

Living in Emacs

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. About this tutorial	2
2. Origins	3
3. Getting started with Emacs	5
4. Common text operations	9
5. Cursor navigation in Emacs	12
6. Search and replace	15
7. Buffers and files	18
8. A glimpse of the depths	22
9. Summary, resources, and feedback	25

Section 1. About this tutorial

What does this tutorial cover?

This tutorial gives you a guide to the basics of using Emacs, a popular modeless text editor with many powerful features. The tutorial covers fundamental concepts and common activities, and then builds on those foundations to quickly familiarize you with this excellent editor.

Getting started with Emacs requires navigating a steep learning curve. Our goal is to help you past the initially unfamiliar interface so that the power and utility of Emacs become apparent. Then you'll be ready to explore further on your own, following up on the resources and tips at the end of the tutorial.

Who should take this tutorial?

The primary users of Emacs are programmers and Web developers who want to get the most out of this powerful and flexible text editor and thereby increase their productivity. Additionally, at least a passing familiarity with Emacs is useful for anyone who performs administrative duties in UNIX or similar environments.

Before you begin

All you need to work your way through this tutorial is a copy of Emacs, either [GNU Emacs](#) or [XEmacs](#).

If you're running Linux, then you might already have it loaded. Check by typing `emacs` at a command-line prompt. If nothing happens (or you get a message like "command not found"), then use the package tools that come with your distribution to install one package or the other.

Running another operating system? Check the sites linked above for a version of Emacs that will work for you.

About the author

Brian Bilbrey is a system administrator, Webmaster, product and PWB designer, author, and Linux advocate. His business card has been known to read NPS, standing for No Particular Specialty. Brian uses Linux in his daily work, and changes his window manager and favorite applications the way some people change clothing. New distributions are like bread and water -- fundamental -- and Brian is currently happily experimenting with Gentoo Linux. His daily blog on life with Linux and other adventures can be found at [OrbDesigns.com](#).

For technical questions about this tutorial, please contact Brian at bilbrey@orbdesigns.com.

Section 2. Origins

Overview

In this tutorial, we'll cover a lot of ground very quickly. First we'll have a look at what Emacs is and where it comes from. Then we'll jump right into using the editor, starting with keystrokes, commands, the Emacs environment, and some of the elemental commands you need to get started. I'll show you how to add and delete, kill and yank text in a variety of ways.

The next leg of our journey is an introduction to Emacs' cursor navigation scheme. That's followed by an examination of the search and replace features. After that, I'll show you what Emacs does with files and buffers. I'll wrap the trip up with a few glimpses at the higher functions and extra features that you can only find in Emacs, from modes to coding to connectivity to games.

At the end of this tutorial, you will be comfortable moving around in the Emacs environment and have a sense of the power that's available to you through it. Let's get started.

What is Emacs?

According to a description at GNU.org, *Emacs is the extensible, customizable, self-documenting real-time display editor. It offers true LISP -- smoothly integrated into the editor -- for writing extensions and provides an interface to the X Window System.*

It has also been said (perhaps not entirely in jest) that Emacs can do so very many different things so well that it would make a fine operating system indeed -- if only it had a decent text editor.

But seriously: Emacs is a robust and extensible text-editing environment that has many, many additions designed into it, from compiling and debugging interfaces to e-mail, games, and Eliza. Especially for those who write or code (or both) for a living, it's easy to start up several Emacs sessions in the morning, start working, and never execute another application all day, thus the name of this tutorial: *Living in Emacs*.

Origins and alternatives

The original Emacs was written by Richard Stallman for the Incompatible Timesharing System (ITS) at the Massachusetts Institute for Technology in the 1970s. GNU Emacs, first released in 1984, is also the brainchild of the talented Richard Stallman, is available from GNU.org, and is licensed under the Free Software Foundation's GNU GPL (see [Resources](#) on page 25 for a link).

There is one major "competitor" to GNU Emacs -- XEmacs -- which is the result of a fork in the Emacs codebase. This fork took place far enough back that, while major portions of the user interface are identical or highly similar, the underlying extensions and LISP code are not compatible. Porting between the two is possible however.

Many Linux distributions are accompanied by both versions of Emacs, although preferentially one is installed over the other, depending upon the choices made by the publisher. Debian, for instance, installs GNU Emacs if you choose to install Emacs, as does Red Hat 7.2. The last time I installed Caldera OpenLinux, it defaulted to XEmacs.

For the purposes of this tutorial, our descriptions, examples, and screenshots are based upon GNU Emacs. Point your Web browser to <http://www.gnu.org/software/emacs/emacs.html> for more details.

Section 3. Getting started with Emacs

Emacs keystroke conventions

Native Emacs documentation has a unique way of describing the keystrokes that are used to define actions. These are as follows:

`C-<chr>` == Control + character, pressed at the same time.

`M-<chr>` == Meta + character, pressed at the same time.

But what's Meta? Meta can be a dedicated key (sometimes so labeled), it might be the Alt key, or perhaps it doesn't even exist in the keymap that your system uses. That's okay; there is a fallback to Meta, which is to first press the Esc key and then the following character in turn (instead of together). This yields the same result as `M-<chr>`.

Now start up your copy of Emacs (or XEmacs), and let's make some quick progress. Type `emacs practice1.text` in a terminal or console to get started.

Commands and key-bindings

Emacs implements a version of LISP, a threaded language, to build its commands and extensions. All commands have names, like `Buffer-menu-bury`, `backward-char`, and `forward-paragraph`. And while they're logically arranged and named, there are over 1800 of them in my current installation, and that's one heck of a lot of typing.

That's why many of the commands are bound to key combinations, prefaced with the Control and Meta keys. To invoke a named command, start by typing `M-x` followed by the command name. To get a list of the key bindings, the long form command is `M-x describe-bindings`. Fortunately, there's a keybinding for that: `C-h b`

Type `C-x o` to swap to the listing window, `C-s` to do an incremental search, `C-x o` to switch back to your working window, and `C-x 1` to close all windows except for the current buffer. Give that a try, and have a look at some of those commands -- there are about 600 or so that have key-bindings. Also, don't worry about the commands we used in this quick side trip, as we'll revisit all of them in turn later in the tutorial.

First instructions

Quitting: When I first started using Emacs, I found that I would get lost someplace in the documentation, or in a welter of buffers that I was sure I hadn't opened myself, and so on. At that point, all I wanted to do was exit the system so that I could start over again and figure out where I went wrong. Here's the sequence you type to exit Emacs: `C-x C-c`.

From the keystroke convention that you saw in the previous panel, that means to press `Ctrl`

+ x, followed by `Ctrl + c`. If you made any changes in any open files, then Emacs will prompt you, for example:

```
Save file /home/bilbrey/practicel.txt? (y, n, !, ., q,  
C-r or C-h)
```

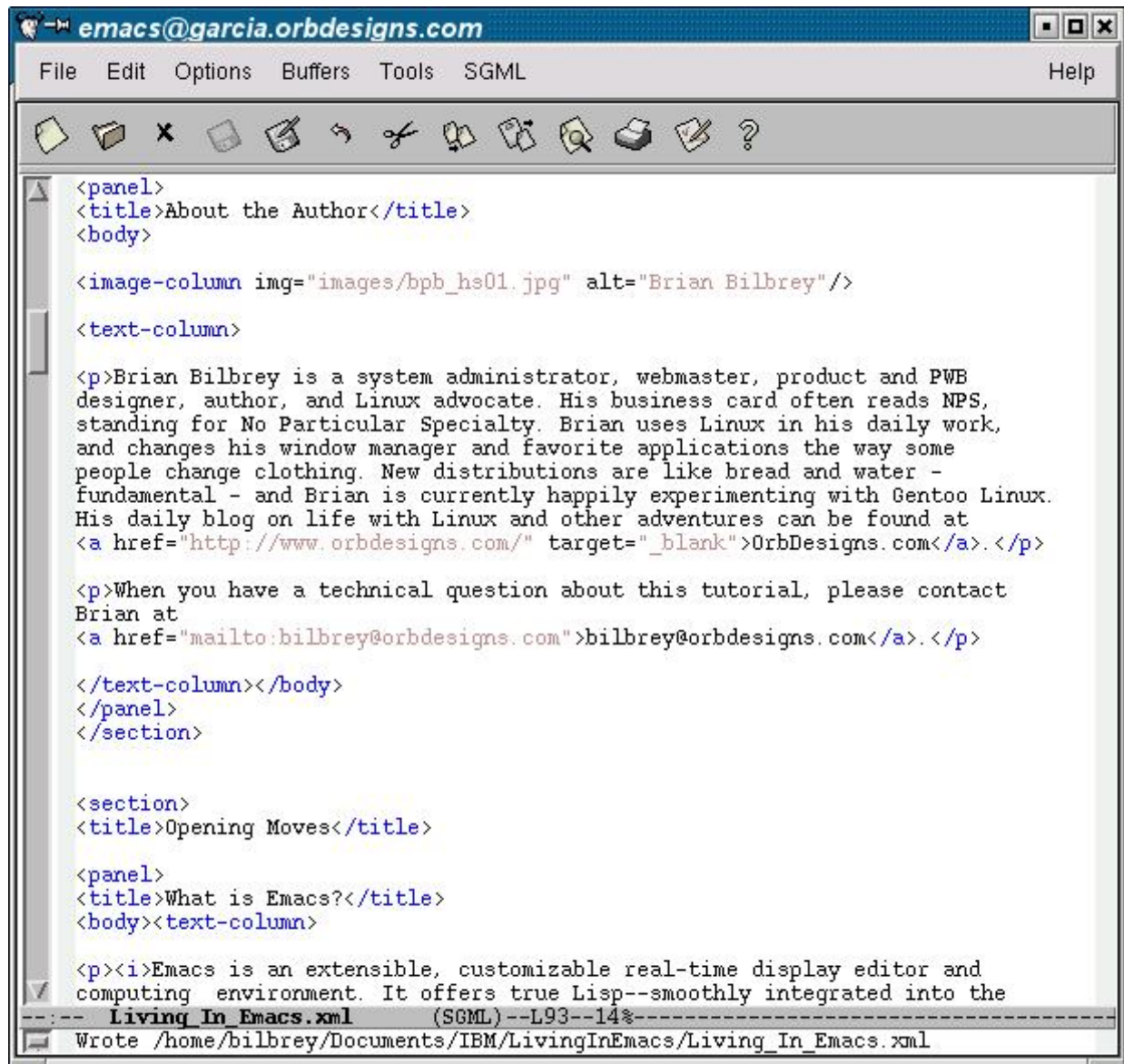
I'll reply `y` to any such prompts if I've made changes I care about, or press the `!` to simply proceed with quitting, nothing saved.

To open an existing file after Emacs is started, type `C-x C-f` to find a file and load it into a buffer.

On the other hand, I most often want to save the work I've done and then continue typing. So to save my work and continue, the keystroke combination is `C-x C-s`.

The Emacs view, part 1

There are three major sections to any Emacs or XEmacs screen: buffer(s), the status bar, and the mini-buffer at the bottom. This tutorial, in its XML formatted version, appears in the image.



The Emacs view, part 2

The screenshot in the previous panel is from the X-enabled version of GNU Emacs. The bits in that view that aren't relevant for a text-mode only version (as in a console or terminal window) are the upper GUI button menu and the mouse-enabled scroll bar (in most cases).

The main editing window can be split into two or more windows, which can be views of the same buffer (file), or of different buffers. See the [Windows in Emacs](#) on page 19 panel for more details.

In the initial configuration, the editing window has a demarcation at the bottom by a status bar

(also known as the mode bar). With multiple visible windows, each will have its own status bar. The status bar has indicators for whether the text in the buffer has changed, the file name associated with the name, the mode (shown as SGML in the screenshot), the current line number, and the position of the cursor as a percentage of the entire text. The mode indicates what type of text Emacs thinks it is working with and modifies the menus and functions accordingly.

The bottom line, which contains a *[Wrote...]* message in the screenshot, is called the mini-buffer. It's used to display partially-typed commands, the results of commands run, and occasionally to show minimal help.

Section 4. Common text operations

Inserting text

Emacs is very easy in one important sense. No need to get into an insert mode or exit from any special command mode -- just type and you're inserting text. Let's repeat one thing here: save your work, early and often, with the Save Buffer command, `C-x C-s`.

Did you enjoy that? This is the shortest, easiest panel in this tutorial. Now take a deep breath, and let's dive in to deleting text.

Basic deleting and undo

There are two different ways to delete text. In this panel we'll address the first: Character deletion. Single characters are deleted in the manner to which you are likely already accustomed: by using the Delete key or the Backspace key.

Delete, at least, has an Emacs equivalent: `C-d` deletes the character under the cursor. To undo character deletion, use the `C-x u` command or the real shorthand, `C-_`. The latter is easier for multiple undos. Practice these operations just a bit now to start training your fingers in Emacs.

Note: Some of the documentation I have read indicates that the Delete key should delete backwards (the backspace or `^H` equivalent) and `C-d` takes the place of Delete. This depends on your operating setup and terminal configuration.

Deleted characters are only saved in a buffer for undo, and you can only reach those modifications by undoing all that's changed since the deletion. The more "advanced" form of deletion, for multi-character regions, is saved to a different structure as well, and we'll look at that next.

Emacs cut and paste, part 1

Here are the commands you need for deleting larger blocks (it's called "killing"):

Key-binding	Action (command)
<code>M-d</code>	<code>kill-word</code>
<code>M-Delete</code>	<code>backward-kill-word</code>
<code>M-k</code>	<code>kill-sentence</code>
<code>C-x Delete</code>	<code>backward-kill-sentence</code>
<code>C-k</code>	<code>kill-line</code>

`C-k` has a bit of a trick to it. Used once, it kills the text on the line but not the newline character. That takes a second `C-k`. There are also commands to kill paragraphs, `kill-paragraph` and `backward-kill-paragraph`, although key bindings don't exist for those.

So where does your deleted stuff go? Into the *kill ring*, of course. Multiple sequential deletes (for instance, repeating `C-k` several times) goes into the kill ring as a block, which is very handy. In the next panel, we'll look at accessing that data.

Emacs cut and paste, part 2

The kill ring is so called because it stores deleted text larger than a single character. Also, it can be accessed sequentially, from the latest back to the first item deleted during the editing session, and then it wraps back to the most recent again. Thus, it is a ring, topologically.

Type `C-y` to *yank* the most recent block. Repeating `C-y` merely yanks that block again.

To get at the older "killed" items, type `C-y` first, and you'll see the most recent block. Then, type `M-y` to step back through the kill ring. Each step replaces the prior yank. Give it a try now -- it's really quite handy.

The universal argument

The command `universal-argument`, with a key-binding of `C-u`, can be used as a prefix for a great number of other actions, including many of the delete commands I've shown you in the previous panels.

For example, typing `C-u 6 C-k` kills three lines. Yes, that's three lines, not six. Remember that with `kill-line`, the text on the line and the newline are done separately. Not hard to get your head around, once you've used it a few times.

Without a numerical argument, `universal-argument` defaults to a count of 4.

Basic operations in review

Here's a table of all the commands and their key-bindings discussed in this section. Give them a glance and make sure you know what they are. Practice with these briefly to gain more familiarity with the actions. First off, just type in the main window to insert text.

Key-binding	Action (command)
<code>C-g</code> (Esc Esc Esc)	<code>keyboard-quit</code> to get out of a command that's been started
Backspace	<code>backward-delete-char</code>

Delete (C-d)	delete-char
C-x u (C-_)	advertised-undo
M-d	kill-word
M-Delete	backward-kill-word
M-k	kill-sentence
C-x Delete	backward-kill-sentence
C-k	kill-line
C-y	yank is the paste equivalent
M-y	Traverse the kill ring, must follow C-y
C-u, C-u N	universal-argument, adds count prefix to commands

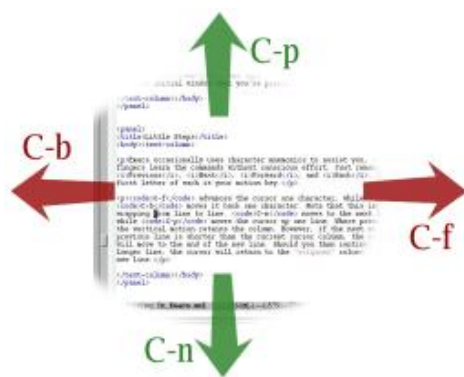
Section 5. Cursor navigation in Emacs

Getting the cursor from here to there

Running Emacs in a GUI environment means you can use a mouse or directional keys like the Up and Down arrows and the Home and End keys to move the cursor around in a document. However, I'm going to review the native navigation scheme for Emacs, since this is the only method that's guaranteed to work, whether you're on a dial-up line from a terminal, accessing a machine via a console or SSH connection, or any of myriad other ways.

The native key navigation has the additional advantage of keeping your hands on the keyboard, where they belong, both for productivity and ergonomic reasons. I find that the context switch between keyboard and mouse costs me about 10% productivity when I'm using a tool in GUI mode.

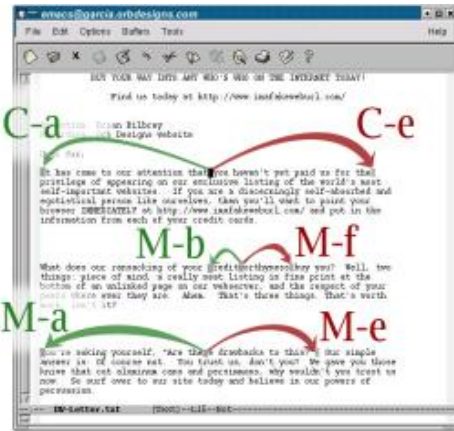
Fire up Emacs as before (type `emacs practice1.txt`), and type a few lines (or copy this panel) into the initial window that you're presented with.



Little steps

Emacs occasionally uses character mnemonics to assist you, as your fingers learn the commands without conscious effort. Just remember *Previous*, *Next*, *Forward*, and *Back*. The first letter of each is your motion key.

C-f advances the cursor one character, while C-b moves it back one character. Note that this includes wrapping from line to line. C-n moves to the next line, while C-p moves the cursor up one line. Where possible, the vertical motion retains the column. However, if the next or previous line is shorter than the current cursor column, the cursor will automatically move to the end of the new line. Should you then continue onto a longer line, the cursor will return to the "original" column, in the new line.



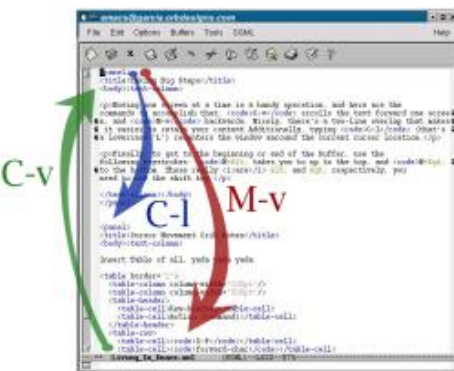
Words, lines, and sentences

To move from word to word, *Forward* and *Back* still guide you, using the Meta key instead of the Control key. Note that words are defined as contiguous spans of letters and numbers. Punctuation counts as whitespace for word movement purposes. Try each of these commands several times as we go over them. *M-f* moves the cursor forward one word, while *M-b* moves back one word.

The mnemonic guidance starts to crumble a bit as we head into more line operations, where the 'a' and 'e' keys are beginning and end respectively. *C-a* takes you to the first column in the current line, and *C-e* takes you to the line's end.

At least we get to keep the same characters for stepping through sentences. Typing *M-a* takes us backward to the beginning of the current sentence (or the previous sentence if the cursor is at a sentence start to begin with). *M-e* moves forward in the same manner, relative to sentence ends.

Sentences are defined by punctuation and either a carriage return or two spaces. Depending on the text, the results might not always yield true sentence steps, but something closer to paragraphs.



Taking big steps

Moving one screen at a time is a handy operation, and here are the commands to accomplish that. *C-v* scrolls the text forward one screen, and *M-v* backwards. Conveniently, there's a two-line overlap that makes it easier to retain your context. Additionally, typing *C-l* (that's a lowercase 'L') re-centers the window around the current cursor location.

Finally, to get to the beginning or end of the buffer, use the following keystrokes: *M-<* takes you up to the top, and *M->* to the bottom. Those really *are* < and >, so you will need to use the shift key.

Cursor movement crib notes

Key-binding	Action (command)
C-f	forward-char
C-b	backward-char
C-n	next-line
C-p	previous-line
M-f	forward-word
M-b	backward-word
C-a	beginning-of-line
C-e	end-of-line
M-a	backward-sentence
M-e	forward-sentence
C-v	scroll-up
M-v	scroll-down
C-l	re-center

Practice these in the test document and keep using them. I found that I had to force myself not to use the cursor keys or the mouse for a while. By keeping my fingers on the home row and consciously using these commands, I was soon navigating through each file's buffer with ease.

Section 6. Search and replace

Incremental searches, part 1

Incremental searches are one of my favorite features in Emacs. These start matching in the text immediately when you start typing. The advantage is that often you don't have to type the whole word before you've completed the search.

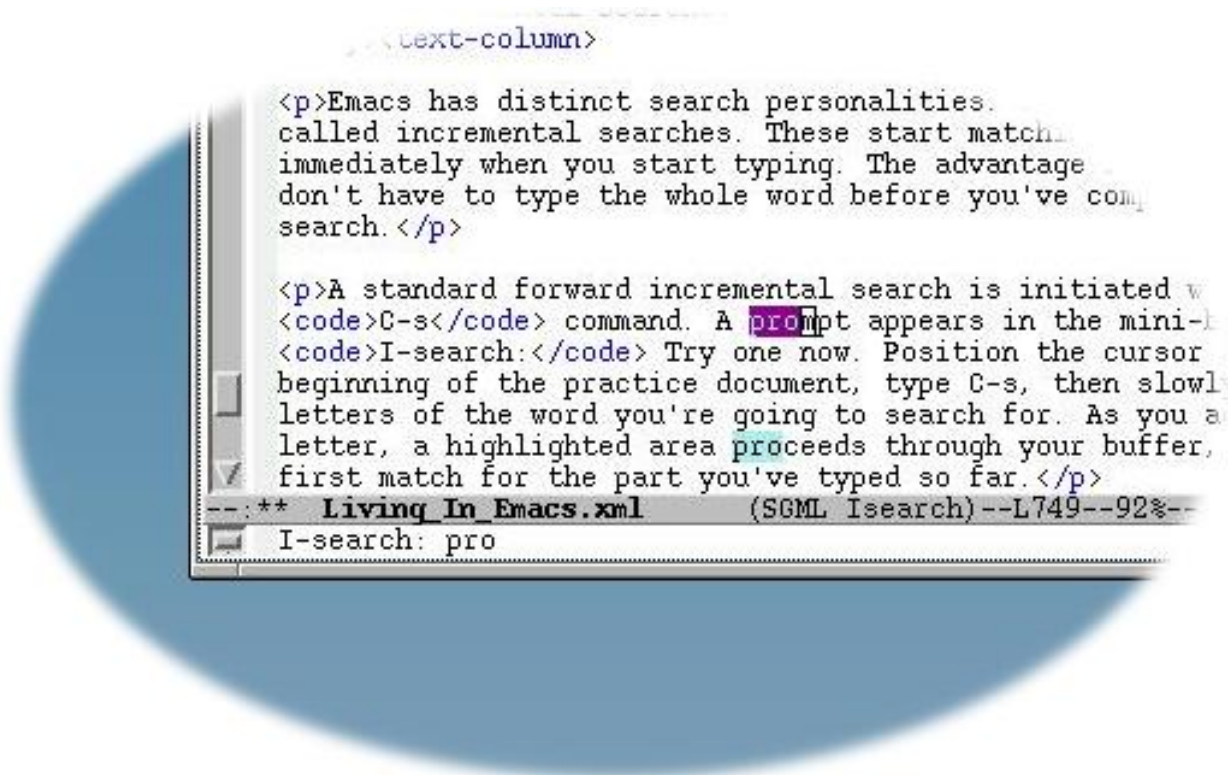
A standard forward incremental search is initiated with the `C-s` command. Searching backwards from the cursor position is accomplished with the `C-r` (`isearch-backward`) command. There are a variety of in-search commands available; you can get a complete description by typing:

`C-h d isearch-forward`

Highlights include incrementing through the matches by typing `C-s` for forward or `C-r` for backward steps. Also, press Enter or `C-g` to terminate the search when you've reached your goal.

Incremental searches, part 2

Try using incremental search now. Position the cursor at the beginning of the practice document and type `C-s`. A prompt appears in the mini-buffer -- `I-search:`. Then, slowly type the letters of the word you're going to search for. As you add each letter, a highlighted area proceeds through your buffer, showing the first match for the part you've typed so far. In the screen fragment below, you can see the first match bounded in magenta, with the next potential match in light green.



Regex searches

Regex searches are also incremental but make use of regular expressions to permit more powerful search capabilities. I won't cover regular expressions in this tutorial, but you can find many good resources in print and online (see [Resources](#) on page 25).

To start a forward regex search, type `ESC C-s` (that is, Escape then Control plus the 's' key). To search backwards similarly, use `ESC C-r`.

For example, let's say that I have the words *bartok* and *footok* someplace in my text for a weird reason. I want to find the closest instance of either one, and I can use a single regex search for the purpose, instead of searching for both and taking note of line numbers, etc.

From this point, I'd perhaps type

```
ESC C-r bar\|foo
```

which first matches the *bartok* above. Then as I add the "or foo" part of the expression, the command re-checks from the point of search and finds that *footok* is indeed the closest. From here, I can use `C-r` or `C-s` to increment through the assorted foo's and bar's in the buffer, backward or forward respectively.

Replacing text

There are two basic types of replace commands in Emacs. The first is an unconditional replace, based either on string or regular expression specification. There is no key-binding by default (I must therefore conclude that it's not regarded as significant), but it can be accessed by typing `M-X replace-string` (or `M-x replace-regexp`). This is followed by the target string/expression and the replacement string. Replacement is unconditional and forward from the cursor location only.

The second command, `query-replace`, is bound to `M-%` (another shifted keystroke). After typing in the target and replacement strings at the prompts in the mini-buffer, each match in turn is highlighted, and you're prompted for the action to take. Pressing `?` displays the complete list of possibilities here. The most common are `'y'` to replace and continue, `'n'` to skip and continue, `'q'` to quit, and `!` to replace all the remaining matches unconditionally.

Try out these commands in a practice buffer.

Search and replace summary

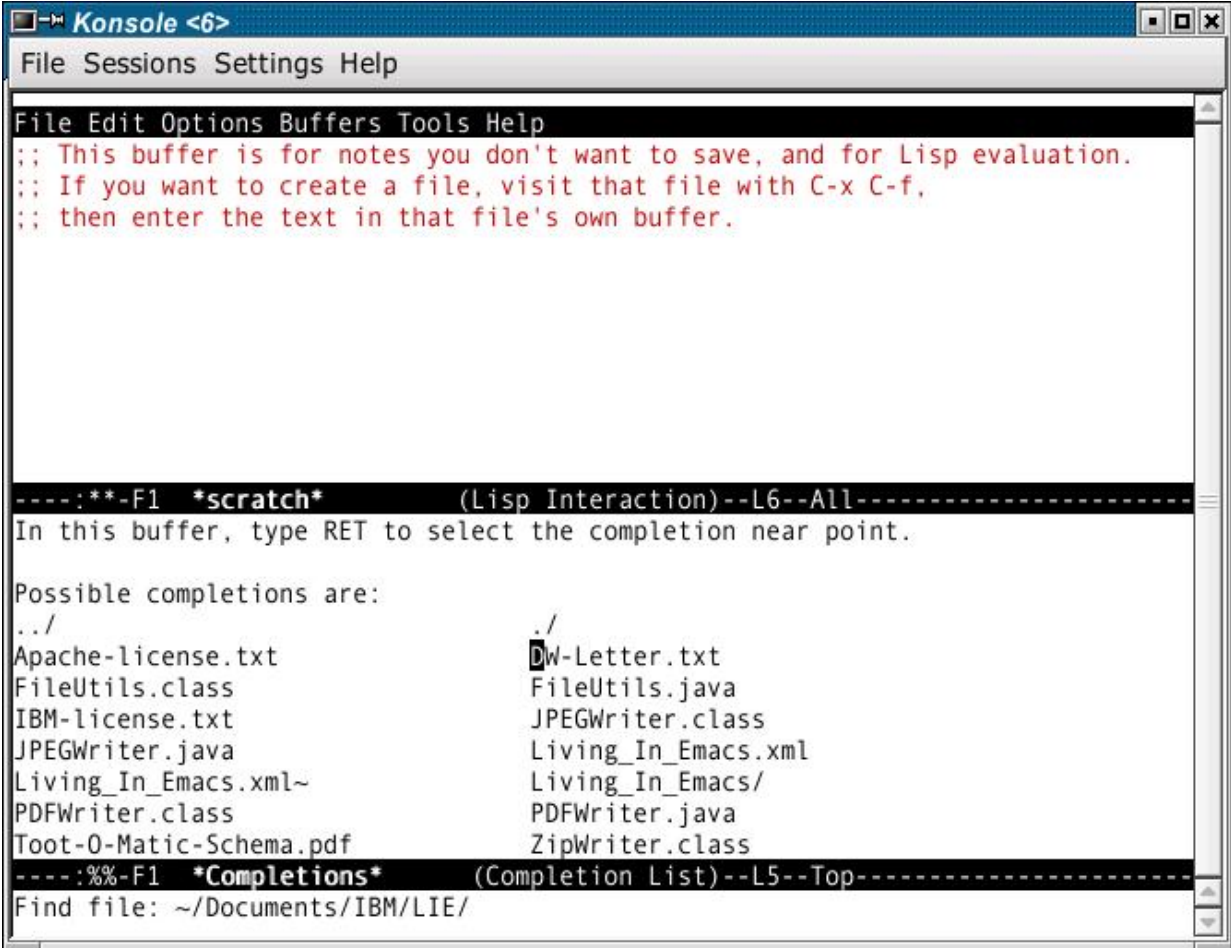
Here's the table summarizing the fundamental search and replace capabilities of Emacs that we've covered. Remember that you can get detailed help for *any* command, with or without a key-binding, by typing `C-h d command-name`.

Key-binding	Action (command)
<code>C-s</code>	<code>isearch-forward</code>
<code>C-r</code>	<code>isearch-backward</code>
<code><find></code>	<code>search-forward</code>
<code>Esc C-s</code>	<code>isearch-forward-regexp</code>
<code>Esc C-r</code>	<code>isearch-backward-regexp</code>
n/a	<code>replace-string</code>
<code>M-%</code>	<code>query-replace</code>

Section 7. Buffers and files

Finding files

In Emacs, files are not edited from disk. Instead, a copy of the specified file is placed into a buffer and all editing takes place in the buffers; writing back to disk file is an explicit action. When you want to get a file into a buffer to edit it, you "find" it. Typing `C-x C-f` yields the following default prompt in the mini-buffer: `Find file: ~/.` Press `Tab` a couple of times to get a directory listing that you can navigate through to get to the desired file (as shown in the image below). Then press `Enter` to read the file into a buffer.



```
Konsole <6>
File Sessions Settings Help
File Edit Options Buffers Tools Help
;; This buffer is for notes you don't want to save, and for Lisp evaluation.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.

----:**-F1 *scratch* (Lisp Interaction)--L6--All-----
In this buffer, type RET to select the completion near point.

Possible completions are:
./
Apache-license.txt          DW-Letter.txt
FileUtils.class            FileUtils.java
IBM-license.txt            JPEGWriter.class
JPEGWriter.java           Living_In_Emacs.xml
Living_In_Emacs.xml~      Living_In_Emacs/
PDFWriter.class           PDFWriter.java
Toot-O-Matic-Schema.pdf   ZipWriter.class
----:%%-F1 *Completions* (Completion List)--L5--Top-----
Find file: ~/Documents/IBM/LIE/
```

Autosave, save, and save as

Now for the good news -- Emacs does include an autosave option, which can be configured to save your files to a specific location. With my configuration, the autosave file for this tutorial, for example, is named `#Living_In_Emacs.xml#`, and the location is the same directory as the original file. Other configurations have different naming conventions and save locations

(commonly `/var/tmp`). By default, Emacs autosaves after 30 seconds of idle time, or after 300 input events.

I introduced the `save-buffer` command at the beginning of this tutorial: `C-x C-s`. To save the contents of a buffer as a different file name, the corresponding key-binding is `C-x C-w`. A path/filename prompt appears in the mini-buffer that can be expanded to a directory listing with a pair of Tab strokes, as with finding files.

Note that after using the `write-file` command to do a Save As, the buffer is associated with the new file name. If you're accustomed to an editor retaining the original file name, this may take a bit of getting used to.

Buffers at start-up

Working in multiple buffers is a snap. However, Emacs is natively a text application, so there's a group of commands for switching buffers and viewing them. When Emacs starts without a file argument, there are two initial buffers called *scratch* and *messages*. Other buffers that you open are named after the files whose contents they contain for editing purposes.

The scratch buffer is aptly named. Use it for temporary storage and for quickie Emacs LISP macro development and testing. It is not saved when Emacs exits, so don't leave anything there that you care about. Messages is a buffer that contains the "system-level" output of commands and background activities, as shown in the excerpt below.

```
Loading sgml-mode...done
Auto-saving...done
Wrote /home/bilbrey/Documents/IBM/LIE/Living_In_Emacs.xml
Auto-saving...done
```

Windows in Emacs

Next, there are windows to consider with Emacs. To start with, you can get two views of the current buffer by typing `C-x 2`, to split them horizontally (`C-x 3` splits them vertically instead). This doesn't open a new buffer, since that would be an independent copy of the data. Instead, it's a window into the same buffer.

To switch between visible windows, the key-binding is `C-x o`, which is bound to the command `other-window`. It cycles through the visible windows. When learning Emacs, I most frequently used this command to swap in and out of the help window. It's also very helpful in coding when I need to frequently swing back and forth between module and header files.

To reduce your window count to just one, type `C-x 1`, which *maximizes* the window that currently contains the cursor, closing other windows.

Buffers in action

To experiment a bit with buffers, first open a couple of test files. Then list all the buffers using the command `C-x C-b`. Your listing should resemble this:

```
MR Buffer          Size Mode          File
-- ----          -
.* practice1.txt   490 Text           ~/practice1.txt
  test2.txt        1  Text           ~/test2.txt
  test1.txt        0  Text           ~/test1.txt
* *scratch*       191 Lisp Interaction
* *Messages*      501 Fundamental
```

The MR column reflects the "Modified" and "Read-Only" status of each buffer. Buffer (name), Size and File are self-explanatory, and we'll address *modes* towards the end of the tutorial. Switch to the buffer listing window (using `C-x o`), then position the cursor on the line of the new buffer you wish to open, and press Enter to select it. The buffer listing is replaced in the window with the selected buffer. Then you can maximize that window, if you wish.

Additionally, there are a variety of buffer-menu related commands and associated key-bindings. List them in a window by typing `C-h d Buffer-menu-`, and then press Tab to get a command listing. The most useful for me is 'q', for quitting. This doesn't close the window that was opened, however. You'll need to do that yourself.

More about buffers

If you know the name of your destination buffer (which is usually the case for me), then using the buffer listing is overkill. Type `C-x b` to get a prompt in the mini-buffer, and then type the name of the destination buffer -- or at least type enough so that tab completion works. Press Enter to open that buffer in the current window.

To kill the current buffer, type `C-x k`. The first prompt that appears in the mini-buffer confirms the name of the buffer being killed. If the contents of the buffer are unchanged, pressing Enter closes the buffer. Otherwise, there's a second confirm that requires a "yes" or "no" response to discard a modified buffer.

Review: files, buffers and windows

The key concepts to take away from this section follow:

- Files are entities on disk.
- Buffers are copies of the data in a file, editable by Emacs.
- Windows are views into buffers.

The following table summarizes the commands found in the preceding panels.

Key-binding	Action (command)
<code>C-x C-f</code>	find-file

C-x C-s	save-buffer
C-x C-w	write-file
C-x 2	split-window-vertically
C-x 3	split-window-horizontally
C-x o	other-window
C-x 1	delete-other-windows
C-x C-b	list-buffers
C-x b	switch-to-buffer
C-x k	kill-buffer

Section 8. A glimpse of the depths

Modes

Modes are the methods by which Emacs features are expressed in the context of specific types of content. That is, indenting behaves differently in a C source code file than in an HTML file or in a letter to your boss. For any buffer, the major mode is shown in parentheses to the right of the buffer name on the status line.

There are two different types of modes: major and minor. Major modes are only active one at a time, but they modify how the minor modes are interpreted. For example, in most coding, indents only happen in the context of the previous line. Press Tab in a `.txt` document (which has automatically invoked the text-mode), and a tab character is inserted and shown as 8 columns.

On the other hand, in this XML document, Emacs invokes SGML mode by default. Here pressing Tab only has an effect if there is leading whitespace on the previous line, in which case the cursor is placed in the first column that matches non-whitespace above, and the distance is filled with space characters, not a tab character.

Indent behavior is part of a minor mode, whose activity is modified by the major mode currently invoked. Other immediate evidence of modes is the differences in syntax highlighting and the way that text is autofilled.

More about modes

The major mode is usually correctly set by Emacs based upon the filename or sometimes by the content in the file. You can explicitly set the mode of a buffer by typing `M-x` followed by a valid mode name.

For example, if I open a file named `bob.txt`, the buffer will open in text-mode. To start working in c-mode, I can type this: `M-x c-mode`. This actually invokes `cc-mode` for me, according to the messages buffer, and is shown on the status bar as `C Abbrev`).

To list all the major modes that Emacs recognizes automatically, type `M-x describe-variable`, press Enter, and then type `auto-mode-alist` at the prompt in the mini-buffer. Some of the common modes I've worked in include text, c, SGML and occasionally LISP.

Compiling code

From a look at the list of modes, there are clearly many things you can do with Emacs. First and foremost, though, Emacs is a programmer's editor. Among other things, you can code, compile, debug and test software, all within the Emacs environment. I won't touch on all of these subjects here, but let's presume that I have written a typical C-language *Hello World* type

of program.

Once I've typed in the program and saved the buffer to disk, I type `M-x compile` and the prompt in the mini-buffer reads, `Compile command:` with perhaps a default after. I type in `gcc -o hello hello.c` and press Enter. A compilation window opens containing the following text:

```
cd /home/bilbrey/  
gcc -o hello hello.c
```

```
Compilation finished at Sun Mar 17 16:18:55
```

To see if my program works, I'll run it from inside Emacs: `M-! ~/hello`. There in the mini-buffer is my output: "Hello, World!"

Emacs and LISP

The name Emacs is a sort of acronym for *Editor MACroS*. So another common coding experience inside Emacs involves setting variables and writing macros in the Emacs version of LISP. LISP has been derided as the acronym for "Lots of Insane Stupid Parentheses", but it has been a successful language in a number of areas, not least with Emacs.

Emacs variables can be set (once you know their names and the appropriate values) from the command line by typing `M-x set-variable` and then entering the variable name followed by the new value at the prompts. Or you can set variables by evaluating them directly. I'd suggest using the scratch buffer for this purpose. To borrow an example from the "LinuxDoc Emacs Beginner HOWTO" (see [Resources](#) on page 25), let's modify the width for the auto-fill mode (or word wrapping):

```
(setq fill-column 20)
```

Once that's typed in, leave the cursor at the end of the line and type `C-x C-e` to evaluate the expression. The result is a 20 down in the mini-buffer. Test it by reformatting a paragraph of text using the `M-q (fill-paragraph)` command.

You can also code new functions as you learn more. Experiment and find settings that you like. You can then place these into your `~/ .emacs` customization file for future use.

Connectivity in Emacs

There are e-mail and Web browsing tools build right into Emacs. To start a new e-mail message, just type `C-x m`. When you're done, `C-x C-s` saves and sends your message. Reading mail is a little more involved. Web browsing is accomplished either by sending a URL to an external browser or by running a textmode browser like Lynx directly inside Emacs. Type `M-x browse-url-lynx-emacs` to invoke Lynx, enter the URL, and you're off. The example running below shows e-mail composition and browsing in a single terminal mode Emacs session.

The screenshot shows an Emacs shell window titled "Shell - Konsole <3>". The window contains an email message and a menu. The email is from Greg to the user, thanking them for reviewing the Emacs tutorial. Below the email is a menu for the Linux Documentation Project (LDP) with options like Mirrors, Non-English info, Translation effort, Translated Guides, Translated HOWTOS, and Main site. The window also shows status lines for "mail" and "lynx" and a message "Auto-saving...done".

```

Shell - Konsole <3>
Session Edit View Settings Help
File Edit Options Buffers Tools Signals Terminal Help
To: Greg
Subject: Thanks for reviewing the Emacs Tutorial...
--text follows this line--

Greg -

Thanks for taking the time to go through the tutorial and tell me
about the bits I got wrong. It's a big help and it'll make it much
better for the readers. I'll let you know when it's been posted!

.brian
-----F1 *mail* (Mail)--L8--All-----
Linux Documentation Project (pl of 17)
LDP worldwide
- Mirrors
- Non-English info
- Translation effort
- Translated Guides
- Translated HOWTOS
- Main site
-- press space for next page --
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /-search [delete]-history list
-----F1 *lynx* (Term: char run)--L4--Bot-----
Auto-saving...done

```

All work and no play...

Of course there are games built right into Emacs ranging from "Towers of Hanoi" and "Life through a variant of Tetris" (tm) to a remodel of the venerable "Adventure," which is demonstrated in the following listing:

```

E/W Dirt road
You are on the continuation of a dirt road. There are more trees on
both sides of you. The road continues to the east and west.
There is a large boulder here.
>look at boulder
It is just a boulder. It cannot be moved.
>climb boulder
You can't climb that.
...

```

There's even the famous Rogerian psych program, Eliza, to get you through the rough spots at 0300 when nothing you do seems to compile. Type `M-x doctor`. It's far cheaper than *any* of the 900 numbers, right?

Section 9. Summary, resources, and feedback

Stick a fork in it...

We're done! Congratulations on a job well done. When I first dabbled in Emacs too many years ago, I found it somewhat ... intimidating. I've done my level best to give you a good grounding in the concepts and usage of this powerful editing tool.

Use Emacs. Live in it for a while, learn to love it a little bit. Like any complex program, it will take time for you to fully grok it, but the effort's worthwhile: you'll have become fluent in one of the most common UNIX programming utilities and picked up a fundamentally marketable skill to boot!

Resources

Here are a few references to speed you on your way to Emacs mastery:

- [The Emacs Beginner HOWTO](#) at the Linux Documentation Project is a good jumping off point.
- [The GNU Emacs Manual](#) is your ultimate reference for the editor, linked to from the [GNU Emacs homepage](#).
- If XEmacs is your preference, you need to go to the [XEmacs home page](#).
- Here at IBM's developerWorks site, you can find extended resources useful in extending your powers with Emacs, for instance with [Using Regular Expressions](#).
- And no discussion of Emacs is complete without a reference to its arch-rival in the Linux text-editor space. Check out [vi intro - the cheat sheet method](#), and see what it's like for yourself.
- The dead-tree resources of choice for Emacs are the ever-popular [Learning GNU Emacs](#) and [GNU Emacs Pocket Reference](#), both from O'Reilly. Pick them up at your favorite online or corporeal bookseller.
- Emacs is licensed under the [GNU GPL](#) by the Free Software Foundation.
- For more on the history of Emacs (and many, many other UNIXish topics), consult [the Jargon File](#).

Your feedback

We look forward to getting your feedback on this tutorial and on future directions in providing up-to-the minute information about the fast-moving Linux arena. You may also contact the author directly at billbrey@orbdesigns.com.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.