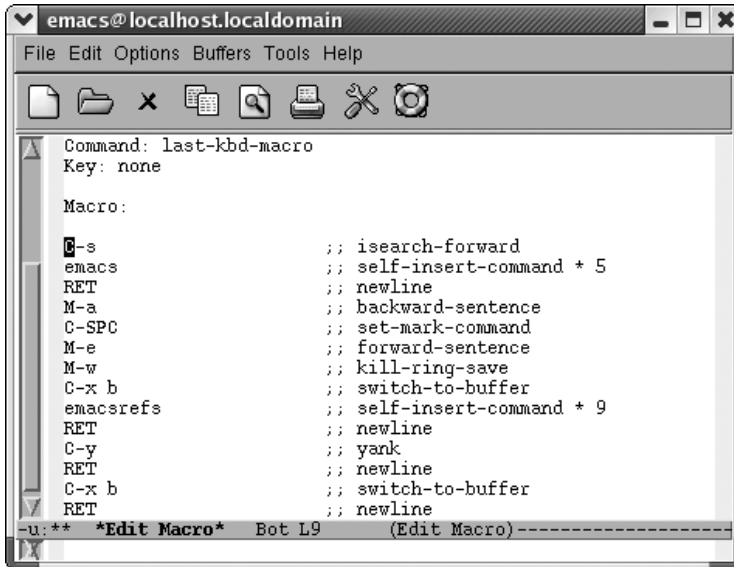


Emacs prompts you for the type of macro to edit.

Emacs asks you if you want to edit the last keyboard macro (**C-x e**), a named macro (**M-x**), the last 100 keystrokes as a macro, termed "lossage" (**C-h l**), or keys (meaning the keystrokes you bound a macro to). Yes, that's a lot of choices, and later in the chapter we describe named macros and binding macros to keys (you can experiment

on your own with creating a macro from lossage). For now, just choose **C-x e** to edit the last keyboard macro.

Type: **C-x e**



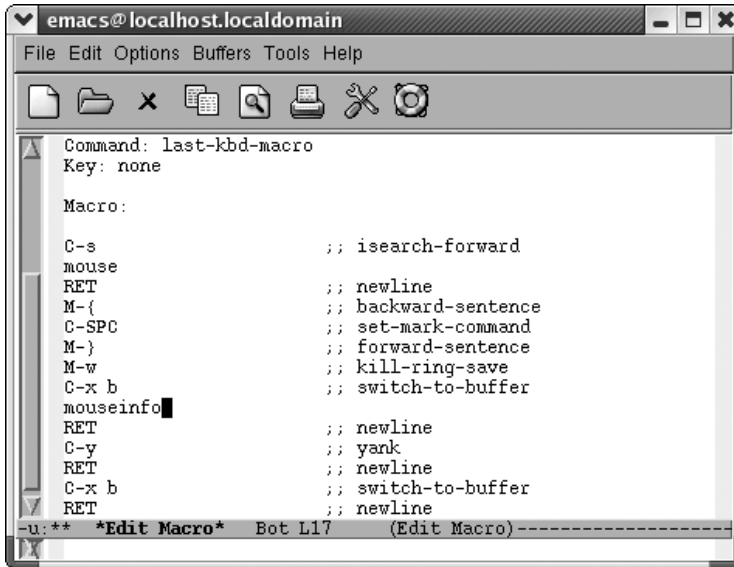
Emacs opens an **Edit Macro** buffer.

Notice two fields near the top of this buffer, *Command:* and *Key:*. Right now, *Command:* says *last-kbd-macro*. If this were a named macro, the command would be the name you gave your macro. Additionally, for frequent use, you can bind your macro to a key, at which point the *Key:* field lists the keystrokes to execute this macro. Right now it says *none* because we haven't defined any keystrokes yet.

Note that Emacs inserts comments all through the macro. It's attempting to map keystrokes to commands. You do not need to update these comments or add comments if you add commands to your macro; Emacs does that itself.

To tweak our macro, we change the search string on the second line from *emacs* to *mouse*. Note that we can just press **C-k** to wipe out the line and type *mouse*. Now change **M-a** to **M-{** and **M-e** to **M-}**. We change the buffer name from *emacsrefs* to *mouseinfo*.

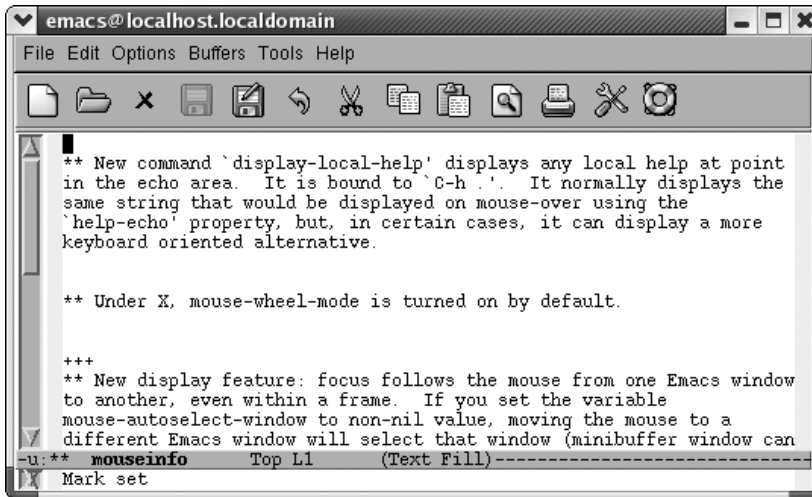
We've made the edits from the previous paragraph. The screen looks like this:



A modified macro that captures information about using a mouse in Emacs.

To exit the macro editing buffer, we have to type `C-c C-c` and go back to our NEWS buffer. Let's do that and then execute the macro again to see what happens.

Type: `C-c C-c C-x b` Enter `M-< M-5 F4 C-x b` Enter `M-<`



The mouseinfo buffer shows paragraphs from our copied NEWS file that mention the mouse.

The Macro Ring

Although our latest macro is interesting, it's not really a general purpose macro. It is a temporary solution to a one-time problem. It saves you some work, but it isn't general enough to save and use again. On the other hand, our macro to transpose names is generally useful. We'd like to use it again. We'd like to bind it to a key. But it is no longer the "latest" keyboard macro.

As we mentioned earlier, Emacs has a macro ring much like the infamous kill ring. It's useful in the case we've just described, but it's also useful because of the fragility of the macro definition process. You create a macro and make a wrong move that rings the bell, and your macro is canceled. It's fairly easy to create a macro that does nothing. Perhaps the macro that you just created was wonderful, and this new non-functional nothing macro has supplanted it. Again, the macro ring is the solution. To delete a macro from the ring, type **C-x C-k C-d** (for **kmacro-delete-ring-head**). This deletes the most recently defined keyboard macro.

What if you want to swap the positions of two macros? Instead, type **C-x C-k C-t** (for **kmacro-swap-ring**). This transposes macros 1 and 2.

In a more general sense, you can cycle to the previously defined macro by typing **C-c C-k C-p** (for **kmacro-cycle-ring-previous**). To move the ring the other way, type **C-x C-k C-n** (for **kmacro-cycle-ring-next**). The familiar **C-p** for previous and **C-n** for next bindings are appended to the general macro keyboard prefix **C-x C-k**.

Before we can work with the transpose names macro, we must either define it again or, if you've been working through our examples, type **C-x C-k C-p** to move to the previous macro.

Binding Your Macro to a Key

Binding a macro to a key is easy. The key sequences **C-x C-k 0** through **9** and capital **A** through **Z** are reserved for user macro bindings. You can choose one that strikes you as mnemonic for your macro.

For example, to bind our transpose names macro to **C-x C-k T**, type **C-x C-k b**. Emacs prompts for the key binding. Type **C-x C-k T Enter**. Emacs confirms, Keyboard macro bound to C-x C-k T. Binding a macro command to a key in this way works for only one session. We want to keep this macro, so read on to find out how to make this binding permanent.

Naming, Saving, and Executing Your Macros

In this section, we'll describe how to save macros so that you can use them in different editing sessions. To save a macro, bind it permanently to a key, and load it in subsequent Emacs sessions, follow these steps:

1. Define the macro, if you haven't already.
2. Type **C-x C-k n** (for **name-last-kbd-macro**). Now type a name for your macro and press **Enter**. A non-Emacs sounding name is best so that Emacs doesn't confuse it with one of its own commands. Once you've executed this command, Emacs remembers the macro for the rest of the editing session. To use it again, type the command **M-x name** (where *name* is the name you've chosen). Emacs treats your named macro like one of its own commands; it shows up in completion lists if you press **Tab** after typing a few letters of the name.
3. If you want to save the macro definition permanently, you must insert the macro definition into a file. This could be your *.emacs* file or a macro file that you load through your *.emacs* file. Type **C-x C-f filename Enter** to find the file into which to insert the definition and move to the end of it by typing **M->**.
4. Type **M-x insert-kbd-macro Enter macroname Enter**. Emacs inserts Lisp code that represents your macro.
5. Add a line to *.emacs* make the key binding permanent. For example, if we called our macro **transpose-names** and bound it to **C-x C-k T**, we would add this line to our *.emacs* file (or other macro definition file):

```
(global-set-key "\C-x\C-k T" 'transpose-names)
```
6. If you save the macro in some other file, it won't be loaded automatically. For example, let's say that you have defined a macro called **transpose-names** and placed it in the file *html.macs*, in the directory *~/macros*. Add this line to your *.emacs* file to load your macros automatically:

```
(load-file "~/macros/html.macs")
```
7. Save the *.emacs* file and, if different, the file in which you inserted your macro. Exit and restart Emacs. You can now execute this macro either by typing **M-x transpose-names Enter** or by pressing **C-x C-k T**.

Building More Complicated Macros

So far, we've covered the basics of writing, executing, and saving keyboard macros. Now let's discuss a couple of more advanced features Emacs lets you add to your macros: pausing a macro for keyboard input and inserting a query in a macro.

Pausing a Macro for Keyboard Input

Sometimes it's useful to pause a macro briefly so you can type something. For example, if you write a lot of letters, you could have a macro that prints out a template and then pauses for you to fill in variables (such as the date and the recipient's name). You can perform this task (and similar tasks) by inserting a recursive edit into a macro. A recursive edit is just a fancy way to say, "Stop and let me type a while, then pick up the macro where I left off."

When you're defining a macro, type **C-u C-x q** at the point where you want the recursive edit to occur. Emacs enters a recursive edit. (You can tell you're in a recursive edit because square brackets appear on the mode line; you'll see them in the screenshots later in this section.) Nothing you type during the recursive edit becomes a part of the macro. You can type whatever you want to and then press **C-M-c** to exit the recursive edit. Notice how the square brackets disappear when you type **C-M-c**. When the square brackets are no longer on the screen, you have left the recursive edit. Anything you type at this point becomes part of the macro. You can put as many pauses in your macros as you want to.

Example

Here's an example of a macro that puts a business letter template on the screen and uses recursive edits to let you type your return address, the recipient's name and address, and the date. Because the brackets on the mode line are a pretty subtle clue to what you are going to type, we'll give the user of this macro explicit instructions about what to type. Table 6-3 provides these instructions.

Table 6-3. Steps for creating a business letter macro

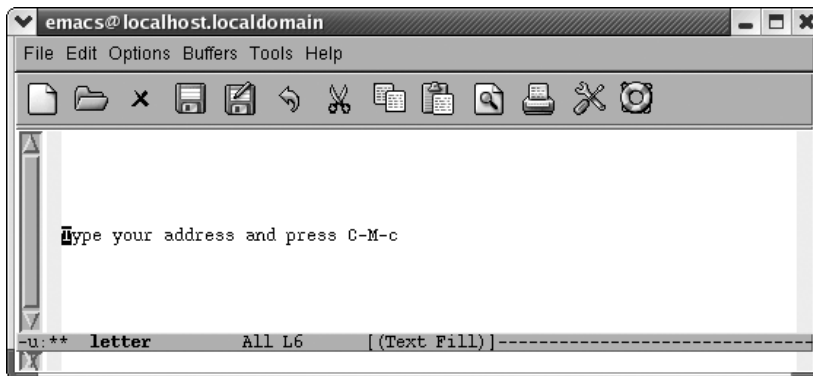
| Keystrokes | Action |
|--|---|
| F3 or C-x (| Start keyboard macro definition. |
| M-5 Enter | Put in 5 blank lines. |
| Type your address and press C-M-c | Display Type your address and press C-M-c on the screen. |
| C-a | Move to the beginning of the line. |
| C-u C-x q | Enter a recursive edit, during which the keystrokes you type are not recorded as part of the macro. |
| C-M-c | Exit the recursive edit. |
| C-e | Move to the end of the line. |
| M-5 Enter | Move the cursor down 5 lines. |
| Type recipient name and address and press C-M-c | Display Type recipient name and address and press C-M-c on the screen. |
| C-a | Move to the beginning of the line. |
| C-u C-x q | Enter a recursive edit. |
| C-M-c | Exit the recursive edit. |

Table 6-3. Steps for creating a business letter macro (continued)

| Keystrokes | Action |
|---------------------------|--|
| C-e | Move to the end of the line. |
| M-5 Enter | Move the cursor down 5 lines. |
| Type date and press C-M-c | Display Type date and press C-M-c on the screen. |
| C-a | Move to the beginning of the line |
| C-u C-x q | Enter a recursive edit. |
| C-M-c | Exit the recursive edit. |
| C-e | Move to the end of the line. |
| M-5 Enter | Move the cursor down 5 lines. |
| Dear Space | Display Dear on the screen. |
| F4 or C-x) | End keyboard macro definition. |

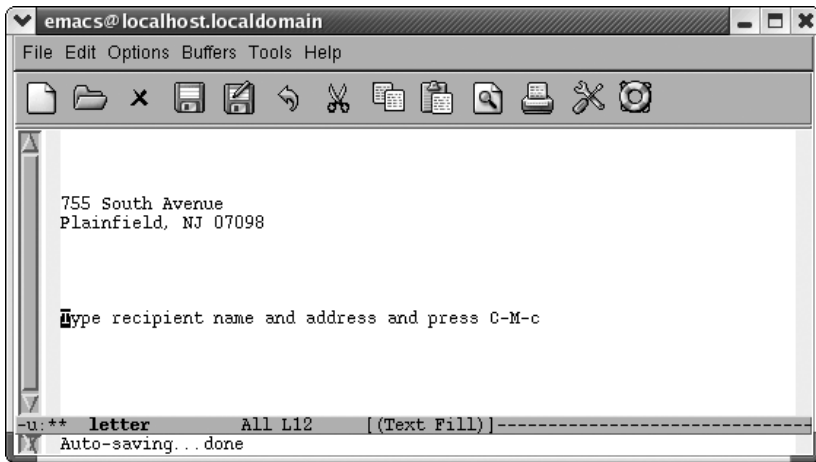
The following screens show what the macro defined in Table 6-3 looks like when you run it.

Type: F4



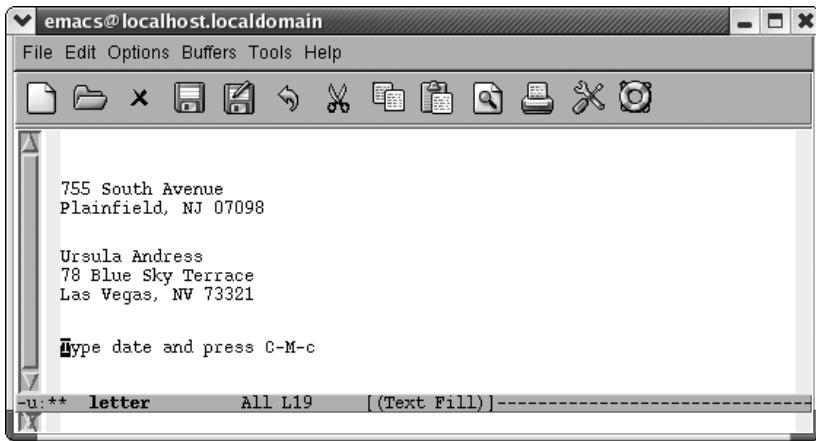
The macro pauses so that you can type your address.

Type your address and press: **C-M-c**



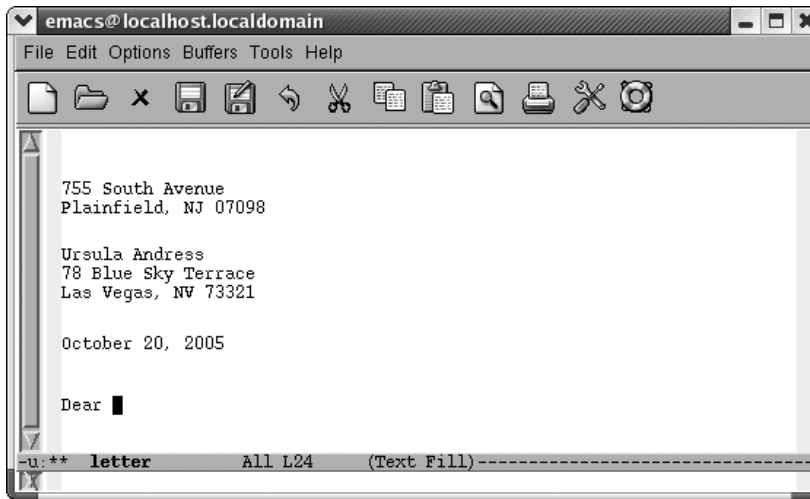
The macro pauses so you can type the recipient's name and address.

Type the recipient's name and address and press: **C-M-c**



The macro pauses so you can type the date.

Type the date and press: **C-M-c**



The macro finishes by typing the opening for the letter.

Now the macro has finished editing; you can type the recipient's name and then the body of the letter, and of course you can go back and edit any of the information you've already filled in.

Adding a Query to a Macro

The more complex the task your macro performs, the more difficult it is to make the macro general enough to work in every case. Although macros can do a lot of things, they aren't programs: you can't have **if** statements, loops, and the other things you associate with a program. In particular, a macro can't get input from the user and then take some action on the basis of that input.

However, one feature lets a macro get input, in a limited way, from the user. You can create a macro that queries the user while it is running; it works much like a query-replace. To create this kind of a macro, type **C-x q** when you reach the point in the macro definition where you want the macro to query the user. Nothing happens immediately; go on defining the macro as you normally would.

Things get interesting later, when you execute the macro. When it gets to the point in the macro where you typed **C-x q**, Emacs prints a query in the minibuffer:

```
Proceed with macro? (y, n, RET, C-l, C-r)
```

The responses listed here are analogous to those in query-replace:

- Pressing **y** means to continue and go on to the next repetition, if any.
- Pressing **n** means to stop executing the macro but go on to the next repetition, if any.
- Pressing **Enter** means to stop executing the macro and cancel any repetitions.
- Pressing **C-r** starts a recursive edit, which lets you do any editing or moving around you may want to and then resume the macro when you exit the recursive edit. To exit a recursive edit, press **C-M-c**. Emacs again asks if you want to proceed with the macro, and you type **y** for yes or **n** or **Enter** for no.
- Pressing **C-I** puts the line the cursor is on in the middle of the screen (this is good for getting a feel for the context). Similar to **C-r**, Emacs again asks if you want to proceed with the macro, and you have to answer **y**, **n**, or **Enter**.
- Pressing **C-g** (although not listed as an option) cancels the query and the macro; it is similar to pressing **Enter**.

Example

Let's say that you write a macro that copies comments from a program to another buffer. The comments in our program are preceded by a slash, so you start the macro with a search for a slash. However, not all comments are worth copying. Following the search with a query lets you decide case by case whether the search has found a comment you want to copy. Table 6-4 shows a macro to copy comments to another buffer.

Table 6-4. *Comment-copying macro with a query*

| Keystrokes | Action |
|----------------|--|
| F3 | Start the macro definition. |
| C-s / | Search for a slash. |
| Enter | Stop the search when it is successful. |
| C-x q | Insert a query in the macro; Emacs asks you if you want to proceed at this point when you run the macro. |
| M-f | Move forward one word. |
| M-b | Move to the beginning of this word. |
| C-Space | Set the mark. |
| C-e | Move to the end of the line. |
| C-f | Move forward one character. |
| M-w | Copy the comment to the kill ring. |

Table 6-4. Comment-copying macro with a query (continued)

| Keystrokes | Action |
|-----------------------|---|
| C-x b comments | Move to a buffer called <code>comments</code> . |
| C-y | Insert the comment in the buffer. |
| C-x b | Move back to the original buffer. |
| F4 | End the macro definition. |

Executing Macros on a Region

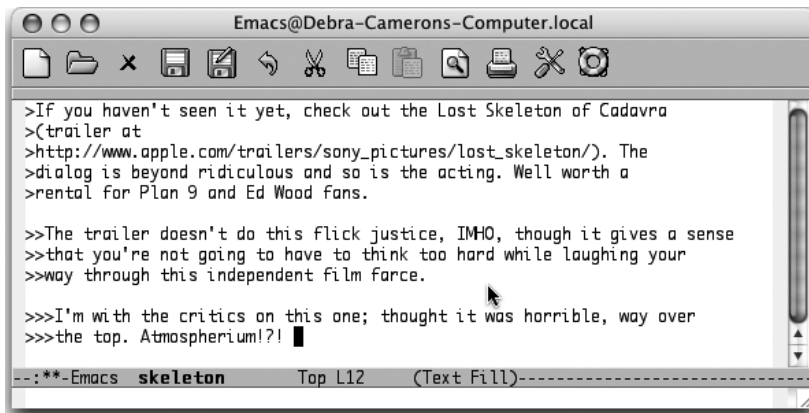
A special command lets you execute a macro on each line in a region. How frequently do you encounter an email with text that you want to yank, but that is quoted several indentation levels? Of course, we can think of several ways to delete the indentation quickly, but a line-oriented macro is a quick approach too. You define the macro and execute it on a region by typing **C-x C-k r** (for **apply-macro-to-region-lines**). Remember that earlier we said that macros should set themselves up to repeat? This command is different because it expects to work on one line at a time. You don't want to set it up to repeat by moving to the next line; it does that automatically.

Table 6-5 shows a quick line-oriented macro that deletes indentation marks from text quoted in an email or newsgroup message.

Table 6-5. Macro for deleting indentation marks

| Keystrokes | Action |
|----------------|---|
| F3 | Start the macro definition. |
| C-a | Move to the beginning of the line. |
| M-f | Move forward one word. |
| M-b | Move to the beginning of this word. |
| C-Space | Set the mark. |
| C-a | Move to the beginning of the line. |
| C-w | Delete the extraneous indentation characters. |
| F4 | End the macro definition. |

Initial state:



The screenshot shows an Emacs window titled "Emacs@Debra-Camerons-Computer.local". The window contains the following text with varying levels of indentation:

```
>If you haven't seen it yet, check out the Last Skeleton of Cadavra
>(trailer at
>http://www.apple.com/trailers/sony_pictures/lost_skeleton/). The
>dialog is beyond ridiculous and so is the acting. Well worth a
>rental for Plan 9 and Ed Wood fans.

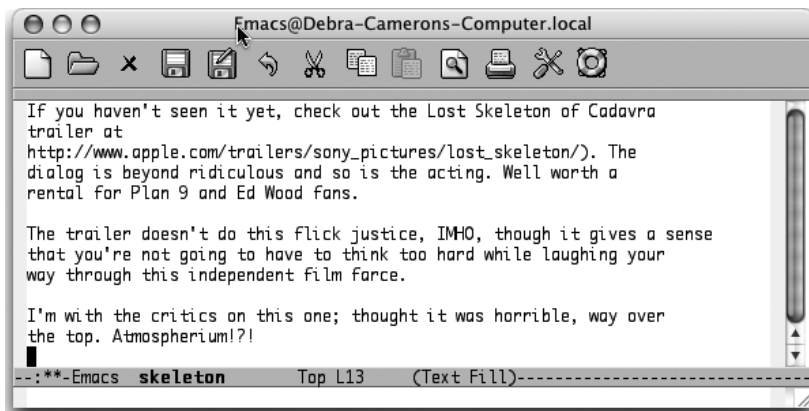
>>The trailer doesn't do this flick justice, IMHO, though it gives a sense
>>that you're not going to have to think too hard while laughing your
>>way through this independent film farce.

>>>I'm with the critics on this one; thought it was horrible, way over
>>>the top. Atmospherium!?! █
```

The status bar at the bottom shows "--:**-Emacs skeleton Top L12 (Text Fill)-----".

Text indented at various levels (Mac OS X).

Mark the text as a region, move to the beginning of the region, then type: **C-x C-k r**



The screenshot shows the same Emacs window after the command **C-x C-k r** has been executed. The text is now left-aligned, and the indentation has been removed:

```
If you haven't seen it yet, check out the Last Skeleton of Cadavra
trailer at
http://www.apple.com/trailers/sony_pictures/lost_skeleton/). The
dialog is beyond ridiculous and so is the acting. Well worth a
rental for Plan 9 and Ed Wood fans.

The trailer doesn't do this flick justice, IMHO, though it gives a sense
that you're not going to have to think too hard while laughing your
way through this independent film farce.

I'm with the critics on this one; thought it was horrible, way over
the top. Atmospherium!?! █
```

The status bar at the bottom shows "--:**-Emacs skeleton Top L13 (Text Fill)-----".

Indentation is deleted (Mac OS X).

Beyond Macros

Macros are an important tool for streamlining repetitive editing. They let you write your own commands for performing complex tasks without needing to know anything more than you already know: the basic Emacs commands for moving around and manipulating text. Even if you're an Emacs novice, you should be able to use macros with little difficulty.

However, Emacs is almost infinitely flexible, and macros cannot do everything. In many situations, there's no substitute for writing a Lisp function that does exactly what you want. If you know Lisp or would like to learn some, you can write your own Lisp functions to do more complex tasks than keyboard macros can handle. Chapter 11 covers the basics of writing Lisp functions.

Table 6-6 summarizes macro commands.

Table 6-6. Macro commands

| Keystrokes | Command name | Action |
|----------------------|---|---|
| C-x (| kmacro-start-macro | Start macro definition. |
| F3 | kmacro-start-macro-or-insert-counter | Start macro definition. If pressed while defining a macro, insert a counter. |
| C-x) | kmacro-end-macro | End macro definition. |
| F4 | kmacro-end-or-call-macro | End macro definition (if definition is in progress) or invoke last keyboard macro. |
| C-x e | kmacro-end-and-call-macro | Execute last keyboard macro defined. Can type e to repeat macro. |
| C-x C-k n | name-last-kbd-macro | Name the last macro you created (before saving it). |
| <i>(none)</i> | insert-kbd-macro | Insert the macro you named into a file. |
| <i>(none)</i> | <i>macroname</i> | Execute a named keyboard macro. |
| C-x q | kbd-macro-query | Insert a query in a macro definition. |
| C-u C-x q | <i>(none)</i> | Insert a recursive edit in a macro definition. |
| C-M-c | exit-recursive-edit | Exit a recursive edit. |
| C-x C-k b | kmacro-bind-to-key | Bind a macro to a key (C-x C-k 0-9 and A-Z are reserved for macro bindings). Lasts for current session only. |
| C-x C-k Space | kmacro-step-edit-macro | Edit a macro while stepping through it (in our opinion, the interface is overly complex). |
| C-x C-k l | kmacro-edit-lossage | Turn the last 100 keystrokes into a keyboard macro. If any mouse clicks are among the last 100 keystrokes, does not work. |
| C-x C-k e | edit-kbd-macro | Edit a keyboard macro by typing C-x e for the last keyboard macro defined, M-x for a named macro, C-h l for lossage, or keystrokes for a macro bound to a key. |
| C-x C-k Enter | kmacro-edit-macro | Edit the last keyboard macro. |
| C-x C-k C-e | kmacro-edit-macro-repeat | Edit the last keyboard macro again. |
| C-x C-k C-t | kmacro-swap-ring | Transpose last keyboard macro with previous keyboard macro. |
| C-x C-k C-d | kmacro-delete-ring-head | Delete last keyboard macro from the macro ring. |
| C-x C-k C-p | kmacro-cycle-ring-previous | Move to the previous macro in the macro ring. |
| C-x C-k C-n | kmacro-cycle-ring-next | Move to the next macro in the macro ring. |
| C-x C-k r | apply-macro-to-region-lines | Apply this macro to each line in a region. |