

The Woodnotes Guide to Emacs for Writers

Randall Wood (www.therandymon.com)

4th July 2007

Contents

1 License and Version History	2
2 Introduction: Why a Text Editor instead of a Word Processor?	3
3 Xemacs or Emacs?	4
4 Setting up your environment	4
5 Adjust your vocabulary	5
6 The basics	6
7 Files (Opening, Saving, Printing, etc.)	7
8 Getting around with the cursor	8
9 Scrolling	9
10 Bookmarks	10
11 Selecting Text ("Regions")	10
12 Cutting and Pasting (Killing and Yanking)	10
13 Searching and Replacing	11
14 Word wrap (Filling)	13

15 Formatting	15
16 Multiple Windows, Buffers, and Tabs	17
17 Spell Checking	18
18 Macros	19
19 Customizing keys	19
20 Customizing Your Work Environment	20
21 Emacs and \LaTeX	21
22 Learning more about emacs	22

List of Figures

1	Emacs Vocabulary and meaning	6
2	The Basics	7
3	File Commands	8
4	Cursor Movement	9
5	Scrolling	9
6	Bookmarks	10
7	Killing (cutting)	11
8	Killing and Yanking	11
9	Accumulating Text in Buffers	11
10	Searching and Replacing	12
11	Transposing, Joining, and Formatting	16
12	Misc. Environment Settings	21

1 License and Version History

This document is published under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 license as described at <http://creativecommons.org/licenses/by-nc-sa/2.5/>. Please send comments, criticisms, and corrections to me at the email address found at my website. Enjoy this guide: I enjoyed creating it.

- 4 July 2007: Added info on changing colors and font display
- 6 October 2006: Added licensing information and this version history. Over 2000 people have downloaded this guide!
- 2 March 2006: second update incorporating comments
- 1 January 2005: Original version, courtesy of a cold spell in Washington DC

2 Introduction: Why a Text Editor instead of a Word Processor?

Computers have become a mainstay of the work environment, and a lot of us use them to write. Of the many tools out there that allow you to create files of text, word processors have stolen the limelight for most, and it's true that for some kinds of documents – short memos and office forms and such – these tools are the best.

But for other kinds of documents word processors are certainly not the best tool for the job: long documents, documents that require careful organization, fiction and other long works of text, and for that matter, documents that require the ultimate in compatibility between computers, systems, and time. For this last purpose, only pure text is guaranteed to transfer successfully—it's the lowest common denominator. Macintosh users call it "simple text" and edit it using the program by the same name. Windows users call it "plain text" and edit it using Notepad. There are other programs available of course. But because so much configuration on those systems require text files Unix and Linux users are better acquainted with text files in general, and several extremely powerful text editors exist Emacs is one of them, and in my opinion, a wonderful solution for writers.

If what you write is text, be it fiction, poems, massive emails, or the like, you may be well served by a text editor over a word processor. Besides the advantages listed above, and particularly the fact that you are guaranteed the software to read your file will exist in the future, there are many other reasons to use text editors instead of word processor. First of all, word processors distract you with usually unnecessary options. Because your editor is going to do the formatting, you will probably be requested to send in a document devoid of any mark up (bold, italics, etc.). And word processors can be distracting in their incessant highlighting of misspelled words that beg to be corrected before moving on – thus interrupting your train of thought. Word processors make you focus too much on the document as a document and distract you from what you should be doing, writing – being creative – producing text. Why worry about page breaks during the writing process? You should be dealing with that at the end, or not at all. In sum, text editors allow you to concentrate on writing, not formatting.¹

Because text is so important to Unix, a tremendous number of powerful text editors are available to you, nearly always without charge. This document describes how to use one

¹If you truly need formatting for a long and complex document, you will probably be better off with a more powerful publishing tool than your typical word processor anyway.

of them – emacs – to write. Windows and Macintosh users, don't despair – versions of emacs exist for your platforms as well. Because many use emacs for developing software – computer code – the focus of this document is rather unique. But emacs is a powerful and extremely useful way to deal with text directly. Give it a try to write your next book, report, or thesis. You may find it a better tool and more worthwhile ally than you expect.

3 Xemacs or Emacs?

The first question you will probably ask yourself is “emacs or xemacs?” But hopefully you won't waste too much time deliberating, because the answer is essentially, “it doesn't matter.” There's a deep historical and political split that divides the two software programs, and now that so much water has passed under the bridge it's unlikely we'll ever see a reconciliation. But from the writer's point of view, the two programs are identical: they both have the same functionality, a nearly-identical interface, and mostly-identical commands, so much so that I can write this manual for both programs without having to specify all the time which program I'm talking about.

That said, let's look at the distinctions: one of Xemacs' design goals was to streamline the emacs interface in ways that made it more logical from the new user's point of view, and integrate it more fully with the X window system (i.e. the graphical display). This happened at a time when emacs was still a console program. Since then, emacs has made its peace with X and has a graphical display of its own. Since splitting, both programs have borrowed liberally from the other.

4 Setting up your environment

4.1 Your Operating System

Your operating system—Linux, Mac, Windows, Unix—usually determines the software available to you, not the other way around. But emacs has been popular enough an application that great effort has gone into porting it to other platforms.

It's primarily a Linux/Unix application, of course, but Mac OSX is Unix under the hood, so it runs emacs natively, and the console version, not the GUI version, is even provided free with new Macs. Open a new Terminal window (look in your applications folder) and enter the command "emacs" (no quotes of course) at the prompt, and emacs will start. But there's a better option. Enrico Franconi (www.inf.unibz.it/~franconi/mac-emacs) has developed a fully carbonized version of emacs which provides the best of all possible unions between emacs and the Mac OSX operating system. The Mac menubar overhead takes the emacs menu, the program interacts perfectly with the Mac clipboard, and you can launch it directly from the OSX Dock. Moreover, the Apple Command key ("flower") serves as "Alt" (shown as M- in this document), which is far more convenient than reaching for the Escape key as you have to do otherwise. This was

essentially a one-man effort and a fine example of how the open-source software model provides software because empowered and impassioned individuals make it happen.

Plain old emacs runs on Windows, but with some trouble, while there's a very good port of Xemacs that installs painlessly. Equally painless is the Windows NT port of emacs, available at www.gnu.org/software/emacs/windows/ntemacs.html.

4.2 X or Console?

This is a question for Unix/Linux users, and you probably already know the answer because it's a matter of personal preference, but it's worth looking into. Emacs was a console application for years, but developed an optional graphical interface (GUI) because users demanded one, and now you can use whichever you like best, or both.

I like using the console because it seems to utilize the screen efficiently and removes all the graphical distractions of the GUI. Emacs presents a menu bar at the top of the screen which you interact with using keyboard commands, and a status bar at the bottom of the screen bearing pertinent information. But that's my choice when I'm using Linux. If you're the kind of person that likes a GUI, that option is available to you, and if you want a clever sort of compromise you can always run the console version of emacs in an X terminal by issuing a command like "xterm -e emacs -nw." The -nw flag tells emacs to run the console version, not the GUI version. As a bonus, this compromise allows you cool options like running emacs in transparent consoles, and so on (atertm -tr -sh 60 -fg white -bg black -e emacs -nw).

5 Adjust your vocabulary

Emacsen – by which I mean Emacs and its variants, namely Xemacs (see, we've begun using new vocabulary already!) are harder to decipher than your average software because they don't use the vocabulary words you'd expect them to and therefore it takes a longer time to find what you're looking for in the manual or learn the options you'd like to understand. You may very well know what you want but not know how to find it. A simple vocabulary lesson will set you a long way forward in your effort to learn to use emacs. Here we go.

- Frames are what any other program would call a "windows." An emacs frame can be simply another view of the same document or show different documents.
- Buffers are what any other program would call a "file." In emacs you can load in several documents to edit and then cut/copy/paste between them, as you would expect. Each file that you load occupies a slot in memory and is called a buffer, and the word "file" is reserved for what's stored on the disk. In most cases it's the same thing.

- Windows are subdivisions of a frame. In other programs you can work on various files at one time; each one is shown as a “tab” which you can switch between. In emacs, the concept is called a window. You can have two vertical windows, two side-by-side windows, and more.
- Filling is what other programs call "word wrap," sort of. To get a paragraph of text to "wrap" you must essentially insert a return character at the end of every line at a certain position, say every 80th character or so to have paragraphs formatted 80 characters wide. Word processors deal with this automatically and even reformat automatically as you resize the window. Emacs does not.
- Kill means to remove text. Everyone else calls it "cut."
- Yank means to insert previously removed text, i.e "paste."
- Copy to Kill Ring is the equivalent of "copying" text in other applications.

Figure 1 shows a summary of emacs lingo and their equivalents for other software packages.

Other Software	Emacs
Window	Frame
File	Buffer
Tab/Pane	Window
Paste	Yank
Formatting/Justification	Filling
Cut/Paste	Kill/Yank

Figure 1: Emacs Vocabulary and meaning

6 The basics

Let's start using emacs. First of all, a word about notation: emacs commands all start with the control key or the alt key. The command Control-X is shown as follows: C-x and the command alt-X is shown as follows: M-x (the M stands for "meta" and goes back to the days before the alt-key). Some commands involve several steps, like the following, which sets the margin to 20 characters: C-u 20 C-x f. So hit control-u, type the number 20, then hit control-x, release, and strike the f key. The status bar at the bottom of the screen shows your progress. If you screw up half way, hit Control-G to cancel the command (you can't edit the command half way, you have to start over). Now that you understand the notation, you'll understand C-x C-c, which means "exit emacs." Use it now if you decide not to go any further.

You'll feel more comfortable exploring emacs once you know how to undo mistakes. The command is C-x u and emacs remembers a long history of your previous commands so it can undo a lot of mistakes. Figure 2 summarizes these basic commands.

Cancel	C-g
Quit emacs	C-x C-c
Undo	C-x u

Figure 2: The Basics

6.1 How Emacs commands function

The "mac" in "emacs" came from the word "macros." Every single command available to you, and commands you write yourself (see Section 18: Macros below) is a function written in a programming language called emacs-lisp. They all have names like canonically-space-region or indent-region or ispell-buffer. Many of those commands are associated with keyboard shortcuts like C-n (to move the cursor down one line) but they also have a long name as well (in this case C-n is the command next-line). Not all of the commands available to you have a keyboard shortcut; those that don't are accessed by hitting M-x and their long name. For example, type M-x ispell-buffer to begin spell checking the entire document. Once you hit M-x you can type just "isp" and hit tab, and emacs will try to complete the command with the options available to it. In this case it will get as far as ispell- because there are several commands whose names start with that sequence of letters. Continue by typing 'bu' and hitting tab. Emacs will now complete the command: ispell-buffer. Hit return and the spell checking will begin.

In the rest of this document, remember that commands that don't have a shortcut like C-t (transpose characters) can be accessed by typing M-x plus the long name of the command, so M-x transpose-characters and hitting return.

You'll notice as you get more familiar with emacs that commands come in basically three flavors: those that begin with C-x, those that begin with C-c, and those that begin with M-x. Don't despair – it's not totally random (though some things about emacs commands certainly will seem bizarre). In general, C-x is reserved for the most common commands and particularly those that involve reading in and saving buffers, so C-x C-f to "find" a file, C-x C-c to quit, and so on. Commands that are less frequently used get relegated to C-c, which you'll notice is a little more of a stretch for your finger to reach. M-x, as explained above, is used to access commands by their long names. While there are a couple of other key combinations, like C-h for commands related to the help system, these are few in number.

7 Files (Opening, Saving, Printing, etc.)

Remember, a file is what's stored on disk. Once you read it into emacs to begin editing, we refer to it as a buffer, because it's been stored in memory. In general, opening, saving, and printing files is straightforward. A couple of notes: To "open" a file and to create a new file are the same thing: emacs will try to find the file you request and if it doesn't exist it will simply create a new, empty file for you. When you "save as" by typing C-x

C-w, emacs will save the file under a new name and then continue to edit the newly-renamed file. This is what most programs do, but it's worth mentioning because there are some programs that simply save your current work to disk under a new name but continue to edit the original file (called "save a copy as").

Inserting a file at the present cursor position is extremely useful. I use it to send templated emails. I start off with a personalized introduction, then insert a file which contains one of several templates, edit as necessary, and send. You'll find other uses for this feature as you go.

The basic commands are shown in Figure 3.

C-x C-f	Open ("Find") a file
C-x C-s	Save
C-x s	Save some or all files to disk
C-x C-w	Save as
C-x i	Insert another file into current buffer
C-x C-v	Replace this buffer with another file

Figure 3: File Commands

Finally, there are several ways to print, but this is one of the nicest:

M-1 M-x ps-print-buffer-with-faces

It's a long and abstruse command, but it will print your text buffer to the printer with fonts, syntax highlighting, and all. It generates very attractive print-outs of your work.

8 Getting around with the cursor

Unless your system is poorly configured or you're working over a strange network connection of some sort, the arrow keys and page up/page down buttons should work as expected. That's a start. With time you'll get to know emacs' other way of maneuvering the cursor up and down, and once you get used to it you'll find it's faster because you don't have to take your hands from the keyboard. The commands are as follows:

- C-f Forward (i.e. to the right)
- C-b Backward (i.e. to the left)
- C-n Next line (i.e. down)
- C-p Previous line (i.e. up)

You can repeat any number of times by prefixing an argument with the command Control-u as follows. Let's say you want to move forward 8 characters. Enter control-u 8 control-f all in a row. We'll see the control-u command later in this document for other

commands that require a number, like setting margins to a certain number or characters wide and so on.

In addition to the keyboard commands to move the cursor by one letter, which we've just seen, emacs has commands to move the cursor by other units as well: by a word, to the beginning or end of the current line, by a sentence, and by a paragraph, as shown in Figure 4:

Entity to Move Over	Backward	Forward
Character	C-b	C-f
Word	M-b	M-f
Line	C-p	C-n
Sentence	M-a	M-e
Paragraph	M-{	M-}
Page	C-x	C-x]
Beginning/End of Line	C-a	C-e
Beginning/End of Buffer	M-<	M->

Figure 4: Cursor Movement

9 Scrolling

You're already a bit more efficient by learning to get around with the keyboard. Yes, a mouse allows you to pinpoint where you'd like to put the cursor, but it requires taking one hand from the keyboard. But as we begin scrolling around, you'll see we are saving more time still. Scrolling doesn't change the position of the cursor, it just changes which portion of the text is presented to you, just like using the scroll bar in other software. The commands to scroll are shown in Figure 5. The two most important commands are C-v to scroll down one screenful and M-v to scroll up one screenful. And no matter where your cursor is in the document you can scroll that point up to the center of the screen by with C-l. Together they are an easy way to navigate quickly up and down through the document. But there's a quicker way still to get where you're going: incremental searching. We'll look at that trick in section 13.

Previous/Next Screen	M-v	C-v
Scroll Left/Right	C-x <	C-x >
Scroll Current Line to Center of Screen	C-u C-l or C-l	
Scroll Other Window	M-C v	

Figure 5: Scrolling

10 Bookmarks

As you write, you may find it convenient to place a bookmark at certain points in your text so you can conveniently return at some future point, either because you left a thought unfinished or because it's important somehow. Emacs allows you to set, remove, and name bookmarks for just this reason.

If you're using the graphical (GUI) version of emacs, the bookmarks functions are available to you under the Edit->Bookmarks menu. Otherwise, remember the following commands (the lisp-function is available to you by hitting M-x and the name of the function, e.g. M-x bookmark-jump).

Action	Keystroke	Lisp Function
Set Bookmark	C-x r m	(bookmark-set)
Jump to Bookmark		(bookmark-jump)
Delete Bookmark		(bookmark-delete)

Figure 6: Bookmarks

11 Selecting Text ("Regions")

Most anything you do, including cutting and pasting (see section 12 below) involves selecting or highlighting an area of text. To do so, you position the cursor somewhere and set a mark, then move to somewhere else and define everything between the mark and your current position as the region. Once you've selected the region you can go on to cut it, format it, etc.

So, put the cursor somewhere and press C-space. The status bar at the bottom of the screen should indicate "mark set." Now using the scrolling and cursor movement commands described in section 8 to get to where you want. Everything between your current position and the mark should be highlighted and is now considered the region. The next command will affect the entire region. Two quick shortcuts: C-< and C-> will select from the cursor point to the beginning/end of the buffer, respectively.

If for some reason, the region does not get highlighted as you select it, it means transient-mark-mode has been toggled off. Toggle it back on by entering M-x transient-mark-mode. Apparently, not everyone likes to see the highlighting, though I certainly do.

12 Cutting and Pasting (Killing and Yanking)

This is the most obvious example of how emacs doesn't follow naming conventions used by most other software. Killing and Yanking, once you get used to the new vocabulary, does what you'd expect it to, but emacs provides some other tricks that are useful to

writers. First of all, emacs remembers more than one thing cut ("killed") and keeps them in a list called the kill ring. You can later paste ("yank") not just the most recent thing killed but previous things as well. Simply hit C-w to "kill" something (i.e. "cut" it). To "yank" it, hit C-y (i.e. "paste"). To copy something to the kill ring (i.e. "copy"), use the Alt key instead of the Control key, that is M-w. The basics are shown in Figures 7 and 8.

Entity to Kill	Backward	Forward
Character	DEL	C-d
Word	M-DEL	M-d
To end of line	C-k	
Sentence	C-x Del	M-k
Entire line	M-x kill-entire-line	

Figure 7: Killing (cutting)

Let's look for a moment at how the kill ring works. Imagine an immense list of everything you've killed during this session that you can cycle through. When you "yank" some text, the most recent item is what you get. But if you immediately hit M-y, that text is replaced with the previous item. Hit M-y again to replace that with the item previous to that, and so on until you get what you want. It's called a kill ring because you cycle through all the items, eventually returning to the most recent item killed.

C-w	Kill ("cut")
M-w	Copy to Kill ring ("copy")
C-y	Yank ("paste")

Figure 8: Killing and Yanking

Lastly, you can also kill things selectively – a word here, a sentence there – and accumulate them as you go, kind of like a selective harvest of your text. You may not use these tricks frequently, but they're highly convenient when you need them:

append-to-buffer	append region to particular buffer
prepend-to-buffer	add the region to the beginning of a particular buffer
copy-to-buffer	replace specified buffer with contents of region
append-to-file	append region to contents of a specified file

Figure 9: Accumulating Text in Buffers

13 Searching and Replacing

Searching for text is important all throughout the processing of writing, but it's also a useful way to navigate a document too, if you can think of words specific to certain areas of your text that will allow you to pinpoint it. Emacs provides a very powerful way to

search your document, and several additional search and replace mechanisms of use to writers.

The first and best is called incremental searching. When you press C-s emacs will prompt you for what you want to search for. It will then search as you type from the cursor position to the end of the document. But as you continue typing it will add those letters to the search. An example is worth a thousand words. Let's say you're searching for the word "iconoclastic" in your document. Hit C-s and start typing "i-c-o-." As you type "i" emacs will highlight the next word that starts with "i," but as you type "ico" it will highlight the first word that starts with "ico," building as you type until finally you've typed in the whole word and emacs is highlighting the next occurrence of the word "iconoclastic." If that's the position in the buffer you want to skip to, press enter at this point, or C-g to remain where you were when you started the search. C-r performs an incremental search backwards from the cursor.

Type M-% to begin the query replace process. This is the equivalent of search and replace in other software. Emacs will ask you at each word if you'd like to replace that occurrence of the word or no, and you can answer yes, no, yes from now on, cancel, and so on.

One limitation of regular searches in documents that have been formatted (filled) is that if the two words are separated by a newline, the incremental search function won't find them. For these cases, use the word search command: C-s C-m C-w (an easier key sequence is C-s Return C-w). Let's say you hit that sequence of keys and then input the expression "slow rabbit" (no quotes, of course). Emacs will search the buffer for that sequence of words even if they are separated by punctuation or even newlines. That is, it will find any of the following combinations:

```
slow rabbit
slow. rabbit,
slow
rabbit
```

If you find the three part command sequence too burdensome for you, you can access the command directly: M-x word-search-forward, or simply bind it to an easier key to remember, as described in Section 19.

Incremental search forward	C-s
Incremental search backward	C-r
Query replace	M-%
Word search forward	C-s RET C-w

Figure 10: Searching and Replacing

14 Word wrap (Filling)

Quite frankly, this is emacs' greatest disappointment as a text editor – something so fundamentally important to so many people and something so many other editors do well. Without modification, emacs does not soft wrap the way you'd like it to. Will that change? Perhaps – open source software's greatest strength is that clever people with programming skills can modify projects to suit their own needs. But if this is something you can't overlook, emacs will drive you crazy and you're better off with another application.

Word wrap comes in two forms: soft wrap and hard wrap. Hard wrap means that at the end of every line a "newline" character is inserted. Most plain text email is sent this way. If you have 80 character wide paragraphs and want them to be 120 characters wide, you have to reformat. In other text editors that's a real pain, but in emacs it's easy, and that's one reason few programmers feel compelled to do anything about the word wrap problem. Soft wrap means the program recognizes the width of the window on your screen and reformats the words to fit the window, without inserting any newline characters. If you resize the window, the words adjust automatically. Emacs does not do this.

Emacs presents three options:

- No wrap: Lines do not wrap around the screen at all, but continue on and on to the right until you finally hit "return," which for text, would be at the end of the paragraph. This is acceptable for people writing code, but not acceptable for text.
- Wrap: Lines wrap around the screen, but emacs doesn't pay attention to words, and will wrap right in the middle of the word, showing a little symbol, probably a backslash, at the right edge of the screen to show the line is being continued below. If you can deal with this, it's the best way to go for writers. But it's not ideal.
- Fill: Emacs calls it "filling" a paragraph, and it means inserting a newline at a certain distance from the beginning of the line, for example 120 characters, paying attention not to split a word. This gives you nicely formatted paragraphs that look nice, but can get distorted if you add words in the middle of the paragraph afterward, for example. Then you have to reformat the paragraph (which is a simple keystroke). If you're writing using \LaTeX (more about that later), this is the best option. But if you eventually want to hand the text to a publisher, they'll be very unhappy about all the newlines and the hard formatting, so you'll be forced to find an alternative.

Let's look at filling first. The command `M-x auto-fill-mode` toggles filling on or off. It will insert a newline at a certain position, taking care to pass a word onto the next line if it would be otherwise split. At what character will it do so? Probably around 72 unless you tell it otherwise. Here's how to choose: `C-u 80 C-x f` sets the width (80 characters, in this example) of your paragraph but does not reformat the paragraph. `M-q` reformats the paragraph.

So let's say you are typing at the console, which is 120 characters wide, and you are starting a new document. Before you start, hit `C-u 120 C-x f` to set the margin, and type `M-x auto-fill-mode` to toggle auto-fill mode on (check the status bar at the bottom of the screen to see if it's on: look for the word "fill" in the mode line). Now start typing. Your paragraphs will be hard wrapped at 120 characters, the width of your screen. Now if you go back to edit your work, the paragraph will be out of whack. Hit `M-q` to reformat the paragraph. If you later decide you want the paragraph to be 72 characters wide again, you can hit `C-u 72 C-x f` to set the new margin and `M-q` to reformat it.

There are two other useful commands available to you if you've selected a region you'd like to format. The command `fill-individual-paragraphs` (remember, as explained in section 6.1 you would access this by typing `M-x fill-individual-paragraphs`) reformats each paragraph in the region. This is probably what you want if you want to globally change all the paragraphs in your document from 72 to 85 characters wide, for example. The command `fill-region-as-paragraph` will take all the fragments of text in your region and make them into a single paragraph, removing extraneous blank lines and double spaces, etc. Very handy way to reformat hacked-up text.

14.1 longlines.el

There is one way to have the word wrap you want: Emacs is extensible, and by writing and loading packages of elisp code you can add functionality to the program. Naturally, someone has already written a package that deals with the word wrap situation and it's called `longlines.el`. Find the package on the Internet and install it with the rest of the lisp code on your system. Where you install it might not be obvious. On my Linux system it was a matter of copying the file to `/usr/share/emacs/21.2/site-lisp/` and setting the permissions to `-rw-r--`. I added the following line to my `.emacs` file:

```
(autoload 'longlines-mode "longlines.el"
  "Minor mode for editing long lines." t)
```

That line makes the additional functionality available to emacs. But to start using it you have to activate it by issuing the command `M-x longlines-mode`. The status line will reflect the change with the letters `LL`. If you turn off `longlines` mode (by issuing the same command), the text will not be reformatted unless you manually refill the paragraph using `M-q`. I always use `longlines` when I'm writing, because it makes text behave the way I expect it to after using so many other software packages. It's an especially nice effect when you're working at a virtual console and `longlines` wraps your text at the screen edge (120 characters, in my case). If you too use `longlines.el` whenever you write, you can make it load automatically whenever you load emacs by adding the following line to your `.emacs` customization file (see section 19):

```
(add-hook 'text-mode-hook 'longlines-mode)
```

Be aware that `longlines`-wrapped text is subject to the same limitation in searching that auto-filled text is (see section 13), an issue you can easily get around by using the

powerful word search command C-s RET C-w (that is, C-s, hit return, C-w) which is able to find phrases even if they are separated by newlines.

14.2 Reformatting Hard Wrapped Documents

If you have a document that has already been hard-wrapped, getting rid of all those new-line characters and going back to soft wrapping is not quite intuitive either. One place where you'll run into this is reformatting plain text email for use in another program. Before you can do much with the text you need to get rid of the carriage return at the end of each line. There are two easy ways to do this.

The first way is the most simple: First, set your fill-column variable to some huge number greater than the probable maximum number of characters in a single paragraph, like 10,000. Then select the whole document and invoke "fill-individual-paragraphs." Remember, to set your fill-paragraph variable the key binding is C-x f. So to set it to 10,000 you'd hit C-u 10000 C-x f.

The second way is easier, but may only work if you're working on the Linux operating system, which has the following tool available. Simply highlight the area or the whole document, and type:

```
M-1 M-| fmt -w2000
```

This pipes the text to the GNU `fmt` ("format") command with a paragraph width of 2000 characters. If you can remember the keystroke, this is the most elegant way to do it.²

If you're going to use this command frequently it may make sense to define it as a macro and bind a keystroke to it so you can evoke this function with a single keystroke (see section 18 for more info on macros and section 19 for more info on binding keys). This is the code you would add to your `.emacs` file:

```
(defun fix-screwed-up-paragraphs(beg end)
  (interactive "r")
  (shell-command-on-region beg end "fmt -w2000" nil t))
```

15 Formatting

The commands presented here complement the filling techniques described above and provide some additional functionality as well that's useful for writers. Let's start with transposition. Word processors don't offer anything of the sort. The basic commands are shown in Figure 11. C-t with transpose your current character with the character previous and M-t will do the same with words. C-x C-t will do the same with lines

²Thanks to Jerry Sievers for the first technique, and Rod (author of "Linux for Non-Geeks – Clear-eyed Answers for Practical Consumers") for the second technique. Thanks to Marc Girod for the lisp function

(remember a line is not the same as a sentence here). M-T will transpose your current line with the line above it. Emacs offers other transpose commands; find them by typing M-x apropos, hitting return, and then typing "transpose." You'll be surprised by the variety.

C-o is one of the commands I'm most grateful for. It inserts a blank line and forces all the rest of the text down from the position of the cursor. In other programs you have to hit return and then arrow your way back up to do this. The more you use it the more you'll come to like it. Just as useful is the opposite: say you've got several blank lines between two paragraphs and you want to clean it up. Rather than manually deleting each line, just hit C-x C-o to remove all blank lines except one. Two additional commands – M-backslash and M-space clean up space between words, the former removing all spaces and tabs thereby juxtaposing the two words, and the latter removing all spaces but one. The last command is two clean up in general. Invoke M-x canonically-space-region after having selected a body of text. It will remove all extraneous spaces so that there's one space between words and two after a period. If you've seriously mashed up your text, this is a quick way to put it back together. Shortcuts like these are the ones that give you the advantage over word processor-users. M-^ will join your current line to the previous. If you ever want to put together the text of an email that has been hard formatted (hard wrapped), this is a good way to do so, and use it in a macro (see section 18 below) to do so more efficiently, starting at the bottom of the text and working your way up.

C-t	Transpose two characters
M-t	Transpose two words
C-x C-t	Transpose two lines
C-o	Insert blank line
C-x C-o	Remove all blank lines but one
M-backslash	Delete all spaces and tabs around point
M-space	Remove all blank lines except one
M-^	Join this line with the previous

Figure 11: Transposing, Joining, and Formatting

15.1 End of Line Characters

If you frequently deal with text files created by Windows users, you will no doubt encounter the frustrating \M character littered throughout the text. Remember that Unix, Windows, and Macintoshes all deal with the end of lines differently. Windows marks the end of a line with two characters – an end of line (\ n) and a carriage return (\ r). Unix just uses the end of line (\ n), and Macintosh just uses the carriage return (\ r). When you open a text file originally created in Windows, the \M characters represent left-over carriage returns emacs didn't know what to do with. There is an easy way to get rid of them by just searching and replacing. Navigate to one of them, select it the way you would any other character or expression, and copy it using M-w. Then Hit M-% to begin a search and replace session. When emacs asks what to replace, hit C-y (yank). When

emacs asks with what to replace the character, just hit return. Emacs will then remove all those `\M` characters.

16 Multiple Windows, Buffers, and Tabs

Remember that what you would call "files" in other programs are "buffers" to emacs. That distinction becomes important when we start dealing with multiple buffers and introduce the concept of windows and frames (for a quick review look at figure 1).

First of all, let's say you begin working on a file called `one.txt`. But you want to work on `two.txt` as well, maybe because you're going to cut and paste text from one file to the other. So once you've got `one.txt` on your screen, type `C-x C-f two.txt`. The second file should be loaded into a buffer and that buffer should be the active one. What happened to `one.txt`? It's still in memory, but your window/frame is only presenting you one buffer, and it's `two.txt`. You can switch between the two buffers by hitting `C-x b`. Emacs will ask you "Switch to buffer (default `one.txt`). By simply pressing enter emacs will switch to `one.txt`. Alternatively, you can hit the tab key, and emacs will divide into an upper and lower section, and one section will show a list of all possible buffers. Some are special emacs buffers like logs. `*Messages*` is an example of one. You can type the first few letters of the buffer you want to switch to and emacs will auto-complete for you. When you've chosen the buffer you want to work on, press enter. By default emacs will assume you want to work on the buffer you were dealing with last, which means it's easy to switch between two buffers simply by hitting `C-x b` and return, accepting the default.

If you want to see a list of all buffers currently in use, hold down the control key as you hit the `b`, that is, type `C-x C-b`. You'll see a window listing all current buffers in use.

Once you've got two windows open though, how do you get rid of the second one? There are several ways.

- Make your current window (the one the cursor's in) by hitting `C-x 1` (mnemonic: "one window"). The other window will disappear, though as you've already learned, the buffer will still be open, just not displayed.
- Switch to the other window and kill the window. Switch by hitting `C-x o` (mnemonic: "other window"). The cursor will now be in the other buffer. Now kill it with `C-x 0`. The current window will disappear and the other one will occupy the full screen.

Let's look at some other way you can deal with multiple buffers and windows. Hit `C-x 2` to divide the current window in two pieces, one upper and one lower (two vertical windows). Because emacs doesn't know what else to do, both windows will at first show the same buffer, so if you're currently working on `one.txt` and split the window, you'll have `one.txt` in both the top and the bottom windows, and both will reflect changes made in the other (think of them as viewports looking onto the same area of memory). This can be very useful when you want to have on the screen at the same time two different areas of your text.

Hit C-x 3 to divide a window into two side-by-side (horizontal) windows. This can be handy to view side by side two buffers, for example to compare them or look for changes in one.

Needless to say, you can have different buffers in each window. Start with one window and then using C-x 2 divide it into two windows. Now switch to the other window with C-x o and then open up a new buffer with C-x f (to deal with a new file) or C-x b (to deal with an already opened file). Use C-x k to "kill" a buffer (close it permanently; emacs will ask if you want to save changes first) or C-x 0 to close a window without closing the buffer. Now is it clear why it's important to understand the difference between files, buffers, and windows?

Time to look at frames. Remember, a frame is what other programs call "windows" so be careful to distinguish with your vocabulary. An emacs frame is like a detached window, and like windows, just provide a viewport to a buffer, so you can add and delete additional frames without affecting buffers at all. Needless to say, the concept of a frame only works if you're using emacs in a graphical environment. If you're working at a virtual console, frames are not available to you (fortunately windowing commands should be enough to help you get your work done).

Type M-x find-file-other-frame. Emacs will ask you for the name of the file and then open it up in a new frame. Or type M-x new-frame to display the current buffer in a new frame.

17 Spell Checking

Spell checking is one of several ways emacs interfaces well with other software to expand the tools available to you. Emacs spell checks via the ispell program, available on all Unix/Linux systems and on Mac OSX as well (though note Mac OSX uses its own spell checking mechanism, and your ispell dictionary is separate from the others).

To spell check a word, simply hit M-\$ while the cursor is somewhere in or at the end of the word. Ispell will check the word and allow you to correct it, if necessary. To spell check the entire document, enter M-x ispell-buffer and have fun. You can add words to your dictionary as necessary as you go.

Flyspell mode is the equivalent of that check-as-you-go spell checking that some word processors use, and is one of the features that proves you can have in a text editor the same features you have in expensive, proprietary word processors. Enter M-x flyspell-mode to toggle the mode on or off. Flyspell mode uses the ispell program to spellcheck your document as you type and changes to a different color all the words that appear to be misspelled. It only checks what you type from the moment you toggle the mode on, however. If you've already typed quite a bit and would like to flyspell all the existing text, once you've toggled on flyspell-mode, enter M-x flyspell-buffer to have ispell look over your entire buffer for spelling errors. If you get tired of the distraction of misspelled words (it's often better to just do all your spell checking at the end of the day instead of as you go, so you don't get distracted) you can toggle flyspell mode back off again with the same command: M-x flyspell-mode.

Finally, lots of good dictionaries have been developed for ispell, all of which you can download for free. Most word processing programs don't have that luxury. Getting the dictionaries is your own task. Start with the disks that your Linux distribution provided, for example, or download them. But once you've installed them on your system it's an easy matter to enter M-x ispell-change-dictionary and change the dictionary ispell will use for the next command. This is very handy if you use emacs for email and find yourself communicating in more than one language. Ispell is open source software, and it's available not only for Unix/Linux but for Windows and Macintosh as well.

18 Macros

Macros are a great way to take the burden out of repetitive, annoying tasks. As mentioned above, the "mac" in emacs stands for "macros" and it's something emacs does very well. Simply hit C-x (to begin recording a macro, and C-x) when you're done. The status bar at the bottom of the screen will read "defining keyboard macro" as you type. Once the keyboard macro has been defined, it's an easy matter to invoke that macro throughout your typing session by typing C-x e and to re-invoke it by just striking e again as necessary. Or you can select a region and use M-x apply-macro-to-region-lines to edit only a certain portion of your text with the macro (C-x C-k r).

If you want to keep the macro available to you, save it and give it a name: Type M-x name-last-kbd-macro or by hitting C-x C-k n. You can edit that same macro by typing M-x edit-kbd-macro. A new buffer will open with the commands that make up your macro, and you can fine-tune them if you wish.

19 Customizing keys

It is at this point that you may run into differences between Xemacs and emacs, so proceed with caution. Emacsen are most frequently derided for making use of abstruse commands like M-x M-c or horrid key combinations using one or more of shift/alt/control etc.³ You can create easier keystroke combinations to suit your needs by editing the .emacs file in your home directory (for emacs). Xemacs uses ~/.xemacs/init.el for its customization. In the configuration file appropriate for whichever software you're using, add lines like the following:

```
(global-set-key [f1] 'goto-line)
(global-set-key [?\M-\l] 'next-line)
(global-set-key [?\C-x ?\w] 'beginning-of-buffer)
(add-hook 'text-mode-hook 'longlines-mode)
```

The first three lines above set the F1 key to the command "goto-line," Alt-l to "next line" and C-x w to "beginning of buffer." Type C-h b to get a listing of all current key bindings.

³Common joke: "What does emacs stand for?" "Escape-meta-alt-caps-shift"

The fourth line instructs emacs to load the `longlines.el` package whenever it edits in text mode.

20 Customizing Your Work Environment

I've left the subject of fonts and colors until the end, because emacs' biggest challenge for new users is getting familiar with the functionality. But now that you know how to use the tool and have gotten a sense of how customizable it is, it should come as no surprise that you can also change many aspects of its user interface. Here are some commands that help you customize your work environment:

20.1 Colors

The default emacs frame or window might be good enough for some, but if you get tired of black on white or simply prefer another layout like yellow on blue (like the good old Wordperfect for DOS days), or white on grey if you're working at night in low light conditions might give your eyes a rest. And the relentless desktop themers who need everything on their desktop to match color schemes can have some fun with this as well. Set the foreground color and background colors with M-x `set-background-color` and M-x `set-foreground-color`, respectively. As you press return, a buffer window will open up requesting you input the color you like. If you're working in X (as opposed to a console) you have dozens of colors at your disposal. Start typing 'bl' for blue and then press tab and note how many blues are available. Same goes for most colors. For that matter, you can choose the cursor color and mouse color in the same way, namely M-x `set-cursor-color` and M-x `set-mouse-color`.

If you decide to make these your defaults, it's a matter of adding something like the following to your `.emacs` preferences file:

```
(set-foreground-color "white")
(set-cursor-color "red")
(set-mouse-color "goldenrod")
(set-background-color "black")
```

20.2 Default Display Font (“Face”) and Window Size

I have never had much of a problem with emacs' default font, and since writers of long works don't have much need for font changing, once you've chosen a good font you can stick with it. Remember, since this is a text file the font is only used for display on the screen anyway, not printing. First, choose a font you like. That takes some experimenting. Type M-x `set-default-font`, and when emacs prompts you for a font, hit tab to see what's available. Note which one works the best for you, and then add it to your `.emacs` file with something like the following:

```
(set-default-font
 "-Misc-Fixed-Medium-R-Normal--15-140-75-75-C-90-ISO8859-1")
```

Add the following to your `.emacs` file to set a default for new window sizes if you constantly find yourself resizing them as they appear.

```
(setq default-frame-alist
 '(
 ; frame width and height
   (width . 80)
   (height . 40)
 )
)
```

20.3 Menus and toolbars

You might be interested in the menubar and icon toolbar at the top of an emacs window but personally I find the emacs menus unintuitive and the icon toolbar kind of superfluous, and use neither. Fortunately you can turn both off if you like using `M-x menu-bar-mode` (it's a toggle, so turn it on and off with the same command) and `M-x tool-bar-mode`.

To make these preferences permanent, add `(tool-bar-mode -1)` and `(menu-bar-mode -1)` to your `.emacs` file, respectively.

20.4 Other Environment Settings

You'll be amazed by just how many aspects of emacs' behavior are customizable. But you just want to write, not spend all day customizing. In a nutshell, here are some aspects you might want to ensure are set to your liking and then never touch again. Add these to your `.emacs` file to set them and forget about them.

<code>(setq inhibit-startup-message t)</code>	Disables the splash screen
<code>(global-hl-line-mode 1)</code>	highlights the current line
<code>(show-paren-mode t)</code>	Toggles emacs' showing matching parentheses
<code>(setq transient-mark-mode t)</code>	Highlights the selected region

Figure 12: Misc. Environment Settings

21 Emacs and \LaTeX

This Woodnotes guide won't get into the details of \LaTeX but let it be said that if you intend to develop documents using \TeX or \LaTeX software, you could hope for no better

text editor than emacs. Find the Auctex package for emacs – it probably came with your distribution anyway – and install it. Auctex will be loaded immediately whenever you open or begin a file with the extension `.tex`. It makes a whole host of commands available to you at a keystroke and has good documentation that will help you get familiar with its capabilities. It's very convenient to simply hit `C-c C-c` to run \LaTeX on your document and again to view the DVI output file. Auctex also contains provisions to help you organize and create your Bibtex files, and lots more.

22 Learning more about emacs

This Woodnotes guide doesn't even scratch the surface of the power of emacs, but for the writer or author interested in getting down to work, this should be more than enough to allow you to find and use the commands that you need most. There is far, far more about emacs than what's covered here. For example, emacs includes calendar functionality (even the Mayan calendar!), games (even Tetris!), a calculator, a datebook/planner, a fun psychiatrist, and more. If you want to, you can use emacs to read and compose your email, surf newsgroups, and so on. These things may not be immediately useful to writers, but if you decide you like emacs dive in and explore: it's a powerful tool.

If you're interested in learning more about this powerful software package, there are lots of options available to you. Of course, the program has its own help manuals and documentation available on screen. `C-h ?` will give you a list of all help commands and you can choose from there. In the GUI version of emacs, simply navigate the Help menu (on consoles, hit `M-'` to do the same thing). For example, as shown in the help menus, the command `C-h k` allows you to find out what command a particular key sequence is bound to, that is "what does `C-x C-o` do?" `C-h b` will show you all keybindings presently in use.

You can find new commands by using emacs' "apropos" utility. Enter `M-x apropos` and hit return. When emacs asks you for an expression, enter what you'd like to know more about, and it will present you a list of all commands that use that expression. For example hit `M-x apropos`, return, and then enter "ispell" (no quotes). You will see a list of a half a dozen commands, variables, and functions that include the word `ispell`, including `check-ispell-version`, `checkdoc-ispell`, and `ispell-change-dictionary`. All items shown as commands are available to you by entering `M-x` and the command name. Functions and variables are not very useful to anyone but emacs lisp programmers, for whom this document is not intended.

On the web, check out the GNU website: <http://www.gnu.org>, and the emacs wiki, a collaborative website with tips, tricks, and more: www.emacswiki.org. Just Googling for emacs will net you a plethora of websites where emacs fans share their tips.

A lot of us prefer books, and for a good reason. Start with the O'Reilly Guide *Learning GNU Emacs* by Debra Cameron, James Elliott, and Marc Loy, available in the computer section of your favorite bookstore or on line at www.oreilly.com. O'Reilly also produces a useful pocket reference guide (by Debra Cameron) with less explanation but a good list of the most-used commands and settings. Purchase of the books helps support

the programmers that work on emacs, which is nice. Emacs itself has some useful information available to you. If you're using a GUI version of emacs, simply click on the help menu and browse through the FAQ or take the tutorial. From a console, access the menu using M-' (that's the back tick, located to the left of the number 1 on your keyboard) and wind your way through the menus. The tutorial is quite good and will help you gain confidence in using this particular piece of software, although it doesn't cover much more than the basics.

The best way to learn about emacs is simply to start using it. You'll quickly find solutions to your own problems, and the enormous emacs user community is usually more than happy to support you in your effort to learn it. Brace yourself, and come on board. Happy writing!

Thanks to Bill Harris, Kai Grossjohann, Chong Yidong (one of the authors of longlines.el), Johann "Myrkraverk" Oskarsson, Jerry Sievers, Marc Girod, Aidan Kehoe, and everyone else that supplied recommendations and additional resources for this document.

A companion reference card is available for this document at www.therandymon.com/papers/emacs-writers-cheatsheet.pdf.