Emacs Tutorial
Version 20

University of Waterloo

Version 2.1

Peter A. Buhr

January 10, 2000

Contents

1 Introduction

There are two editors in common use on UNIX[1] systems: vi and Emacs. There are versions of both these editors that run on personal computers. This tutorial describes Emacs, which is the easier of the two to use for both basic and advanced text entry.

Emacs is not just an editor. It provides a *desktop* environment so that it is unnecessary to end Emacs or use multiple windows to edit multiple files, compile and execute a program, or manage files. This tutorial starts with a brief introduction to the Emacs editor (version 20) and then progresses to advanced features in the Emacs environment. In total, it goes over approximately 40 commands, which are sufficient to accomplish most tasks. Emacs has its own interactive tutorial, which is discussed in Section 15. There are some books about Emacs [SBA92, Sta94, CRR96], but ensure the book covers at least Emacs Version 19 as there are considerable changes between versions 18 and 19.

Emacs is better than the traditional UNIX editor vi in at least two ways:

multiple undo: Emacs allows multiple editing changes to be undone. This capability results in a more relaxed and exploratory editing session because mistakes and changes can be easily undone.

windows: Emacs supports multiple windows even on tty terminals, i.e., terminals other than X-window terminals. For example, one window can have a program, another the results of the compilation of the program, and yet another, the output of the compiled program running in a shell. What is important is that Emacs operations can be easily and uniformly applied among theses windows; each window is not an island unto itself.

Emacs has many other interesting capabilities but these two produce a significant advantage over traditional editors.

This tutorial assumes a basic familiarity with the UNIX operating system and X-windows. Throughout the tutorial, the following symbols are used:

⇒ This symbol indicates that you are to perform the action marked by the arrow.

⋈ This symbol indicates that the section explains a concept that may be unfamiliar even if you have some previous editing experience. Make sure you understand this concept before advancing in the tutorial.

2 Before Starting

⇒ Copy the file /u/cssystems/emacs/.emacs to your home directory if it is not already there.[2]

This file contains Emacs initialization code to simplify Emacs usage and contains several options that you can modify to affect your Emacs environment. At the end of the tutorial, read through this file and adjust it to obtain the options you desire. As well, you can add new options in the future by extending this file.

3 Minimal Command Set

This section contains the absolute minimum you must know to manipulate a file with Emacs. The term file applies to any text based collection of data. Using the commands in this section, it is possible to create new or access existing files, add to or modify a file, and save the changes back to a file.

To edit a new or existing file start emacs, which has the following command syntax:

emacs *[file-name]*

⇒ Enter the command: emacs foo &

Emacs creates a new window, which may require positioning depending on the window manager used. In either case, the window or screen then has the general appearance of that in Figure 1. The menu and scroll bar only appear on X-window terminals; as well, the window manager may provide a border around the Emacs window and have a menu bar of its own. Note, the window manager is not Emacs, and so nothing in this tutorial applies to the window manager. As stated earlier, this tutorial assumes familiarity with an X-window manager.

The basic components of the root Emacs window are:

---

[1] UNIX was a Trademark of AT&T Bell Laboratories

[2] If you are not working on the undergraduate machines, you must also copy the file /u/cssystems/emacs/.emacs-20.elc, and edit the first line of the .emacs to reference it.

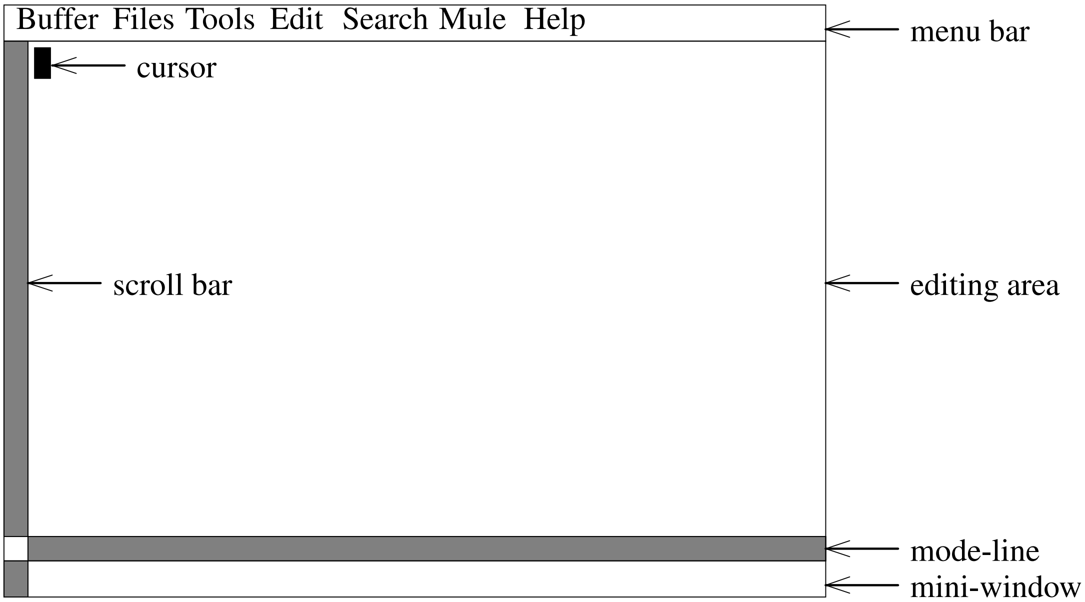

Figure 1: Basic Startup Display

editing area where the file is displayed for reading or editing purposes.

mini-window where prompt messages are displayed and user responses are entered for certain commands.

The editing area can be divided into multiple panes (discussed later). Each pane has administrative information at the bottom in a highlighted window, called the mode-line. In this section, there is only a single pane, and hence, one mode-line.

The file specified in the Emacs command is copied into the editing area if it exists; otherwise the editing area is blank for a new file (as is the case for file foo). In either case, the name of the file appears in the mode line.

⇒ Locate the file name foo in the mode-line.

When editing in Emacs, work is performed on a copy of the file, not the actual file. Emacs copies the file into a temporary area called a buffer. It does not modify the original file until the buffer is *saved*. Saving a buffer means writing its contents back into the file. Until a buffer is saved, the file remains unchanged.

The cursor, ▮, is in the editing area marking the location where additions and changes occur in the buffer. The standard keys—letters, digits, <Return>, punctuation, etc.—work as they do on a typewriter: characters or actions occur immediately in the editing area.

⇒ Watch the left side of the mode-line and type any character.
Two asterisks appear after the first character is typed to indicate that the buffer is now different from the file.
⇒ Type a few lines into the editing area. (It doesn't matter what you type.)
⇒ Use the cursor keys (◁ or ←, ▷ or →, △ or ↑, ▽ or ↓) to move the Emacs cursor around among the text.

Mistakes can be corrected by using the <Delete> key (⌫). The delete key removes the character under the cursor.

⇒ Delete several characters in different locations.

In general, holding a key down causes the key to repeat after a brief interval. For example, by holding down the delete key, entire words or lines are deleted.

To save the buffer:

⇒ Enter the command: C-x C-s

To type C-x, hold down the key labeled <Control> (or <Ctrl>) and press the x key. The <Ctrl> key is usually found somewhere to the left of the "a" key. The <Ctrl> key is like a shift key: it does nothing itself; it modifies the meaning of other keys that are pressed at the same time. So, to type C-x C-s, simply hold down the <Ctrl> key and press x and then s.[3] This command writes the current contents of the buffer to the file foo. Notice that the two asterisks in the mode-line are gone because the buffer and file are now the same.

To make sure that the save has really worked, quit emacs:

⇒ Enter the command: C-x C-c

The Emacs window or screen disappears, and the Shell prompt appears.

⇒ Enter the command: ls

The file name foo should be in the list of files.

In general, existing files are edited rather than creating new ones.

⇒ Enter the command: emacs foo & or emacs foo depending on the kind of terminal you are using.

The contents of file foo—whatever you saved in file foo—appears in the editing area.

While the cursor keys are sufficient to move anywhere, two additional movement commands are introduced.

- C-a moves the cursor to the beginning of the current line.
- C-e moves the cursor to the end of the current line.

⇒ Try moving to the beginning and end of a line in the editing area.

When you visit a large file, or add a lot of text to the buffer, not all the buffer fits in the editing area. Part of the text is invisible, even though it is still in the buffer. When the cursor is moved past the topmost visible line, the text is scrolled down so the next line is visible. Similarly, when the cursor is moved past the bottommost visible line, the text is scrolled up, displaying the next line.

It is possible to scroll by an entire window or screen of text rather than one line at a time using the commands:

- C-v moves forward through to the file, a screen full at a time.
- <Esc> v moves backward through to the file, a screen full at a time.

The Escape (or <Esc>) key is often on the upper left part of the keyboard. It is *not* a shift key; do *not* hold it down while you press v.

One of the most useful features that emacs provides is the "undo" facility. To undo a modification that you have just made to the buffer, type C-_. That is, hold down the <Ctrl> key and type underscore. On most keyboards, the underscore is a shifted key, so hold down both the shift and control keys, and press the underscore key. Repeated undos are possible in Emacs to undo a large number of changes.

⇒ Make a few changes in the buffer in different places and then undo them.

Accidents do happen such as typing the wrong command or a command you do not know about. The most common symptom is that the mini-window becomes active when you do not want it to be. You can recover from this by typing C-g, which beeps and terminates the current command. In some situations it may be necessary to type several C-g to get nested commands terminated (pause for 1-2 seconds between C-g commands). C-g can be entered at any time and it always resets back to the cursor in the editing area.

⇒ Enter the command: C-x C-c

[3] When using Emacs from a remote terminal there may be problems entering C-s. See Section 14 for a solution.

3.1 Summary

The commands in this section are the minimum needed to create and manipulate a file. This command set does not take advantage of facilities like a mouse or high-resolution X-window screen. The rest of this manual explains how to use the hardware facilities of an X-window terminal with Emacs, and more powerful editing and desktop features of Emacs. What is interesting is that the minimal set of commands is small and that Emacs can work on both new and old technology terminals.

COMMAND	**ACTION**
emacs *file-name*	edit a new or existing file
C-x C-s	save editing area into file
arrow keys	single character movement
delete key	remove single character
C-a, C-e	move to beginning or end of line
C-v, <Esc> v	scroll up and down in editing area
C-_	undo last change to editing area
C-g	abort current Emacs command
C-x C-c	quit Emacs and return to the Shell

4 Functionality Versus Interface

T.V. is style, radio is substance, Peter Gzowski

Before discussing more advanced features of Emacs, it is important to separate two important concepts. These concepts are functionality and interface. Functionality is the set of operations provided by an editor/desktop to display and effect change. The more powerful the functions and the more they can interact, the faster a user can perform required tasks. Functionality and integration are the measures by which any system should be judged. Interface is the mechanism that a user uses to invoke the functionality. An interface is largely (but not wholly) independent of the functionality.

Hence, in looking at a system it is important to differentiate between these two facets. This is because the best interface cannot compensate for lack of functionality, and poor functionality forces a user to produce, at best, awkward solutions or, at worse, no solution at all. Conversely, a poor interface is tolerable to access powerful functionality and integration, e.g., UNIX. The best solution is to have powerful functionality with a good interface. The problem is that people's tastes vary significantly with regard to an interface, and therefore, many different styles of interfaces have been designed and implemented for systems.

This tutorial covers the functionality needed to create, debug and test computer programs. The tutorial also covers four different ways that this functionality can be accessed in Emacs. While four different forms of interface may seem excessive, many people prefer different styles of interface and different kinds of interfaces work better for different skill levels. The tutorial asks that you try all 4 interfaces in the hope that you will find one that is appropriate for you. After the tutorial, you will probably adopt one particular interface and use it exclusively.

The first kind of interface is the most complex and the lowest level. This interface calls the low-level Emacs operations, mostly written in the programming language Lisp, to provide editing functionality. It is even possible to write your own Lisp routines to change and enhance the editing environment.

In general, invoking routines directly is a slow interface. Instead, Emacs provides the three other interfaces, which are:

- short command sequences entered solely from the basic QWERTY keyboard

  For example, the command sequence C-a invokes the Lisp routine beginning-of-line, which moves the cursor to the beginning of the current line. This interface is useful for experienced Emacs users who type quickly. These users do not like to move their hands off the basic QWERTY keyboard because, to do so, slows them down.

  Emacs has a predefined set of command sequences bound to routines, but a user is allowed to change the bindings, create new bindings to existing routines, and add new bindings to new routines. In this tutorial, some of the predefined bindings have been changed to deal with peculiarities in the MFCF system and to simplify certain common operations. All differences with standard Emacs are stated.

- function keys that exist around the basic QWERTY keyboard

  For example, function keys at the top and the numeric keypad on the right of the keyboard can be programmed to invoke Lisp routines. This interface is useful because it is possible to make the 15-25 most used operations (which comprise 90% of all operations invoked) available in a single keystroke. These users do not mind the additional hand movement to access operations and often label the function keys with the operation it performs.

- pull-down menus with command options

  For example, the menu bar contains headings for lists of commands that appear when the mouse cursor is placed over it and a mouse button is also pressed. Commands that appear can be selected and are invoked by their selection. This interface usually requires a high-resolution screen and a mouse to be effective. Furthermore, it requires large movements of the right or left hand away from the keyboard and good hand-eye coordination to select operations in the menus. This interface is good for beginners because speed in not a concern, the names on the menus act as a simple reminder, and the menus provide a mechanism to look for available functionality.

Multiple interfaces are fairly common; for example, many window systems provide both pull-down menus with keyboard accelerator commands. Unfortunately, in Emacs, none of these interfaces are complete in their own right, and therefore, an Emacs user needs to know a little of each.

5 Notation

The following table discusses the notation used in Emacs and in the remainder of this tutorial.

Notation	Meaning
<...>	means press the key labelled .... For example, <Esc> means press the key labelled Esc on the keyboard.
C-	means press the <Ctrl> key (control key) at the same time as the following character (two keys are pressed in parallel). For example, C-x means press the <Ctrl> key and the x key at the same time.
S-	means press the <Shift> key (shift key) at the same time as the following character (two keys are pressed in parallel). For example, S-<Bsp> means press the <Shift> key and the <Bsp> key at the same time.
M-	means one of two things depending on the style of keyboard. On keyboards without meta-shift keys (i.e., <Alt> or <Compose> or <Meta>), it means press the <Esc> key and then press the following character. For example, M-x means press the <Esc> key and then the x key (two keys are pressed sequentially). On keyboards with meta-shift keys, it means press the <Alt> or <Compose> or <Meta> key at the same time as the following character. For example, M-x means press the <Alt> or <Compose> or <Meta> key and the x key at the same time. Some keyboards have both the <Esc> key and meta-shift keys so either mechanism can be used.
M-C-	means one of two things depending on the style of keyboard. On keyboards without meta-shift keys (i.e., <Alt> or <Compose> or <Meta>), it means press the <Esc> key and then press the <Ctrl> key and the following character. For example, M-C-x means press the <Esc> key and then press the <Ctrl> key and the x key at the same time. On keyboards with meta-shift keys, it means press the <Alt> or <Compose> or <Meta> key and the <Ctrl> key at the same time as the following character. For example, M-C-x means press the <Alt> or <Compose> or <Meta> key and the <Ctrl> and the x key at the same time.
K-	means press the following character on the keypad (number pad), which is located in the bottom right corner of the keyboard. For example, K-2 means press the key labelled 2 on the keypad.

Finally, the notation for using the pull down menu is *menu-name/menu-item*. *menu-name* is the name that appears in the menu bar and it pulls down when buttoned on with any mouse button. *menu-item* is an item from the pull-down menu, which may itself have another pull down menu.

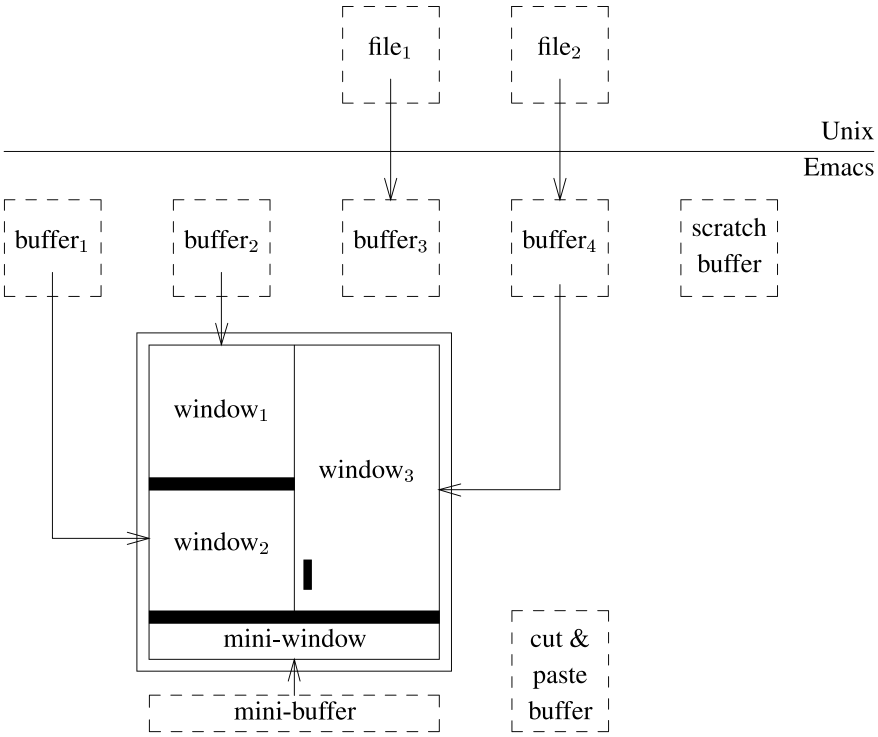

Figure 2: Emacs Components

6 ⋈ General Structure

There are three concepts that are used in Emacs: file, buffer and window and the relationship among these three components is illustrated in Figure 2.

1. A file is a UNIX file.
2. A buffer contains information in memory, such as a *copy* of a file or output from a program like the shell or a compiler.
3. A window is a section of the terminal screen that displays some portion of a buffer. Emacs windows are *not* the same as X windows; they are actually panes *within* a terminal screen or an X window. A window pane is tied to its base window and cannot be operated on independently of the base window, for example, if the base window is moved, the window panes move with it. Emacs tiles the terminal screen or X window with window panes; Emacs window panes never overlap one another. In this document, window panes are called windows to be consistent with Emacs documentation. All Emacs windows except for the mini-window are composed of two parts: the editing area and the mode-line located at the bottom of the window.

The window containing the Emacs cursor, ▮, is the *active* window (window$_3$ in Figure 2), and when you type, text is *always* interpreted by the buffer associated with that window. (The Emacs cursor is different from the X cursor, which is moved by the mouse.) A window displays as much of a buffer as possible; however, a window is usually not large enough to display all the information in a buffer. Buffer information may be clipped off the top, bottom, left and right sides of the window as the window is moved over the buffer (or the buffer is moved under the window depending on your point of view).

Just as all of a buffer cannot be displayed in a window, not all buffers can or need to be displayed in the emacs window or on the terminal screen. You may be editing several buffers, but only want certain ones displayed; the other buffers exist but are currently invisible. For example, buffer$_3$ represents a UNIX file, which is not currently displayed in any window. When needed, buffer$_3$ can be made visible by creating a new window for it or replacing an existing buffer in a window, e.g., replace buffer$_4$ in window$_3$ with buffer$_3$. An interesting consequence of this design is that a buffer may be visible in multiple windows, which allows different portions of a large buffer to be displayed simultaneously, e.g., the top of buffer_2 can be displayed in window_1 and the bottom of it displayed in window_2.

Buffer_1 and buffer_2 are displayed but not attached to a file. Buffers not attached to files contain temporary or transient information that **is not saved when Emacs terminates**.

The mini-buffer, which is displayed in the mini-window, is where Emacs prompts for input or prints the result of certain operations, like error messages, and it is not attached to a file because this information is transient. Essentially, the mini-buffer and mini-window are owned by Emacs for administration purposes.

The cut-and-paste buffer is for implicit storage and retrieval of text during modification of a buffer. Certain commands implicitly add characters to the cut-and-paste buffer (discussed later). Because the cut-and-paste buffer is not attached to a file, **its contents are not saved when Emacs terminates**.

When Emacs starts, it displays a buffer in the initial window called *scratch* (unless a *file-name* is specified after the emacs command). This buffer can be used as a scratch area for anything. For example, a section of a program can be removed and put in the *scratch* buffer, just in case it might be needed later during an editing session; hence, the *scratch* buffer can be used as an explicit cut-and-paste buffer (but **its contents are not save when Emacs terminates**).

7 Mode-Line

The mode-line looks something like this:

```
--:-- Emacs(cayley): *scratch*    Thu Sep 17 3:18PM   (Lisp Interaction)--L1--C0--All---------------
```

and is normally in reverse video, where reverse video depends on the particular terminal settings (black on white, white on black, underline, etc.). The mode-line contains general information and specific information about the window. The mode line has the following fields:

Status	System	Buffer	General	Mode	Positioning
--:--	Emacs(cayley):	*scratch*	Thu Sep 17 3:18PM	(Lisp Interaction)--	L1--C0--All---------------

Status: Indicates information about the status of the buffer displayed in the window.

--:-- means the buffer has not changed since it was created or last saved.

--:** means the buffer has been changed since the last save.

--:%% means the buffer is read only and cannot be changed.

It is common to undo changes until the buffer status changes from --:** to --:--, which means that the buffer is now the same as the file if this buffer contains a file.

System: Name of the program managing the buffer displayed in the window. In this case, Emacs is managing the buffer and it is running on cayley. Other programs that manage buffers are the mail reader and spelling checker.

Buffer Name: The name of the buffer displayed in the window. When the buffer contains a UNIX file, the name of the buffer is usually the same as the file. Buffer names surrounded by * are usually temporary buffers that are not attached to a UNIX file, like *scratch*.

General: Information like the time and date, system load, and mail arrival indicator may be displayed in the general area. The example mode line has no general information displayed.

Mode: Each buffer has an editing mode associated with it. An editing mode understands something about the information that is being edited. For example, when editing a C program, the buffer mode is C, which understands C programs so that the program can be indented automatically as it is being entered. Different modes provide different levels of understanding and different capabilities appropriate to the information being edited. Normally, modes are turned on automatically based on the UNIX file suffix. For example, editing a file with a .c suffix automatically sets the buffer mode to C.

Position: Indicates the absolute position of the Emacs cursor, e.g., L1 means line 1, C0 means column 0, and the relative position of the window in the buffer, e.g., 23% means the window is 23% from the top of the buffer or All meaning all of the buffer is visible.

8 Advanced Commands

This section discusses those commands that will advance you from a beginner to an intermediate Emacs user. The operations are grouped into management, movement, changing and miscellaneous. Each command is given by its Lisp routine name, followed by other interfaces that invoke the routine. Any Lisp routine can be invoked by typing M-x *routine-name* <Ret>. If arguments must be passed to the routine, they will be prompted for in the mini-buffer. Arguments can be specified directly, but the syntax used is Lisp, which is beyond the scope of this tutorial. The remainder of the tutorial assumes you are using an X-window terminal so that all the different interfaces can be discussed.

⇒ Enter: emacs &
Do not specify a file name.

8.1 Buffer Management

The following operations manage creation and deletion of buffers, and visibility of buffers in windows and movement among windows.

find-file-new-window (C-x C-f or <F7> or *Files/Open File...*) locates a UNIX file, creates a buffer for the file, and displays the buffer in a new window. You are prompted for a file name in the mini-window at the beginning of the operation. A path name is displayed in the mini-window, which can be edited and augmented to construct the path-name to the desired file. Typing <Spc> or <Tab> causes Emacs to try to complete a partially completed file name, or if there are ambiguous completions, it pops up a window and shows a list of the ambiguous names.

If you make a mistake typing, use the <Del> key to remove characters.

⇒ Enter: C-x C-f
Emacs moves the cursor into the mini-buffer, which looks like the following:

    Find file in new window: ~/▌

Emacs always begins the file name for you by starting with the UNIX path-name to the file associated with the current buffer. By default your home directory, ~/, is associated with the *scratch* buffer and therefore it is the starting path-name.

⇒ Enter: .<Spc> (the first character is a period)
A new window appears at the bottom of the screen showing all the files that start with a period. Notice the file .emacs, which you copied at the start of the tutorial.

⇒ Enter: e<Tab>
One of two things happens: either the file name .emacs is completed or a new window appears at the bottom of the screen showing files that start with the letter "e". If the latter occurs, type one more character in the name emacs and a <Tab> and repeat until the name .emacs is completed.

⇒ Enter: <Ret>
The lower window disappears and the buffer for file .emacs replaces the window containing the *scratch* buffer.

Always let Emacs do as much automatic completion as possible, so use the <Spc> and <Tab> to complete file names. In general, you only have to type one or two characters of the directory or file name at each level in a path-name and let Emacs do the rest. Try displaying the file /usr/include/sys/msg.h with minimal typing.

⇒ Pull down the *Files* menu and select *Open File...*
The prompt should look like Find file in new window: ~/▌

⇒ Enter: /u<Spc>
Gradually work along typing as little as possible to get to the desired file and then type <Ret>. The screen splits vertically and the buffer for the file is displayed.

Note, it is unnecessary to erase the path-name in the prompt to start a new path; simply start the new path with a ~ or /. Also, notice the buffer status for this file's buffer. What does this mean? Try to type some characters into the file. Now, look along the centre boundary of the two windows. Notice that some of the lines in the buffer for file .emacs are truncated because the window width is less than the line lengths. The truncated lines are marked with $.

other-window (C-x o (oh) or K-PF2) cycles through the windows on the screen (it does not cycle through the buffers). The order of the cycle depends on the order that the windows are created. On X-window terminals, the Emacs cursor can be moved directly to a window by moving the X cursor into the window and pressing the left mouse button.

- ⇒ Enter: C-x o
  Try the command several times.
- ⇒ If you are using an X-window terminal, move the Emacs cursor between windows using the X cursor.

It is possible to use the command other-window to move out of and back into the mini-window, when being prompted.

close-current-window (C-x 0 (zero) or K-PF3) removes the current window from the terminal screen and the remaining windows are re-tiled to absorb the empty space. The buffer that was associated with that window is still available, just invisible. When there is only one window, the buffer for that window is removed and the next buffer is displayed; hence, it is possible to cycle through the buffers one at a time.

- ⇒ Move the cursor into the buffer for file /usr/include/sys/msg.h and enter the command: C-x 0
  The window for that buffer disappears but the buffer is still there.
- ⇒ Enter: C-x 0
  The buffer for file .emacs disappears and the buffer for file /usr/include/sys/msg.h reappears.
- ⇒ Cycle through the buffers a few times until the buffer for file .emacs is visible.

find-buffer-new-window (C-x b or <F8> or *Buffers*) works like find-file-new-window, except it locates a particular buffer and displays the buffer in a new window.

- ⇒ Enter: C-x b
  The prompt: Find buffer: (default .emacs) appears in the mini-window.
- ⇒ Enter: ?
  A new window appears at the bottom of the screen showing all the buffers.
- ⇒ Enter: m<Tab>
  The buffer name is completed.
- ⇒ Enter: <Ret>
  The lower window disappears, the screen splits vertically, and the buffer is displayed.

save-buffer-and-check (C-x C-s or C-x C-\ or <F9> or *Files/Save Buffer*) writes the contents of the buffer associated with the current window back to the file on disk. A bell sounds when saving temporary buffers, *...*, to warn against accidentally saving a temporary buffer into an existing file. An example of this command is postponed to Section 8.3.

A common mistake when typing C-x C-s is to type C-x s. While the command C-x s is similar to C-x C-s, it performs a more complex operation; C-x s saves *all* buffers that are changed, while C-x C-s only saves the buffer associated with the current window. As well, C-x s prompts for the saving of each changed buffer, while the latter command does not prompt. So if you suddenly are prompted for saving a buffer, it is because you probably typed C-x s by mistake.

kill-buffer-or-window (C-x k or K-PF4) removes the current buffer and refills its window or removes the window if there is no buffer to fill it. If the buffer is associated with a file and it has been changed, you are asked if the buffer contents should be saved back to the file.

- ⇒ move the cursor to the .emacs buffer.
- ⇒ Enter: K-PF4
  The left window disappears, the screen is re-tiled, and the long lines in the msg.h buffer are now displayed.
- ⇒ Check the .emacs buffer is actually gone by pulling down the *Buffers* menu and verifying the buffer name ".emacs" is gone. Do not select any items from the list of available buffers.

8.2 Buffer Movement

Moving the cursor in a buffer is the most common operation. Emacs, therefore, has a large number of movement operations, some of which are described below. The move commands are organized by size and direction. The cursor can be moved forward or backward for any of the following size quantities:

character The arrow/cursor keys ($\lhd$ or $\leftarrow$, $\rhd$ or $\rightarrow$, $\triangle$ or $\uparrow$, $\triangledown$ or $\downarrow$) can be used to move the cursor one character at a time horizontally or one line at a time vertically. On X-window terminals, the Emacs cursor can be moved to the location of the X cursor by pressing the left mouse button. When the cursor moves beyond the top and bottom edge of a window, the window is scrolled one line up or down, so that the cursor is kept in view.

$\Rightarrow$ Use the cursor keys to move around the buffer for file msg.h
$\Rightarrow$ Move the cursor beyond the bottom of the window so the buffer scrolls.
$\Rightarrow$ Move the Emacs cursor around using the X cursor.
Notice, the Emacs cursor can only be placed where there are characters in the buffer.

word

- backward-word (M-b or K-4) moves one word left.
- forward-word (M-f or K-5) moves one word right.

$\Rightarrow$ Move the cursor right and left a word at a time.
Notice what constitutes a word.

line

- beginning-of-line (C-a or K-7) moves to the start of the current line.
- end-of-line (C-e or K-8) moves to the end of the current line.

$\Rightarrow$ Move the cursor back and forth to the beginning and end of a line.

window

- scroll-down (M-v or K-. or <Prev>) scrolls backward through (towards the beginning of) a buffer one window at a time.
- scroll-up (C-v or K-0 or <Next>) scrolls forward through (towards the end of) a buffer one window at a time.

$\Rightarrow$ Scroll the window to the end of the buffer and then to the beginning of the buffer.
Notice there is one line of continuity between windows when scrolling.

buffer

- beginning-of-buffer (M-< or <F5> or S-$\uparrow$) moves to the beginning of the current buffer.
- End-Of-Buffer (M-> or <F6> or S-$\downarrow$) operation moves to the end of the current buffer.

$\Rightarrow$ Move the cursor to the end and the beginning of the buffer.

scroll bar A scroll bar appears to the left of each window. The bar represents the total size of the buffer and the block in the bar indicates what fraction of the buffer is shown by the window for that buffer. By dragging the scroll block up and down the scroll bar, the window is moved over the buffer. There are three ways to drag the scroll block: discrete movements up or down one window at a time or smoothly dragging the block up or down the bar with the X cursor. If you decide not to use scroll bars, they can be turned off (see your .emacs file).

$\Rightarrow$ Move the X cursor directly over the scroll block for buffer msg.h.
$\Rightarrow$ Click the left mouse button to scroll down one window.
$\Rightarrow$ Click the right mouse button to scroll up one window.
$\Rightarrow$ Press the middle mouse button down and hold. Drag the scroll block up and down the bar.

goto-line (C-x @ or K-PF1) prompts in the mini-window for a line number and then puts the cursor at that line, with that line positioned in the middle line of the window. This command is useful when positioning to specific lines from information given in error messages.

⇒ Move the cursor directly to lines 45 and then 96.

isearch-forward-regexp (M-C-s or M-C-\ or K-1 or <Find> or *Search/Regexp Search...*)

- prompts in the mini-window for a search pattern (regular expression)
- search from the current cursor location, forward in the buffer, looking for a matching string
- when found, the window is positioned in the buffer at the current match and you must indicate what to do next using the following subcommands:

SUBCOMMAND	ACTION
C-s or C-\	continue search forward
C-r	continue search backward
<Ret>	exit search, leaving cursor at last location found

- when the end of buffer is reached, it is possible to continue the search at the beginning of the buffer by finding the next match (i.e., the search is circular)
- if the pattern matching characters * ^ ? $ [must appear as part the text in the search string, they must be prepended (escaped) with a \ character.

⇒ Enter: <F5>
which moves the Emacs cursor to the beginning of the buffer.
⇒ Enter: M-C-s
The prompt: Regexp I-search: appears in the mini-window.
⇒ Enter: msg segment
The cursor moves forward as you type characters matching text in the buffer.
⇒ Enter: C-s
The cursor moves forward to the next string that matches the pattern.
⇒ Continue entering C-s and watch what happens (do not stop when the terminal beeps)
⇒ Enter: C-r several times
This subcommand changes the direction of the search.
⇒ Enter: <Ret>
This stops the search, leaving the cursor at the last location matched.

Context searching is one of the fastest ways to locate text in a buffer. If you are not using context searching, you are not making efficient use of your time.

8.3 Buffer Changes

After moving to particular point in a buffer, changes can be made to the contents. The following are a number of operations for changing a buffer. The change commands are organized by size and divided into delete/kill and copy forms. The delete form changes the current buffer. The kill form changes both the current buffer and the cut-and-paste buffer. The copy form changes only the cut-and-paste buffer.

buffer region A region is defined by moving the cursor to a particular location in a buffer and marking that point, called the **mark**.

- set-mark-command (C-@ or <F1> or <Select>) marks the current cursor location as the beginning of a region.

Then the cursor can be moved forward or backward in the buffer to the location of the end of the region; hence, all text between the mark and Emacs cursor define the region. After defining a region:

- kill-region (C-w or <F3> or <Remove> or *Edit/Cut*) removes the region of text from the current buffer and placed it in the cut-and-paste buffer.
- copy-region-as-kill (M-w or <F2> or *Edit/Copy*) copies the region of text into the cut-and-paste buffer without removing it from the current buffer.

yank (C-y or K-Enter or <Insert Here> or *Edit/Paste Most Recent*) copies text from the cut-and-paste buffer into the current buffer starting at the cursor position. **NOTE:**

- The cut-and-paste buffer is emptied at the beginning of each change command if there is an intervening movement command.
- A series of change commands with **NO** intervening movement commands accumulates all of the changed text into the cut-and-paste buffer.

⇒ Enter: *Files/Open File...*
The prompt: Find file in new window: /usr/include/sys/ appears in the mini-window.
⇒ Enter: ~/junk <Ret>
Again, it is unnecessary to remove the path-name /usr/include/sys/, as long as you start with a ~ or /. A buffer for the new file junk appears.
⇒ Move back to the buffer msg.h
⇒ Move to the top and then the bottom of the buffer msg.h using the commands S-↑ then S-↓.
The beginning-of-buffer and End-Of-Buffer routines automatically set the mark at the previous cursor position before moving the cursor to the beginning or end of the buffer, respectively. Hence, going to the top and then the bottom, sets the mark at the top and the cursor at the bottom of the buffer.
⇒ Enter: M-w
The msg.h buffer is copied into the cut-and-paste buffer.
⇒ Move to the empty junk buffer.
⇒ Enter: *Edit/Paste Most Recent*
The contents of the cut-and-paste buffer are copied into the junk buffer.
⇒ Enter: C-x C-s
The buffer is written to disk.

character

- typed characters appear under the cursor and the cursor moves one position to the right.
- delete-backward-char (C-h or <Bsp>) removes the character to the left of the cursor but it is *not* put into the cut-and-paste buffer.
- kill-char (C-d or <Del> or ⌫) removes the character under the cursor and puts it into the cut-and-paste buffer.
- copy-char (C-o or K-3) copies a character into the cut-and-paste buffer without removing it from the current buffer.

⇒ Move to buffer msg.h and remove its window (just the window not the buffer)
The only buffer visible is junk and you can now change it.
⇒ Move to the top of buffer junk.
⇒ Try adding and deleting characters.
Do not worry about making a mess in the buffer.
⇒ Try killing and copying characters and yanking them back again (possibly in different places). Notice the status in the mode-line for buffer junk.

word

- kill-word (M-d or K-,) removes from the current cursor location to the end of the current word and places the text into the cut-and-paste buffer.
- copy-word (M-o or K-6) copies from the current cursor location to the end of the current word and places the text into the cut-and-paste buffer without removing it from the current buffer.

⇒ Try killing and copying words and yanking them back again (possibly in different places).

line

- kill-line (C-k or K- -) removes from the current cursor location to the end of the current line and places the text into the cut-and-paste buffer.
- copy-line (C-t or K-9) copies from the current cursor location to the end of the current line and places the text into the cut-and-paste buffer without removing it from the current buffer.

⇒ Try killing and copying lines and parts of lines and yanking them back again (possibly in different places).

query-replace-regexp (M-% or K-2)

- prompts in the mini-window for a matching pattern (regular expression) and a replacement string
- search from the current cursor location, forward in the buffer, looking for a matching string
- when found, the window is positioned in the buffer at the current match and you must indicate what to do next using the following subcommands:

SUBCOMMAND	**ACTION**
<Spc> or y	replace current match
<Del> or n	leave current match and move to next match
!	replace *all* remaining matches without prompting
<Ret> or q	exit replacement, leaving cursor at last location found

- when the end of buffer is reached, it is **NOT** possible to continue the replacement at the beginning of the buffer (i.e. the replacement is **NOT** circular)
- if the pattern matching characters * ^ ? $ [must appear as part the text in the match or replacement string, they must be prepended (escaped) with a \ character.

⇒ Enter: M-%
The prompt: Query replace regexp: appears in the mini-window for the string to be replaced
⇒ Enter: msg segment <Ret>
The prompt: with: appears in the mini-window for the new string.
⇒ Enter: fred <Ret>
The cursor moves to the first occurrence of the replacement string and stops.
⇒ Enter: y
The replacement string is changed to the new string.
⇒ Enter: n
The replacement string is *not* changed into the new string.
⇒ Replace all remaining occurrences of the replacement string with the new string.

Indentation Emacs indents most programming languages automatically based on the mode of the buffer. For example, the suffix .c informs Emacs that a file contains a C program and it is indented accordingly. Each indentation mechanism works differently for each programming language, but the following is usually true for each indentation mode.

- Indentation of the current line occurs when a <Ret> is entered at the completion of a line or by entering a <Tab> on an existing or partially completed line. If Emacs begins to indent strangely, it is because there is a missing bracket, semi-colon, quotation mark, comment delimiter or some such in the program.
- indent-region (M-i) re-indents a region of a program from the mark to the cursor location. The indent-region operation is used after a number of changes have been made to a region of a program, like removing a nested loop or if statement, to re-establish correct indentation. For example, after substantial changes to a subprogram, it can be selected as a region and all the statements in the subprogram can be re-indented.

undo (C-_ or <F4> or <Do>) undoes the effect of the last change. Multiple undos are possible, each one undoing the effect of the previous change. There is a limit to the number of undos, but this limit is rarely reached. If you undo past the point you want, type a space then continue pressing the undo key. The undos will then start to back up.

⇒ See if you can undo your changes to buffer junk so that the status for the buffer changes to -----.

8.4 Miscellaneous

The following operations perform frequently used but higher-level operations.

set-compile-command-and-compile (C-x c or <F10>) prompts for a compilation command and then executes the compilation command in a new buffer. The prompt for a compile command includes a possible compilation command that can be deleted or augmented. It is also possible to associate a specific compile command with a file by setting the compile-command variable in the file. For example to set the compile-command variable in a C program, put the following comment statements in the .c file (usually at the end):

/ Local Variables: */*
/ compile-command: "gcc -O -o fred fred.c" */*
/ End: */*

Each line must begin (and possibly end) with the comment delimiter of the language contained in the file so that the lines are ignored during parsing of the program. Emacs locates the setting of the compile-command variable when the file is loaded into a buffer and sets the variable with the value given in quotes. Hence, when the compile command is invoked for this buffer, the command is that given in the file.

Any error messages from the compilation appear in the compile buffer. There are two ways to locate the errors in the source program: use the subcommand C-x ` (back quote) in the compile buffer or use the goto-line operation in the source buffer.

shell-new-window (M-s or <F11>) creates a new window with a buffer running a shell. In this buffer, it is possible to execute a program after it is successfully compiled. There are the following subcommands:

SUBCOMMAND	ACTION
C-c C-d	generate an end of file
C-c C-c	interrupt the current shell command
C-c C-z	suspend the current shell command
<Tab>	file name completion on the current string

An Emacs shell is not an interactive shell and should not be used to execute commands that issue screen positioning commands. Emacs is in control of the terminal screen or X-window and does not allow a program executing in a shell window to issue positioning commands. Therefore commands like rn, man and more do not work correctly in an Emacs shell. News should be read outside of Emacs, manual information is read using M-x man <Ret> or *Help/Man...*, and examining a file is done by displaying it in a window.

⇒ Enter: M-s
⇒ Enter: ls -al
⇒ Enter: ls -al /usr/include/sys/msg.h
Use the <Tab> key to help type out the file name.

gdb (M-x gdb <Ret> or <F12>) starts a debugger running in a new window. Gdb has a large number of subcommands for debugging C programs (see man gdb).

help-for-help (M-h or <F1> or <Help> or *Help*) is used to find out information about Emacs. There are the following subcommands:

SUBCOMMAND	ACTION
b	display table of all key bindings
f	describe function
i	info documentation reader
k	describe command key sequence
v	describe variable
w	prints commands bound to a routine

home (C-x C-h or <Home>) vertically recentres the line containing the cursor to the middle of the window.

recenter (C-l) re-draws the entire screen and recentres the current buffer.

quoted-insert (C-q or C-∧) takes the next key press and inserts it in the buffer associated with the current window. This command allows characters that might ordinarily cause an action to occur to be inserted in a buffer. For example, to insert the character C-x in a buffer, enter C-q and then C-x.

ispell-word (M-$ or *Edit/Spell/Check Word*) checks the spelling of the word under the Emacs cursor.

⇒ Check the spelling of both words and non-words in the buffers.

suspend-emacs (C-z) suspends the current Emacs session and iconify the Emacs window. Deiconify the Emacs window to restart Emacs.

save-buffers-kill-emacs (C-x C-c or *Files/Exit Emacs*) terminates Emacs. If any buffers are associated with files and they have been changed, you will be asked if the buffer contents should be saved back to the file. As well, if there is a shell running you will be asked if it can be stopped.

⇒ Enter: C-x C-c

9 Directory Manipulation

As mentioned, Emacs provides desktop capabilities, such as manipulating and traversing directories, as well as manipulating the contents of files. Manipulating directories in Emacs replaces a number of shell commands. To edit a directory use the find-file-new-window, but type <Ret> after a completed directory name instead of a file name.

⇒ Enter: emacs &
⇒ Enter: F7
The prompt: Find file in new window: ~/▌ appears in the mini-window. (Due to an Emacs bug, it is sometimes necessary to remove the trailing / character.)
⇒ Enter: <Ret>
A new window appears showing all the files in your home directory, for example:

```
/u/student:
total 48
drwx------  4 student none  4096 Mar  6 23:46 .
drwx------ 10 student none  4096 Dec 22 15:00 ..
-rw-------  1 student none  4866 Dec 21 15:01 .cshrc
-rw-r--r--  1 student none 11722 Mar  7 20:39 .emacs
-rw-r-----  1 student none    72 Aug 10  1993 .login
drwxr-x---  2 student none  4096 Dec 22 15:00 bin
-rw-r-----  1 student none  1253 Aug 10  1993 junk
```

The output is identical to that from the shell ls command. There are 7 major columns representing: file protection, directory blocks, file owner, file group, file size (bytes), date of last change, file name. Text cannot be entered in a directory buffer; instead each character entered in the directory buffer is interpreted as a command to perform some operation on the directory (so be careful what you type). The target of a directory command depends on what line the cursor is positioned on; for example, if the Emacs cursor is positioned on the line ending with .login, a command applies to that file. There are a large number of directory commands, but the following are the most commonly used:

SUBCOMMAND	ACTION
f	find file
d	delete file
u	undelete file
x	execute deletes
g	update directory information
R	rename file
M	change protection flags

⇒ Position the Emacs cursor to the line for file .cshrc
⇒ Enter: f
The current directory window is replaced with a buffer for file .cshrc.
⇒ Move the Emacs cursor into the .cshrc buffer and scroll through the contents.
⇒ Delete the .cshrc buffer.
⇒ Position the Emacs cursor to the line for directory bin
⇒ Enter: f
The current directory window is replaced with a new buffer for the subdirectory (the previous directory buffer is not gone only hidden). Operations can now be applied on this directory. By doing an f command on directory .., it is also possible to move up the directory hierarchy.
⇒ Delete the bin buffer.
⇒ Position the Emacs cursor to the line for file junk
⇒ Enter: R
The prompt: Rename junk to: ~/█ appears in the mini-window.
⇒ Enter: fred <Ret>
The message: Move: 1 file appears in the mode line and the directory buffer is updated with the new file name Notice that the new file is no longer in alphabetical order.
⇒ Enter: g
The directory is reread and redisplayed in the buffer, which is the same doing an f command on the . (current) directory. The g command may be issued at any time to update the buffer to reflect the current directory contents.
⇒ Position the Emacs cursor to the line for file fred
⇒ Enter: d
A D appears at the beginning of the line for file fred indicating the file is maked for deletion.
⇒ Enter: x
The prompt: Delete fred (yes or no) appears in the mini-window.
⇒ Enter: yes <Ret>
The file is removed from the directory buffer list.

10 Manual Information

To read UNIX man information in Emacs enter M-x man <Ret> or *Help/Man...*, which then prompts for the name of the manual entry. Emacs creates a new window and buffer called *man ...* containing the manual information for .... It is possible to page forward and backward in this buffer to read and understand the information, further the manual buffer can be referenced at any time.

⇒ Enter: M-x man <Ret>
The prompt: Manual-entry: (default: xxx) █ appears in the mini-window.
⇒ Enter: ls <Ret>
The screen splits vertically and the buffer, *man ls* is displayed containing the manual entry for ls.
⇒ Move the Emacs cursor into the manual buffer and scroll through the contents.
⇒ Delete the manual buffer.

11 Mail

There are two commands for use with mail: one to send mail and one to read mail. To send mail enter C-x m, which creates a new window with a buffer called, *mail*, containing the following template:

To:
Subject:
--text follows this line--

Move the cursor to the appropriate lines filling in the necessary information. If a carbon copy is to be sent to others, create a new line after To:, start the line with Cc:, and type the list of users. When the message is complete, it is sent with the command C-c C-c.

⇒ send a message to yourself

Reading mail in Emacs is not compatible with reading mail using the UNIX mail command. Emacs mail is stored in a file called RMAIL, which is organized differently from the UNIX mail file. Therefore, if you decide to read mail in Emacs, you must read all mail in Emacs. If you are not going to read mail using Emacs, go to the next section.

To read existing mail messages enter M-x rmail <Ret>, which creates a new window with a buffer called RMAIL, and the last message received in displayed in the window. There are many subcommands available in the RMAIL buffer:

SUBCOMMAND	ACTION
<Spc>	next screen (same as Next Screen)
<Del>	previous screen (same as Prev Screen)
n	move to next non-deleted message
p	move to previous non-deleted message
>	move to the last message in RMAIL file
number j	jump to message specified by numeric position
d	delete this message, move to next nondeleted
u	undelete last deleted message prior to the current message
q	quit rmail: save undeleted messages to RMAIL
r	reply to this message
f	forward this message
h	print history list of all messages in RMAIL

⇒ watch the mode line until the word Mail appears
⇒ read the message
⇒ Enter: *Files/Exit Emacs*

12 Backups and Recovery

Emacs keeps an automatic backup of each file that is currently copied into a buffer. After every 100 key strokes in a window displaying a file, the buffer is automatically written to a backup file in the same directory as the original file. The name of the backup file is the actual file name surrounded by "#"s (e.g. #a1q1.c#). The backup file is erased each time the file is saved. If a backup file exists for a file when it is brought into a buffer, you are informed. The backup can be recovered by entering M-x recover-file <Ret>.

13 Getting the Most Out of a Terminal

All editors and especially Emacs work best given the largest possible editing area so that more information can be displayed. Therefore, make the Emacs window as large as possible.

14 Emacs Problems

There is a problem when the characters C-s and C-q are used to control the flow of characters to and from a terminal. C-s is a command to the low-level communications manager to stop sending characters to the terminal; C-q is a command that tells the communication manager to begin transmitting characters to the terminal again. Neither character is sent to Emacs. Unfortunately, there are several Emacs commands that use C-s and C-q. The solution is to remap the characters C-s and C-q to C-\ and C-^. Therefore, if Emacs documentation says to type C-s, type C-\ instead; similarly, type C-^ for C-q. However, this means that any commands associated with commands C-\ or C-^ are no longer available directly.

15 Further Information

If you would like to learn more about Emacs, there is an interactive tutorial available through the help command, M-h t or *Help/Emacs Tutorial*. However, be forewarned that some of the default command binds have been changed so some parts of the tutorial behave differently. There is also a large amount of online documentation, which is available through the help command, M-h i or *Help/Browse Manuals*. The information is organized in a tree structure that can be "walked" through, with more detailed information appearing at the lower levels of the tree.

References

[CRR96] Debra Cameron, Bill Rosenblat, and Eric Raymond. *Learning GNU Emacs*. O'Reilly and Associates, Inc., 2nd edition, 1996.

[SBA92] Michael A. Schoonover, John S. Bowie, and William R. Arnol. *GNU Emacs: UNIX Text Editing and Programming*. Hewlett-Packard Press Series. Addison-Wesley Publishing Company, Inc., 1992.

[Sta94] Richard Stallman. *GNU Emacs: Version 19*. Free Software Foundation, 1994.

Index

A Buffer Management

FUNCTION	KEYBOARD	FUNCTION KEYS
find-file-new-window	C-x C-f (replaces find-file)	<F7>
other-window	C-x o	K-PF2
close-current-window	C-x 0 (zero) (replaces delete-window)	K-PF3
find-buffer-new-window	C-x b (replaces find-buffer)	<F8>
save-buffer-and-check	C-x C-s or C-x C-\ (replaces save-buffer)	<F9>
kill-buffer-or-window	C-x k (replaces kill-buffer)	K-PF4

B Buffer Movement

FUNCTION	KEYBOARD	FUNCTION KEYS
backward-char	C-b	◁ or ←
forward-char	C-f	▷ or →
previous-line	C-p	△ or ↑
next-line	C-n	▽ or ↓
backward-word	M-b	K-4
forward-word	M-f	K-5
beginning-of-line	C-a	K-7
end-of-line	C-e	K-8
scroll-down	M-v	K-. or <Prev>
scroll-up	C-v	K-0 or <Next>
beginning-of-buffer	M-<	<F5>
End-Of-Buffer	M-> (replaces end-of-buffer)	<F6>
goto-line	C-x @	K-PF1
isearch-forward-regexp	M-C-s or M-C-\	<Find>
next match	C-s or C-\	
previous match	C-r	
stop search	<Esc>	

C Buffer Changes

FUNCTION	KEYBOARD	FUNCTION KEYS
delete-backward-char		<Del> or ⌫
kill-char (under cursor)	C-d or C-h	<Bsp>
copy-char	C-o (replaces open-line)	K-3
kill-word	M-d	K-,
copy-word	M-o	K-6
kill-line	C-k	K- -
copy-line	C-t (replaces transpose-chars)	K-9
set-mark-command	C-@	<F2> or <Select>
kill-region	C-w	<F3> or <Remove>
copy-region-as-kill	M-w	<F2>
query-replace-regexp	M-% (replaces query-replace)	K-1
replace match	<Spc> or y	
skip match	<Del> or n	
replace remaining matches	!	
stop replacement	<Esc> or q	
yank	C-y	K-Enter or <Insert Here>
indent-region	M-i (replaces tab-to-tab-stop)	
undo	C-_	<F5> or <Do>

D Miscellaneous

FUNCTION	KEYBOARD	FUNCTION KEYS
set-compile-command-and-compile	C-x c	<F10>
next error message	C-x ` (back quote)	
shell-new-window	M-s	<F11>
end of file	C-c C-d	
interrupt shell command	C-c C-c	
suspend shell command	C-c C-z	
file name completion	<Tab>	
gdb	M-x gdb <Ret>	<F12>
help-for-help	M-h (replaces mark-paragraph)	<Help>
key bindings	b	
function description	f	
info documentation reader	i	
command description	k	
variable description	v	
commands bound to a routine	w	
home	C-x C-h (replaces inverse-add-mode-abbrev)	<Home>
recenter	C-l	
quoted-insert	C-q or C-^	
ispell-word	M-$	
suspend-emacs	C-z	
save-buffers-kill-emacs	C-x C-c	

E Keypad

F1	F2	F3	F4
set mark	copy region	kill region	undo

F5	F6	F7	F8
beginning of buffer	end of buffer	find file new window	find buffer new window

F9	F10	F11	F12
save current buffer	compile current buffer	start shell new window	gdb

<table>
<tr><td>PF1
goto line</td><td>PF2
next window</td><td>PF3
close current window</td><td>PF4
kill buffer</td></tr>
<tr><td>7
beginning of line</td><td>8
end of line</td><td>9
copy line</td><td>-
kill line</td></tr>
<tr><td>4
backward word</td><td>5
forward word</td><td>6
copy word</td><td>,
kill word</td></tr>
<tr><td>1
search forward regexp</td><td>2
query replace regexp</td><td>3
copy character</td><td rowspan="2">Enter
yank</td></tr>
<tr><td colspan="2">0
scroll down</td><td>.
scroll up</td></tr>
</table>