Emacs Beginner's HOWTO

Table of Contents

Table of Contents

Emacs Beginner's HOWTO

Jeremy D. Zawodny, jzawodn@wcnet.org

v1.7, 14 October 1999

This document introduces Linux users to the Emacs editor. It assumes minimal familiarity with `vi` or a similar editor. The latest version of this document is usually available from http://www.wcnet.org/jzawodn/emacs/

5.Popular Packages

6.Other Resources

7.Credits

1.Introduction

1.1 Copyright

1.2 Audience and Intent

This document is targeted at the Linux user interested in learning a bit about Emacs and trying it out. This actually began as the outline of a brief tutorial that I was to give at a Toledo Area Linux User Group meeting: http://www.talug.org/. It has since grown a bit as the result of the helpful feedback I have received from the community. See the Credits section for details.

Having said that, there is virtually nothing Linux−specific in this document. It applies to virtually all flavors of Unix and even Emacs running on Microsoft Windows. But since this document is part of the Linux Documentation Project, I make a point of saying that it was developed for Linux users−−because it was.

And finally, those of you who prefer the name GNU/Linux to simply ``Linux'' (read http://www.gnu.org/gnu/linux−and−gnu.html to see why one might) are welcomed to mentally substitute

GNU/Linux for all occurrences of Linux in this document. While I don't disagree with the reasoning and spirit behind that idea, I don't feel compelled to write GNU/Linux.

1.3 What is Emacs?

Emacs is different things to different people. Depending who you ask, you'll could get any of the following responses:

- Text Editor
- Mail Client
- News Reader
- Word Processor
- Religion
- Integrated Development Environment
- Whatever you want it to be!

But for our purposes, let's just pretend it's a text editor−−an amazingly flexible text editor. We'll dig deeper into the question later on. Emacs was written by Richard Stallman (founder of the Free Software Foundation: http://www.fsf.org/ and the GNU project http://www.gnu.org/) and he still maintains it today.

Emacs is one of the most popular and powerful text editors used on Linux (and Unix). It is second in popularity only to **vi**. It is known for it huge feature set, ability to be easily customized, and lack of bugs. It's large feature set and ability to be customized actually are the result of how Emacs was designed and implemented. Without going into all the details, I'll simply point out that Emacs isn't ``just an editor''. It is an editor written mostly in the programming language **Lisp**. At the core of Emacs is a full−featured Lisp interpreter written in C. Only the most basic and low−level pieces of Emacs are written in C. The majority of the editor is actually written in Lisp. So, in a sense, Emacs has an entire programming language ``built in'' which you can use to customize, extend, and change its behavior.

Emacs is also one of the oldest editors around. The fact that is has been used by thousands of programmers over the past 20 (?) years means that there are many add−on packages available. These add−ons allow you to make Emacs do things that Stallman had probably never dreamed possible when he first began work on Emacs. More on that in a later section.

There are many other web sites and documents which give a better overview of Emacs, its history, and related matters. Rather than attempt to reproduce much of that here, I suggest that you check out some of the places listed in Section Other Resources section of this document.

Ports and Versions

It's worth pointing out that there are actually two different Emacs editors: GNU Emacs and XEmacs. Both come from the same heritage and share most of the same features. This document focuses on GNU Emacs (version 20.3, specifically) but much of what you'll read here will apply just as well to XEmacs and earlier versions of GNU Emacs. Throughout this document I will simply refer to ``Emacs''. When I do so, bear that in mind.

Getting Emacs

Getting Emacs is easy. If you are using a popular Linux distribution like Debian, RedHat, Slackware, or any of the others, Emacs is probably an optional package that you can install from your distribution media. If not, you can get the Emacs source code and compile it yourself. Visit the GNU web site for the exact location: http://www.gnu.org/software/emacs/emacs.html

---

2.Running Emacs

2.1 Starting & Quitting Emacs

As a new user, you'll probably want to launch Emacs just to mess around and try it out. Once you're into Emacs and want to exit, however, you may not be able to figure out what to do. So if you've never used Emacs before, give it a shot right now. At your shell prompt, type `emacs` and hit enter. Emacs should start up. If not, it is either not installed or not in your path.

Once you've seen Emacs, you need to know how to exit. The keystrokes for leaving Emacs are `C-x C-c`. The `C-x` notation means hold down the `Ctrl` key and press `x`. In this case, you'll then need to hold down `Ctrl` and press `c` to finish the task.

The keystrokes used in Emacs will likely seem odd, foreign, and maybe even uncomfortable to you at first--especially if you're a `vi` user. Unlike `vi`, Emacs doesn't have separate modes for editing text and issuing commands.

To re-cap: `emacs` will start Emacs. `C-x C-c` will exit Emacs.

What you'll see

When Emacs starts up it will consume a whole X window (or screen if you're running on a console instead of in the X Window System). You'll see a menu across the top, some text in the main part of the screen, and a couple of lines at the bottom.

It will look something like this ASCII sketch:

```
+----------------------------------------------------------------------+
|Buffers Files Tools Edit Search Mule Help                             |
|                                                                      |
|Welcome to GNU Emacs, one component of a Linux-based GNU system.     |
|                                                                      |
|                                                                      |
|                                                                      |
| ...                                                                  |
|                                                                      |
|---1:---F1  *scratch*         (Lisp Interaction)--L1--All-------------|
|For information about the GNU Project and its goals, type C-h C-p.    |
+----------------------------------------------------------------------+
```

NOTE: Emacs will usually fill the entire screen/window. I've shrunk the above example to save space here. You will also see a welcome message in Emacs when you first start it. I omitted that as well and substituted ``...'' instead. The welcome message simply identifies the exact version of Emacs you are using as well as pointing you to the on−line help and related items.

The Menu Bar

The topmost line of the Emacs interface is a menu. If you're running X, you'll recognize them as traditional pull−down menus that you can access using your mouse. Otherwise you'll need to use keyboard shortcuts (not covered here) for accessing the menus.

The Status Bar and Mini−buffer

Of the last two lines in the Emacs interface, the topmost one is essentially a status bar. It contains information about the buffer you're working in, which mode Emacs is in, and various other things. For now, just realize that it's there.

The bottommost line is called the **mini−buffer**. It is separated from the main buffer by the status bar we just discussed. You can think of the mini−buffer as the Emacs ``command−line''. It is where commands that you give Emacs appear and it is where status messages are printed in response to things you do.

You'll find that what I've called the status bar is usually referred to as the mode line in Emacs related documentation. It is where Emacs displays information about the current modes(s) you may be using as well as things like the current date and time, line number, file size, and almost anything else you might want to see there.

2.2 Some Terminology

This section covers the most basic of Emacs terminology that you'll encounter when using and reading about Emacs.

Buffers & Files

Unlike some editors, when you open a file in Emacs it does not stay ``open'' the entire time you're working with it. Instead, Emacs reads the file into a **buffer** in memory. While you're editing the buffer and working with the data nothing is changed on disk. Only when you actually save the buffer does the file on disk get updated. There are advantages and disadvantages to this approach but it is only important that you understand that it works this way.

As a consequence, you will see the term ``buffer'' used in Emacs documentation, modes, packages, and so on. Just realize that buffer means ``a copy of the file that is currently in memory.'' Oh, it's worth pointing out that a buffer doesn't always have to refer to a specific file on disk. Often times Emacs will create buffers as the result of commands you run. Such buffers may contain the result of the command, a list of selections to pick from, and so on.

Point & Region

In Emacs lingo, you'll often hear or see references to the **point**. In general terms the point is the cursor. The actual distinction between the point and cursor probably isn't important when you're first starting out with Emacs. But if you are curious, think about it this way. The cursor is the visual representation of the point. The cursor is always ``on'' a particular character position in the current buffer. The point, on the other hand, lives in the space *between characters* on in the buffer. So you might say that if the cursor is on the letter `h' in the word ``the'' then the point is between the `t' and the `h'.

Like many modern editors, Emacs allows to perform operations (indent, spell–check, reformat, cut, copy, paste, ...) on a section of the current buffer. You can highlight (or ``mark'') a block of text using the keyboard or mouse and then perform operations on just the selected block of text. In Emacs, that block of text is called a **region**.

Windows

Okay, this will be a bit confusing to anyone who has ever used a GUI interface before. Just remember that Emacs was developed long before GUI interfaces and window managers were popular.

A **window** in Emacs is an area of the screen in which a buffer is displayed. When Emacs is first started, you have one window on your screen. Some Emacs functions (such as the help and documentation) often [temporarily] open up additional windows in your Emacs screen.

Emacs windows have nothing to do with X windows in the GUI sense. You can open up additional X windows to display Emacs buffers, maybe to compare two files side by side. Those new X windows are referred to as **frames** in Emacs lingo. Read on.

Frames

In Emacs, a **frame** is a separate X window in which an Emacs buffer is displayed. But both are part of the same Emacs session. The behavior is somewhat (but not too much) like what happens if you hit Alt+N in Netscape Navigator.

2.3 Keyboard Basics

This section covers the basics of keyboarding for Emacs. Like every powerful editor, everything that you can do with Emacs is just a few keystrokes away.

If you're a `vi` user, the notion of using the `k, j, l, h` keys to move up a line, down a line, forward by a character, and backward by a character probably took some getting used to. In fact, it might have taken you a few hours or even weeks of practice before you could comfortably navigate a file using the various key combinations available in `vi`.

Emacs is no different. There are different keystrokes and commands to learn. Just like `vi`, you only need to master the basics to get a lot of work done. Then, as time goes on, you can slowly expand your knowledge and find faster ways of doing things.

Command Keys (Meta, Esc, Control, and Alt)

As you'll soon learn, Emacs makes heavy use of multi−key combinations. Because it is not a modal editor like `vi`, you don't have to think about being in ``command mode'' or ``editing mode'' before you can try to move the cursor or execute a command. Instead, you just press the right combination of keys and Emacs does as told (usually).

The keys that Emacs makes the most use of are usually abbreviated in the documentation as `C` (for Control or Ctrl) and `M` for (Meta). While most modern PC keyboards have one or more keys labeled `Ctrl` few have one labeled `Meta`. You'll want to mentally substitute either `Esc` or `Alt` for the Meta key. In most standard configurations, both Esc and Alt do essentially the same thing.

So when you see a reference in any Emacs related documentation to `C−x f` it means ``press control−x and then f.'' And if you see a reference to something like `M−x shell` is means ``press alt−x and type the word shell''.

A very useful command for beginners is `M−x apropos` or `C−h a`. apropos will search the Emacs on−line documentation for all functions and search for the regular expression you type. This is a great way to discover all commands related to frames. Simply `C-h a` and then `frame`.

Moving Around in a Buffer

Now that you know what all those fancy abbreviations mean, here's a list of the most common keystrokes for moving within a buffer:

```
Keystrokes  Action
-----------------------------------
C-p         Up one line
C-n         Down one line
C-f         Forward one character
C-b         Backward one character
C-a         Beginning of line
C-e         End of line
C-v         Down one page
M-v         Up one page
M-f         Forward one word
M-b         Backward one word
M-<         Beginning of buffer
M->         End of buffer
C-g         Quit current operation
-----------------------------------
```

And, as you might expect, the cursor keys (or arrow keys) usually work just as you'd expect. Your `Backspace` may not. That's another story. `:-(`

Essential Commands

Okay, now that you know how to move around within a buffer what about opening and saving files? Search? Here are some basic commands.

Before we jump straight to those commands, I need to briefly point out how this works.

All ``command keystrokes'' in Emacs (those that are `M-x something` or `C-something`) are actually just shortcuts to functions which are part of Emacs. You can call any of those functions by typing `M-x function-name` and hitting `Enter`. You can also use the keyboard shortcut for that function (if it has one).

For example, the Emacs function which saves a buffer to disk is called `save-buffer`. By default it is also bound to `C-x C-s`. So, you can either use they shortcut to save the current buffer, or you could type `M-x save-buffer` and achieve the exact same result.

All of the most common functions have keyboard shortcuts by default. Some of them are listed below.

```
Keystrokes  Function            Description
-------------------------------------------------------------------
C-x C-s     save-buffer         Save the current buffer to disk
C-x u       undo                Undo the last operation
C-c C-f     find-file           Open a file from disk
C-s         isearch-forward     Search forward for a string
C-r         isearch-backward    Search backward for a string
            replace-string      Search & replace for a string
            replace-regexp      Search & replace using regexp
C-h t       help-with-tutorial Use the interactive tutorial
C-h f       describe-function   Display help for a function
C-h v       describe-variable   Display help for a variable
C-h x       describe-key        Display what a key sequence does
C-h a       apropos             Search help for string/regexp
C-h F       view-emacs-FAQ      Display the Emacs FAQ
C-h i       info                Read the Emacs documentation
C-x r m     bookmark-set        Set a bookmark. Useful in searches
C-x r b     bookmark-jump       Jump to a bookmark.
-------------------------------------------------------------------
```

As you try many of those functions, you'll notice that many will prompt you for input. They will always to do in the mini–buffer. This is similar to using the `:` commands in `vi` or most commands that you'd use within your favorite Unix shell.

Emacs has literally hundreds of built–in functions available. The list above is a tiny sample that represents those that I use regularly. See the on–line help for a more complete listing of the available functions and more complete documentation on those I mentioned above.

Tab Completion

Like many popular Unix shells (bash, csh, tcsh, ...) Emacs offers command completion via the `Tab` key. In fact, the command completion in bash was modeled after that in Emacs, so if you use that feature in bash you'll be right at home.

As an example, try `M-x search` and then hit `Tab`. Emacs will append a hyphen to indicate that there are several possible completions but they all have a hyphen as the next character. Hit `Tab` once more and Emacs will display a list of the possible matches for you to choose from. Notice that it does so in a *new window*. It temporarily splits your display into two windows: one which contains the buffer you were editing and the other contains the list of possible completions for ``search–''. You may hit `C-g` to exit out of the selection process and close the new window.

2.4 Tutorial, Help, & Info

Emacs comes with an on–line tutorial which walks you through the basic editing features and functions that everyone should know. It also explains how to use the other help features in Emacs.

I highly recommend that you spend some time going through the tutorial if you plan on making a serious effort to learn Emacs. As shown in the table above, you can enter the tutorial via `C-h t`. The tutorial is self-guided and aimed at folks who are just getting started with Emacs.

If you are running Emacs in X, you will see that the rightmost menu on the menu bar is labeled Help. As you explore the Help menu notice that some items have keyboard shortcuts and those are listed right in the menu.

Finally, to see the volume of documentation available with Emacs, you should try `M-x info` or `C-h i` which launches Info, the Emacs documentation browser.

3.Emacs Modes

Emacs modes are different behaviors and features which you can turn on or off (or customize, of course) for use in different circumstances. Modes are what make one editor (Emacs) equally useful for writing documentation, programming in a variety of languages (C, C++, Perl, Python, Java, and many more), creating a home page, sending E-Mail, reading Usenet news, keeping track of your appointments, and even playing games.

Emacs modes are simply libraries of Lisp code that extend, modify, or enhance Emacs is some way.

3.1 Major vs. Minor Modes

There are fundamentally two types of modes available: Major and Minor. The distinction isn't the easiest thing to grasp until you've worked with a few of them off and on, but let's give it a shot.

Only one major mode can be active at a given time. Many minor modes can be active at a given time. Major modes tend to be language or task-specific, while minor modes are smaller and less specific utilities that cut across many tasks.

Sounds kind of abstract, so let's try an example. There's a mode that I use quite often when I'm writing plain old text files. It's called `text-mode`. This mode was designed for writing free form text like a README file. It understands how to identify words and paragraphs and generally makes sure that it does what I expect when I use the normal navigation keystrokes.

When I'm writing text for human consumpiton, I typically want it to look good. It should be properly word-wrapped to a reasonble value and so on. To enable word wrapping I just turn on the `auto-fill` minor mode. This mode tries to do the Right Thing when I'm typing along and hit the end of the line. The fact that it is a minor mode means that it can work with several different major modes. My notion of the ``Right Thing'' to do when I hit the end of the line is different when I'm in `text-mode` than it is when I'm in `java-mode` for example. I don't want my Java code to be word-wrapped as if was English text. But I *do* want the blocks of comments in my Java code to be word wrapped! `auto−fill` mode is smart enough to figure that out.

The authors of various Emacs modes have done a great job of making sure that things that should work as minor modes are minor modes.

If you look back at that ASCII sketch of an Emacs screen, you'll notice that the mode line identifies the mode(s) that Emacs is in. In that case it was in a mode called ``Lisp Interaction'' which is the default mode. It's really only useful if you're going to be writing Lisp code. (But since most of Emacs is written in Lisp, why not?)

3.2 Programming Modes

First and foremost, Emacs was designed by a programmer for programmers. There are high−quality modes available for almost every popular programming language you can think of (and even some not so popular ones). I only briefly describe a few of them here.

Most programming modes share some common characteristics. Usually, they'll do some or all of the following:

- Provide color−syntax highlighting for the language.
- Provide automatic indentation and code formatting for the language.
- Provide context (language) sensitive help.
- Automatically interface with your debugger.
- Add language−specific menus to the menu bar.

In addition, there are some non−language specific modes that help out with tasks that are common to programming in many languages. Things like interfacing to your version control software, automatically adding comments to your code, creating Makefiles, updating Change Logs and so on.

When you add all these modes together and consider the maturity and stability of the Emacs code, it compares quite nicely to commerically marketed Integrated Development Environments (IDEs) for languages like C++ and Java. And, of course, it's free.

C/C++/Java

Because the syntax of C, C++, and Java are quite similar, there is one Emacs mode which handles all three languages (as well as Objective−C and IDL). It's a very mature and complete package and it included in the Emacs distribution. This mode is called either `cc−mode` or `CC Mode`.

For more details or to download a newer version, visit http://www.python.org/emacs/.

Perl

There are actually two modes for editing Perl code in Emacs. The first is called `perl−mode` (as you would expect) and the second is `cperl−mode`. I don't have a good grasp of this history and why there are two modes (the docs don't say), but it would appear that `perl−mode` was the original mode for editing Perl code in Emacs. It seems to have fewer features than `cperl−mode` and is lacking the ability to recognize some of Perl's fancier langugage constructs.

Personally, I use and recommend `cperl−mode` which seems to be quite actively maintained and has just about every feature I could ever want. You can find the latest release here: ftp://ftp.math.ohio−state.edu/pub/users/ilya/emacs.

But don't take my word for it. Try them both and pick the one that best meets your needs.

Python

Python (another very popular scripting language) has an Emacs mode available for it as well. As far as I can tell, it is *not* distributed with GNU Emacs but it distributed with XEmacs. It works quite well in both editors, though.

You can get `python−mode` from the official Python web site http://www.python.org/emacs/python−mode/.

Others

There are many many other editing modes available to help out programmers. Such modes help out with things like:

- Shell Scripts (Bash, sh, ksh, csh, ...)
- Awk, Sed, Tcl, ...
- Makefiles
- Change Logs
- Documentation
- Debugging

And much more. See the last section of this document for more information on finding other modes and add−ins.

3.3 Authoring

Fancy Emacs modes are *not* limited to just those who write code. Folks writing documentation (of any sorts) can also benefit from a wide selection of Emacs modes.

Spell−Checking (`ispell` mode)

Authors of many types of documents need to spell−check once in a while. If you have **GNU ispell** installed, you can type `M−x ispell` and spell−check the current buffer. If ispell finds words that it doesn't know, it prompts you with a list of possible replacements and lets you select one (or none) of them. It's functionally equivelant to the spell−checkers in many popular non−free software packages.

HTML (`html−helper` mode)

If you find yourself writing HTML files once in a while (or even a lot), you might want to try out `html−helper−mode`. It is available from http://www.santafe.edu/~nelson/tools/ as is the documentation and related stuff.

As its name suggests, `html−helper−mode` provides lots of things to help out those folks who still write HTML by hand−−the old fashioned way.

TeX (`tex−mode`)

When you're writing documents in TeX, it's often helpful to get Emacs to add some color and highlight the backslashes, braces and other charcters. `tex−mode` takes care of that for you.

Though I don't write much directly in TeX anymore, when I did this mode proved to be quite helpful in making my TeX source a bit more readable.

SGML (`sgml−mode`)

The document you're now reading was written in SGML (and probably converted to the format you're reading it in). `sgml−mode` provides all the basics for SGML documents: validation, highlighting, forward−tag, backward−tag, and much more. It is a standard part of Emacs.

3.4 Other Modes

Of course, there are lots of other handy modes to make life easier. Here's just a sampling of the popular ones:

Version Control (`vc` mode)

`vc` mode interfaces with most of the popular version control back–ends (RCS, SCCS, CVS) to make it very easy to check files in and out, manage releases and so on. It is a standard part of Emacs and is documented in the Emacs documentation.

Shell Mode

Why switch to another X window or virtual console just to run a few shell commands? Do it from within Emacs and save yourself the trouble. `:-)`

`M-x shell` will launch a shell within an Emacs buffer. You can do most things with this buffer that you could do with a normal shell prompt (except for running full screen programs like `vi` or `pine`) because Emacs is talking to your real shell behind the scenes.

This is a standard part of Emacs, too, so you'll find it documented in the Emacs docs.

Telnet and FTP

Why switch to another X window or virtual console just to run telnet or FTP? Do it from within Emacs and save yourself the trouble. (Notice the pattern yet?)

Just like running a shell inside of Emacs, you can telnet and ftp. Try `M-x telnet` or `M-x ftp` to experience it for yourself. See the documentation for all the gory details.

Man

Why switch to another X window or virtual console just to read a manual page? Do it from within Emacs and save yourself the trouble. (I promise. I'll stop.)

Just like running a shell inside of Emacs, you can read manual pages. Try `M-x man` to experience it for yourself. See the documentation for more.

Ange−FTP

To quote the `ange−ftp` documentation:

> This package attempts to make accessing files and directories using FTP from within GNU Emacs as simple and transparent as possible. A subset of the common file−handling routines are extended to interact with FTP.

That means you can treat files on remote machines as if there were local. So if you need to edit a file on a different computer, just tell Emacs to open it (using a slightly different path syntax) and it takes care of all the details of logging in and retrieving the file. Then, when you save the file via `C−x C−s`, `ange−ftp` intercepts the save and writes the file back to the remote machine.

The slightly different path syntax goes like this... A file named ``myfile'', in a ``user'''s directory, on a machine named ``my.host.org'' can be opened by opening (`C−x f`) the file:

```
/user@my.host.org:~user/myfile
```

This, also, is a standard part of the Emacs distribution so you can find it documented in the Emacs documentation.

Thanks to Etienne Grossmann (etienne@anonimo.isr.ist.utl.pt) for the example above.

4.Customizing Emacs

Virtually all Emacs customization is done via Lisp code. You can modify variables which influence the way Emacs operates or you can add new functions to Emacs (or override existing functions−−replacing them with your own.

4.1 Temporary Customization

While experimenting with Emacs customization, you'll probably want to do it in a way that is temporary. If you do something horribly wrong, you can just `C−x C−c` to exit emacs and run it again. Once you've figured out what changes you'd like to make permenant, you can add them to your very own `.emacs` file so that they take effect every time you start Emacs. This is discussed in the next section.

Variable Assignments

The easiest customizations are accomplished by changing the value of a variable in Emacs. The list code to do this looks like this:

```
(setq variable-name new-value)
```

Where `variable-name` is the name of the variable and `new-value` is the value you'd like to give the variable. (In Lisp–speak, you're binding a variable to a value.) The `setq` function in lisp is analagous to the assignment operators (ususally =) in other programming languages.

NOTE: I'm glossing over many details here for the sake of simplicity. You may also see me or others use the Lisp functions `set` and even `setq-default`. If you're really curious, feel free to look them up in an Emacs Lisp reference.

Let's look at a line from my `.emacs` file

```
(setq-default transient-mark-mode t)
```

The variable `transient-mark-mode` controls whether or not a region becomes highlighted when I mark it. In many GUI applications, if you click and drag the mouse to select a range of text it becomes hilighted in reverse video or some other color. Emacs will do the same thing it the `transient-mark-mode` varible is set (to a non–nil value).

A *WHAT* value?

Okay. Brief digression. Most programming languages have some notion of true/false values. In C/C++ a value is considered true if it is a non–zero value. In Perl, a non–null or non–zero value is true. In Lisp, the same idea applies but the names and symbols are different.

True is usually written as `t` and false (or null) is written as `nil`. Like in other languages, though, any non–nill value is considered true.

To get the full description of what `transient-mark-mode` does, you can use the on–line help. Type `C-h v` or `M-x describe-variable` and then `transient-mark-mode`. If you're lazy like me, you can take advantage of variable name completion using the `Tab` key. Just type part of the variable name and hit the `Tab` key. If you've typed enough of it that Emacs can already uniquely identify it, you'll see the whole name completed for you.

Another variable that folks often set is `fill-column`. It tells Emacs how wide the screen should be for the purposes of word−wrapping (and `auto−fill−mode` respects this value). To set the value to something absurd, you could type:

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

But that won't actaully do anything. You need to tell Emacs to **evaluate** the expression you typed. To do so, put the point (cursor) at the end of the expression end then type `C−x C−e`, which calls the function `eval−last−sexp` in case you care. When you do that, noice that `20` (or whatever value you used) is echoed back to you in the mini−buffer at the bottom of the screen. That's just the return value from the expression you evaluated.

Just to prove that it works, type a sentence or two. If you happen to have `auto−fill−mode` enabled (you probably don't), you'll notice the text wrapping at the 20 column mark. Otherwise, after you've typed some stuff, type `M−q` which calls the function `fill−paragraph`. It will then perform the word wrapping.

File Associations

You can configure Emacs to automatically do something when you open a file of a particular type (just like some GUIs will automatically launch a specific application if you click on the icon for a particular file). For example, I may want Emacs to automatically switch to `text−mode` every time I open a file with a `.txt` extension. Well, that already happens. `:−)` So let's tell Emacs to always enter `text−mode` when you open a file named ``README''.

```
(setq auto-mode-alist (cons '("README" . text-mode) auto-mode-alist))
```

Huh?

Without diving into lots of Lisp programming that you really don't need to know (but it wouldn't hurt you to learn), let just say that the variable `auto−mode−alist` contains a list of pairs. Each pair contains a regular expression and an Emacs mode name. If a file you open matches the regular expression (in this case, the string `README`) Emacs starts the mode you specified.

The funny syntax above is because we're actaully adding another pair to that mode list. You wouldn't want to just assign to `auto−mode−alist` without making sure the values that it already contains aren't lost.

And if I wanted Emacs to automatically switch to `html−helper−mode` every time that I opened a file that ended with `.html` or `.htm`, I would add this to my .emacs file:

```
(setq auto-mode-alist (cons '("\\.html$" . html-helper-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.htm$" . html-helper-mode) auto-mode-alist))
```

The possibilities are truly endless.

4.2 Using a `.emacs` File

After you've spent some time with Emacs and have a basic idea of what customization can do for you, you'll probably want to customize a few things permanently (or at least until you change your mind). If you find yourself using Emacs on a daily basis, you'll also notice that your `.emacs` file get bigger as time goes on. That's a *Good Thing* because it means you've figured out how to make Emacs work the way **you** want it do work. It's a shame that more software products don't let you do that.

In case you haven't already guessed, every time you start Emacs, it looks for a file named `.emacs` in your home directory. Your `.emacs` file is where you should put any Lisp code that you want run automatiaclly and that includes the sort of customization we've been dealing with here.

Another example from my `.emacs` file:

```
(setq inhibit-startup-message t)
```

The `inhibit-startup-message` variable controls whether or not Emacs displays that welcome message when it starts. After a while, I got sick of looking at it (because I knew how to find the help and whatnot), so I went in search of a way to turn it off.

As an exercise, try creating a `.emacs` file of your own and add that line to it. Then exit and start Emacs again. You should not see the welcome message.

Often times when your read about an Emacs mode (or a package), the documentation will suggest some code to add to your `.emacs` file in order to make the mode or package work in a particular way.

The GNU Emacs FAQ (`C-h F`) contains some items related to `.emacs` files that you might find useful.

4.3 The Customize Package

As Emacs has grown in popularity and continuted to evolved, someone eventaully said ``there has to be a better way to let novice users customize their Emacs.'' And `customize` was born.

Customize provides a more intuitive method of customizing parts of Emacs. To try it out, either visit the `Customize` sub–menu in your `Help` menu, or type `M-x customize`.

Customize groups customization into logical groups like ``Editing'', ``Programming'', ``Files'', and so on. Some groups contain sub−groups.

If you make changes using the customize interface, Emacs will save the changes to your `.emacs` file. That's rather handy, because you can easily inspect (and change) the changes it made for you.

I don't use the Customize interface, so I can't say much more about it..

4.4 X Windows Display

Like any well behaved X application, Emacs respects your X resources. That means you can control the initial colors, geometry, and other X specfic things just as you could with an `xterm`, `nxterm`, or whatever.

Here's the relevant bit of my `~/.Xdefaults` file:

```
emacs*Background: DarkSlateGray
emacs*Foreground: Wheat
emacs*pointerColor: Orchid
emacs*cursorColor: Orchid
emacs*bitmapIcon: on
emacs*font: fixed
emacs.geometry: 80x25
```

See your `X` manual page for more details about X resources.

Chris Gray (cgray4@po−box.mcgill.ca) also notes:

> In Debian, the `~/.Xdefaults` doesn't seem to be used. However, Debian people can put what you have given in `/etc/X11/Xresources/emacs` and they can have the pretty colors that they had when they were using RedHat.

5.Popular Packages

In addition to the many different modes available for Emacs, there are also many add−on **packages**. I call them packages because they're more than just new modes. They often include extra utilities or are so large that calling them modes just doesn't seem to do them justice. In still other cases, they are software which extends or integrates other Emacs modes and packages. The distinction isn't entirely clear, but that's okay.

5.1 VM (Mail)

To quote the VM FAQ:

> VM (View Mail) is an Emacs subsystem that allows mail to be read and disposed of within Emacs. Commands exist to do the normal things expected of a mail user agent, such as generating replies, saving messages to folders, deleting messages and so on. There are other more advanced commands that do tasks like bursting and creating digests, message forwarding, and organizing message presentation according to various criteria.

When I first began using Emacs, I tried VM out for a while. I found it to be a great replacement for Pine, Elm, or most any other mail program. But I didn't want to use separate programs to read mail and news. VM is actively developed and well supported today.

It is available here: http://www.wonderworks.com/vm/.

5.2 Gnus (Mail and News)

To quote the GNUS Manual:

> Gnus is a message−reading laboratory. It will let you look at just about anything as if it were a newsgroup. You can read mail with it, you can browse directories with it, you can ftp with it−−−you can even read news with it!

> Gnus tries to empower people who read news the same way Emacs empowers people who edit text. Gnus sets no limits to what the user should be allowed to do. Users are encouraged to extend Gnus to make it behave like they want it to behave. A program should not control people; people should be empowered to do what they want by using (or abusing) the program.

GNUS is what I currently use for mail and news (as hinted above). GNUS is also actively developed and well supported today.

It is available here: http://www.gnus.org/.

5.3 BBDB (A rollodex)

BBDB is an Insidious Big Brother Database, a rollodex−like program for Emacs that works with most of the popular Emacs Mail packages (VM and GNUS included).

It is available here: http://pweb.netcom.com/~simmonmt/bbdb/index.html.

5.4 AucTeX (another TeX mode)

AucTeX is another mode for editing TeX files.

To quote the AucTeX web site:

> AUC TeX is an extensible package that supports writing and formatting TeX files for most variants of GNU Emacs. Many different macro packages are supported, including AMS TeX, LaTeX, and TeXinfo.

It is available here: http://sunsite.auc.dk/auctex/.

6.Other Resources

This section covers books, web sites, newsgroups, mailing lists, and other places you can find more information about Emacs.

6.1 Books

There are a a few really good books available for learning Emacs. In addition to these, you'll find that many Linux and Unix books also contain a chapter or two about Emacs (and `vi`).

Learning GNU Emacs

Authors: Debra Cameron, Bill Rosenblatt, Eric S. Raymond

Publisher: O'Reilly & Associates − http://www.ora.com/

You can buy it at a discount from Amazon.com via their Associates program:
http://www.amazon.com/exec/obidos/ASIN/1565921526/

Commentary: This is probably the best book to start with. After you've read the HOWTO and looked through the FAQ this book serves as a comprehensive and very approachable tutorial.

Writing GNU Emacs Extensions

Author: Bob Glickstein

Publisher: O'Reilly & Associates − http://www.ora.com/

You can buy it at a discount from Amazon.com via their Associates program:
http://www.amazon.com/exec/obidos/ASIN/1565922611/

Commentary: After you've used Emacs for a while and have decided that you'd like to try writing your own mode or maybe try out some advanced customization, this is the book for you. While it doesn't attempt to teach Lisp, it does contain a brief introduction to the language.

Programming in Emacs Lisp: An Introduction

Author: Robert J. Chassell

From the README file:

> This is an elementary introduction to programming in Emacs Lisp for people who are not programmers, and who are not necessarily interested in programming, but who do want to customize or extend their computing environment.

You can retrieve the manual in its entirety via anonymous FTP from the GNU FTP server:
ftp://prep.ai.mit.edu/gnu/emacs/.

You can buy a nicely printed version from Amazon.com via their Associates program:
http://www.amazon.com/exec/obidos/ASIN/1882114418/jeremydzawodny/.

Commentary: This a good introductory manual for Emacs Lisp−−even if you're not a heavy−duty programmer.

The GNU Emacs Lisp Reference Manual

Author: Richard Stallman

Publisher: The Free Software Foundation − http://www.fsf.org/

You can retrieve the manual in its entirety via anonymous FTP from the GNU FTP server: ftp://prep.ai.mit.edu/gnu/emacs/.

Commentary: This is the definitive guide to the Emacs Lisp programming language.

6.2 Web Sites

EMACSulation

EMACSulation is a column written by Eric Marsden that appears in the on−line magazine Linux Gazette located at http://www.linuxgazette.com/. The most recent column as of this writing is located at http://www.linuxgazette.com/issue39/marsden.html. Scan to the bottom of the article for links to previous ones.

6.3 Newsgroups

Search you local news feed for newsgroups which contain the string ``emacs'' and you'll probably find many. Those which my server carries are:

- comp.emacs
- comp.emacs.sources
- gnu.emacs
- gnu.emacs.bug
- gnu.emacs.help
- gnu.emacs.sources

6.4 Mailing Lists

The only mailing list devoted to Emacs that I know of right now is the NT−Emacs list. It is a list for folks who are using the Micro$oft Windows version of Emacs. See the NT−Emacs FAQ http://www.cs.washington.edu/homes/voelker/ntemacs.html for more. information.

6.5 The Emacs Lisp Archive

From the Emacs Lisp Archive README:

> The Emacs Lisp archives on ftp.cis.ohio−state.edu contain various pieces and packages of Emacs Lisp code. Emacs Lisp is the language used to extend the GNU Emacs editor published by the Free Software Foundation. Although much Emacs Lisp code is included in the GNU Emacs distribution, many people have written packages to interface with other systems, to better support editing the programming language they use, to add new features, or to change Emacs' default behavior. Most of the contents of this archive have been written by individuals and distributed publicly over the Internet through the info−emacs or info−gnu−emacs mailing lists or the comp.emacs, gnu.emacs, or gnu.emacs.sources newsgroups.

The archives are available via anonymous FTP from ftp://ftp.cis.ohio−state.edu/pub/emacs−lisp/.

NOTE: As far as I can tell, the Emacs Lisp Archive is slowly becoming out of date. I see very few new (or updated) packages appearing there, though I know they exist. They *do* get posted to the `comp.emacs.sources` newsgroup. (Feel free to correct me if this is wrong.)

7.Credits

The following people have contributed to the success of this document.

- Robert Vollmert rvollmer@gmx.net
- Larry Brasfield larrybr@seanet.com
- Etienne Grossmann etienne@anonimo.isr.ist.utl.pt
- Thomas Weinell kf6mli@amsat.org
- Adam C. Finnefrock adam@bigbro.biophys.cornell.edu
- Chris Gray cgray4@po−box.mcgill.ca
- Robert J. Chassell bob@rattlesnake.com
- Isaac To kkto@csis.hku.hk
- Matteo Valsasna valsasna@elet.polimi.it
- Tijs van Bakel smoke@casema.net
