Emacs on Athena

Emacs on Athena

On this page:

Overview

Learning how to use a text editor is an important skill you can acquire on any computer system, including Athena. You need to know an editor before you can use many other facilities on Athena, including formatting text and programming.

This document gets you started using the GNU Emacs text editor (Athena's primary supported editor), covering fundamentals of Emacs such as how to start and exit Emacs and how to edit files. It also covers advanced features that help you edit files efficiently and perform multiple tasks (not just text editing) within a single Emacs process. Since Emacs is primarily used for word processing, the emphasis in this documentation is on doing word processing quickly and efficiently, and getting through some of the more common trouble spots.

Emacs is a complex editor with a myriad of features. Choose different topics to learn during different sessions. No one uses every feature of Emacs.

This Emacs documentation assumes that you are familiar with concepts of working with files and directories, using the X Window System to manipulate windows on your workstation screen, and understanding special keys and control characters. For more help, type **help** at the Athena prompt.

The Consulting Office and Copy Tech also stock the **Emacs on Athena Reference Card** (AC-43), which lists most of the common Emacs commands.

The definitive reference for Emacs users is the **GNU Emacs Manual**. This book covers Emacs thoroughly, and is available for reference in the Athena Reference Sets in Hayden and Barker libraries, and on the Web.

Getting Started

Here, we tell you the commands which you need to run Emacs at a basic level.

Starting Emacs

To start the Emacs editor, you enter a form of the **emacs** command at the Athena prompt.

If you are using a workstation with the X Window System, including an & at the end of the line:

```
athena% emacs &
```

will allow you to continue working in that window. You can also start Emacs from the Text submenu of the Text/Graphics menu of the Dash menu bar.

If you don't have multiple windows (e.g., via dialup), don't include an & at the end of the line:

```
athena% emacs
```

When you enter the **emacs** command without any further specifications, the program gives you an empty workspace.

If you want to specify a file when you start Emacs, you can tell Emacs to automatically load one or more files by including the filenames in the command call:

```
athena% emacs filename [filename ...]
```

For example, you might enter:

```
athena% emacs paper.tex &
```

If the file you want to edit exists, but is not in your current directory, change directories or specify the pathname. If there is no file with that name, Emacs creates a new, empty file for you. (See Creating or Finding a File for more information.) If you specify two filenames, both buffers appear on the screen. Specifying more than two causes the last buffer named and a bufferlist to appear on the screen. (See Working with Multiple Buffers for more information on editing multiple files; see About Buffers for a definition of buffers.)

You do not have to start up a new Emacs session to edit another file. Many users find it convenient to keep Emacs running on their workstation (or terminal) and specify files as they need them. (See Creating or Finding a File for more information.)

You can specify the window dimensions and location on the screen with the **-geometry** flag, as with all X programs. The form is:

```
athena% emacs -geometry width_x height+ xoff+ yoff
```

The *width* and *height* parts of the geometry specification specify the dimensions of the Emacs window and are measured in characters (width) and lines (height). The *xoff* and *yoff* parts are measured in pixels and are used to specify the position of the window relative to the edges of the screen. You can specify positive or negative coordinates for location. For example, (0,0) is in the upper left corner of the screen, (-0,-0) is the lower right corner. Positive coordinates are measured from the left or top of the screen; negative coordinates are measured from the right or bottom. The example in the table below creates a small window in the upper right corner of the screen. Note that this only applies if you are using Emacs on a workstation.

To change the size later, use window manager commands. This is often more convenient than initially specifying the size.
Here are some examples of a few options to the **emacs** command:

Option	Example
multiple filenames	emacs Makefile main.c &
filenames with wildcards	emacs *.c &
size and position	emacs -geometry 80x24-0+0 &
reverse video	emacs -r &
start Emacs in the current xterm (not its own window)	emacs -nw
ignore ~/.emacs file	emacs -q &
set foreground color	emacs -fg green &
set background color	emacs -bg black &
combination of options	emacs -r paper.tex &

Remember not to use & if you are accessing Athena remotely (e.g., via dialup). The order and combination of multiple options does not usually matter. For information about other command line options, type **man emacs** at the Athena prompt.

Parts of the Emacs Window

The Emacs window is divided into four zones:

1. The **menu bar** is at the very top of the window. Here, Emacs lists possible options in a drag-down menu format. (Click on the menu which you wish to access; drag your mouse down to the option you want to choose, and let go. Many options (such as Buffers, Files, Tools, Edit, Search, and Help) are common to all modes. However, some modes have specialized options. (For instance, the HTML mode has an HTML option.) In this picture, the menu bar, shown at the top of the screen shot, has commands from Buffers to Help.
2. The **buffer editing area** starts below the menu bar and takes up most of the window's area. When you edit an existing file, its text appears here; if the file doesn't exist yet, and you haven't entered any text, this area shows the GNU Emacs "copyleft" notice, which disappears as soon as you do anything in the Emacs window. (See Customizing Your Emacs Environment for information on how to suppress this notice.) Here, this contains the beginning of the document that you are reading right now.
3. The **bar** – usually displayed in reverse video – is the Emacs **mode line**. Here Emacs displays status information about the editing session, such as the name of the file being edited and the modes that are in effect. (Modes are covered under Buffer Modes.) You can customize Emacs to display additional information in the bar, such as the current date and time. (See the sample ~/.emacs file in Customizing Your Emacs Environment.) The name of the file shown here is Emacs.html; you can also see that Emacs is in HTML mode (from the HTML in parentheses) and that the cursor is on line 1, the top of the document. If the cursor were further, this area would display a percentage.
4. The **minibuffer** is the area at the bottom of the screen beneath the mode line; the minibuffer is also known as the Emacs *echo area*. Whenever Emacs wants to tell you something, or ask you a question ("prompt you" for information), it does so here.

Using Emacs Commands

Command Syntax

Besides typing text into Emacs, you often perform tasks such as deleting text, moving the cursor to another position, or saving text to a file on disk. You accomplish these operations and many others by using Emacs *commands*.

These commands can be accessed by the menu bar, as mentioned above; they can also be typed in. Emacs must have a way of distinguishing what you type as text from what you enter as an Emacs command. This is accomplished through the use of *prefix characters.* A prefix character is a special keystroke or a sequence of keystrokes that changes the effect of a regular keyboard character.

Emacs uses several prefix characters:

Name	Abbreviation in Documents	Key(s)
Control	Ctrl-, C-, ^	Press and hold Ctrl
Meta	M-, Esc	Press/hold Alt, Compose, or black diamond; Press/release Esc; Ctrl-[
Esc	Esc	Esc
execute extended command	M-x	M-x

For the Meta command, sometimes the other keys don't work over dialup; **C-[** always works.

For example, to move the cursor to the beginning of the buffer you use the command **Meta-<** (press and hold the **Meta** key, then press the **<** key). Similarly, to move the cursor to the beginning of the line, you enter the Emacs command **Control-a**, (press and hold the **Ctrl** key, then press the **a** key).

All Emacs commands have names, such as **find-file** or **beginning-of-buffer**. Most of the common commands have shorter key bindings (e.g., **beginning-of-buffer** is invoked by **M-<**, mentioned above). Key bindings can be changed, but the command name always works when used with **M-x**; thus typing **M-x find-file** is the same as typing **C-x C-f**.

Notice that **C-x C-f** is different than **C-x f**. **C-x C-f** means to hold down the **Ctrl** key for both **x** and **f**. **C-x f** (**set-fill-column**) means to hold down the **Ctrl** key while pressing **x**, but release it before pressing **f**.

Use **C-u #** to tell Emacs to execute the following command # times. For example, **M-d** kills the word after the cursor, and **C-u 3 M-d** kills the next three words. The default numeric argument is 4, so **C-u M-d** kills the next four words. Similarly, **C-u C-u C-u M-d** kills the next sixty-four (4^3) words.

The Menu Bar

Many Emacs commands can be accessed through the Menu Bar in addition through command key sequences. When you are using a window system (ie. not over dialup and other text-only terminals), you can use the mouse to choose a commands from the Menu Bar as is standard in many windows-based programs. When you're first learning Emacs, the Menu Bar is a good way to get acquainted with some of the features without having to learn what keys are bound to what functions.

The menu bar can be toggled on and off with **M-x menu-bar-mode**

These are the menus available in the default (Fundamental) mode and brief descriptions of some of the commands available through them. Other menus are specific to certain modes and become available in those modes only.

- **Buffers:** The Buffers menu lists all the buffers currently in Emacs. Selecting one of those buffers switches you to that buffer. This menu also has a List All Buffers option which performs the same function as *M-x list-buffers*. (See the section on Working With Multiple Buffers for more information).
- **Files:** The Files menu has standard commands for opening, saving, closing, and inserting files, as well as Emacs frame and window management. The Exit Emacs command is also found in this menu.
- **Tools:** The Tools menu has commands for printing, version control systems, mail, news, compilation and debugging.
- **Edit:** The Edit menu commands include Undo, Cut, Copy, and Paste and allows the user to specify text properties such as faces, colors, justification, and indentation. The Spell sub-menu found here has options to select the language and check any or all of a given buffer.
- **Search:** The Search menu has commands for standard and regexp searching and query-replace searching within the buffer. The first set of commands let you specify the type of search and direction while the second set (Repeat Search) lets you find the next forward or backwards match in the buffer. (See the section on Searching For and Replacing Text for more information).
- **Mule:** MULE refers to "MULti-lingual Enhancement to GNU Emacs." Emacs supports a variety international character sets, and this menu allows you to set the language environment and input method. The details of how to use these character sets is beyond the scope of this document, but more information can be found in Chapter 22 of the **GNU Emacs Manual**.
- **Help:** Besides having an Emacs tutorial, the Help menu has a few useful sub-menus. The Options sub-menu allows the user to set things such as Global Font Lock mode (highlights syntax) and Auto Fill mode (wordwrap). The Manuals sub-menu has access to documentation including a FAQ, the Emacs manual (with ways to search it), and the Emacs man page. Finally, you can use the Describe sub-menu to access detailed information about all emacs commands. Describe Buffer Modes tells you all of the mode-specific commands available. Apropos lets you search for a specific pattern among all commands or variables. List Key Bindings gives a list of what key sequences are bound to what commands. Describe Key/Function/Variable gives a description of what the input key sequence, function, or variable does.

Working in the Minibuffer

The *minibuffer* is the line at the bottom of your screen where Emacs commands appear as they are typed in. When you enter the character sequence **C-x C-f**, for example, Emacs uses the minibuffer to prompt you for a filename to load into Emacs. Within the minibuffer, you can move around and edit. (In fact, you can use any Emacs command in the minibuffer that doesn't itself need the minibuffer.)

One of the features of the minibuffer is command and filename completion. This means you don't need to know or type the entire name of a command. You only need to type enough for the command to be unique. There are three commands for completion in the minibuffer:

Command	Action

Space	minibuffer-complete-word – completes as far as the end of the word (in a hyphenated command, a dash marks the end of a word) and shows possible completions.
Tab	minibuffer-complete – completes as far as possible and shows possible completions.
Return	minibuffer-complete-and-exit – completes the command as far as possible and then executes the command.
?	Show possible completions.

Sometimes, if you exit the minibuffer without completing the command you started running in it, Emacs starts keeping track of the commands you've left in the minibuffer. This isn't usually a problem, but can cause things to look cluttered and a lot more complicated than they really are. You can use **C-x o** to get to the minibuffer (see the section on Working with Multiple Windows), then get rid of the uncompleted command (by completing it, or by canceling with **C-g**. Try hitting **C-g** multiple times, if the first one does not work.)

You can find more information about the minibuffer and completion in Chapter 9 of the **GNU Emacs Manual**.

Interrupting and Undoing Commands

Emacs provides ways of undoing or cancelling commands:

- To *undo* the command you last entered, use **C-x u** (the **undo** command). As you keep typing **C-x u**, it undoes earlier and earlier operations. (The commands **C-?**, **C-Delete**, and **C-_** do the same thing on most workstations.)
- To *cancel* a command (not yet completed or currently underway), use **C-g** (the **keyboard-quit** command). This is the "get me out of here" command. Try this when Emacs doesn't seem to be responding. Look in the minibuffer. If Emacs is prompting you for information, **C-g** cancels the prompt. It may be necessary to press **C-g** several times; if you are totally lost, this is often an excellent action of last resort.
- To *refresh* the screen because it's become garbled, use **C-l** (the **recenter** command; note that it is the lowercase letter l). This doesn't affect any commands or your text; it cleans the screen from line noise so you can see what's really there. This is especially useful over dialup. Additionally, this works in X windows; its use is not limited to Emacs.

Moving the Cursor

You can move the input cursor around a buffer by using the mouse, the arrow keys, or keyboard commands:

- **Mouse:** The easiest way to move the cursor is with the mouse: move the mouse cursor to the desired location, then press the left mouse button to move the input cursor to that spot. This does not work over dialup.
- **Arrow keys:** Arrow keys move the cursor up, down, left, and right. On most keyboards, the arrow keys are to the right of the main keyboard. Play with the arrow keys to get used to their behavior, especially in moving from line to line. (Arrow keys may not work in a dialup session, but often this can be fixed. Use **olc** to ask the Consultants for help, or see the following table for alternatives.)
- **Keyboard commands:** Commands that move the cursor include the following:

Command	Cursor Action
C-f, right arrow, M-x forward-char	move right one character
C-b, left arrow, M-x backward-char	move left one character
M-f, M-x forward-word	move forward one word
M-b, M-x backward-word	move backward one word
C-p, up arrow, M-x previous-line	move up one line
C-n, down arrow, M-x next-line	move down one line
C-e, M-x end-of-line	jump to end of line
C-a, M-x beginning-of-line	jump to start of line
M-e, M-x forward-sentence	move to end of sentence
M-a, M-x backward-sentence	move to beginning of sentence
M-}, M-x forward-paragraph	move to end of paragraph
M-{, M-x backward-paragraph	move to beginning of paragraph
C-v, M-x scroll-up, PageDown (not over dialup)	move to next screenful of text
M-v, M-x scroll-down, PageUp (not over dialup)	move to previous screenful of text
M-<, M-x beginning-of-buffer	jump to start of buffer

M->, M-x end-of-buffer	jump to end of buffer
left mouse button	place Emacs input cursor at current mouse cursor position

Emacs considers a sentence to end wherever there is a period, question mark, or exclamation point followed by the end of a line or two spaces. Paragraphs are separated by blank lines and text formatter command lines; these lines are not part of any paragraph. Also, an indented line starts a new paragraph.

Saving Your Work

The **save-buffer** command writes the contents of the buffer into the file on disk. To enter the command, type **C-x C-s**. When Emacs performs the write operation, the editor displays a message in the Emacs echo area confirming the operation's success. Similarly, **C-x C-w** (write-buffer) saves the work under a different name, functioning similarly to the Save As commands of many word processing packages.

If you try to save the buffer and you haven't changed anything in the buffer since the last write, the command does not write the buffer to disk. Instead, it displays the message "(No changes need to be saved)"

Warning: *Save your work periodically!* In fact, you should use the command periodically, perhaps as often as every five or ten minutes. That way you are less likely to lose large amounts of work should the system happen to crash in the middle of your editing session. A good rule of thumb is this: if you've made a change to the file that you wouldn't want to have to make again, or you have written a sentence, paragraph, or series of paragraphs that you wouldn't want to reconstruct, save your work.

As you insert and edit text, Emacs keeps track of the changes and automatically saves them each time you enter or delete 300 characters (by default). It saves the changes in a separate *auto-save* file, not in the original file. The auto-save file provides another means of saving your work in the event of a system problem.

Each time Emacs saves changes, the editor displays the following message in the echo area:

```
Auto saving ... done
```

The auto-save file is stored on the particular workstation you are working on, in the directory /usr/tmp. The file remains on that workstation for three days or until you save the file, as long as some other user does not come along and remove it. (See Recovering Lost Files for information on how to restore your file from this auto-save file.)

To save changes you have made in multiple buffers in Emacs, use the **save-some-buffers** command. Type **C-x s**; you are asked to confirm saving each modified buffer.

Sometimes the **save-buffer** command cannot successfully write the buffer to disk. Usually when this happens, Emacs informs you with a beep and displays an error message in the minibuffer. The message says something like:

```
I/O Error
Opening output file: no space left on device
Cannot write to
filename; saving in %backup%
```

When this happens, try saving the buffer again. If you still have problems, you might be over quota. Also, confirm that Emacs is trying to save in a directory in which you have write access.

Note that whenever you edit a file, Emacs keeps a copy of the original version under the name *filename*~. You can use these backup files to undo all the changes you've made by renaming (moving) the backup file to the real filename:

```
athena% mv filename~ filename
```

This command overwrites whatever is in *filename*, so make sure this is what you want to do.

Exiting Emacs

The command to quit Emacs is **C-x C-c**, or **M-x save-buffers-kill-emacs**. If you have just used the **save-buffer** command to write your file to disk, the Emacs window vanishes and the editing session is finished.

However, if you have unsaved changes, Emacs displays a message in the minibuffer that asks if you want to save the changes you have made to that buffer. If you type **y**, Emacs writes the contents of the buffer to disk, then the Emacs window vanishes. If you type **n**, Emacs will say: "Modified buffers exist: exit? (yes or no)". You have to type in the entire word to get Emacs to do what you want here.

Emacs never finishes an editing session without first making sure that all the files you have worked on are saved, and it never kills a buffer without asking if you want to save it. You should always use the **safe-buffers-kill-emacs** command, or make sure all your buffers are saved, before logging out of your workstation; otherwise, you may lose your files.

Basic Editing Commands

Creating or Finding a File

You can edit an existing file or a new file by typing the **find-file** command, **C-x C-f**.

When you enter this command, Emacs displays a message in the minibuffer asking you to specify the name of the file you want to find. Enter a filename (including the full path to the file) and press **Return**. Be sure to check the pathname shown in the minibuffer; Emacs usually assumes the desired file is in the directory from which you started Emacs. Emacs creates a new buffer:

- If the file you specified does not exist, you are creating a new file. Emacs creates an empty buffer and places the input cursor at the top of the window.
- If the file you specified exists, Emacs copies the contents of the existing file into a new buffer and places the input cursor at the beginning of the buffer.

Inserting Text

You insert text into the buffer by typing. When you finish a line, press *Return or **Enter** to get to the beginning of the next line. Use the **arrow** keys or the cursor movement commands to position the cursor where you want it.
When you get to the bottom of the screen, Emacs adjusts the text within the window ("scrolls the screen up"), so that you have more visible space in which to type.

To change what you've typed, use the **Delete** key, just as you would when typing commands at the Athena prompt. A backslash appearing at the end of a line in Emacs indicates that the text is actually one line even though it looks like more than one. (When you "fill" the paragraph, as explained in the section Filling and Line Wrapping Within Margins, the backslashes disappear.)

If you type a closing parenthesis, Emacs hops the cursor over to the corresponding opening parenthesis briefly, then hops it back. This is because Emacs was written by programmers for whom matching them up is a major worry. Emacs also does this with paired brackets and braces.

Inserting Files

You can insert the text from another file into your current editing buffer. Do this by moving the input cursor to the location in the buffer where you want the inserted text to begin, then enter the **insert-file** command **C-x i**.

When you enter this command, Emacs prompts you in the minibuffer for the name of the file you want inserted. The message looks something like this:

```
Insert file: ~/
```

(The ~ character is shorthand for your home directory, /mit/ *username*.) Type the full pathname of the file (erasing what Emacs has provided if necessary) and press Return. The text of this file is entered into your current editing buffer.

About Buffers

When you use Emacs to create a new file, the editor sets up a *buffer* for you to work in. The buffer is a temporary workspace. The text you type into a buffer isn't permanent until you tell Emacs to write the buffer's contents to permanent storage ("write the buffer to disk"). In fact, the file isn't created until you write the buffer to disk for the first time.

When you use Emacs to edit an existing file, the editor also sets up a buffer for you to work in, and *copies* the file's contents into the buffer. Any editing you do changes the buffer's contents, not the file's contents. You change the file's contents only when you instruct Emacs to write the buffer to disk.

If you find a second file (with **C-x C-f**, among other ways), the first file remains inside Emacs. This way you can get quite a number of files inside Emacs, each in its own buffer. You can switch between them with the commands listed in Multiple Buffers and Windows.

Each buffer has a name; if the buffer corresponds to a file, it has a related name. Some buffers do not correspond to files. For example, the buffer named **Help** does not have any file. It is the buffer which contains help information when you request it with the **C-h** commands. Any text you see in an Emacs window has to be in some buffer.

The Scratch Buffer

Among any other buffers you may be using, Emacs always creates an empty *scratch buffer* for you. Emacs displays this scratch buffer if you start Emacs without specifying a file. In the Emacs mode line at the bottom of the Emacs window, the buffer is designated like this:

```
-----Emacs: scratch
```

The scratch buffer is a regular editing buffer: you can type text into it, move the cursor around, and so on. As its name implies, the scratch buffer is the Emacs version of a scratch pad. You can use it to practice things or type temporary notes, but in general you won't do real work in it. Instead, you should create a new file or edit an existing one by creating another editing buffer (see Creating or Finding a File).

If you have modified the scratch buffer and want to exit Emacs, the following message appears in the minibuffer:

```
Save buffer scratch? (y or n)
```

If you don't want to save the scratch buffer, enter **n**, then the window vanishes (all the text in the scratch buffer is lost) and your session is over.

If you do want to save the scratch buffer, type **y**. Emacs asks you to give a filename in which to write the contents of the buffer. Enter a filename, and press **Return**. The window vanishes, your session is over, and you have a new file containing the text from that scratch buffer.

Checking Spelling

The ispell command checks your spelling within Emacs. There are three ways to invoke ispell; **M-x ispell-word**, which runs ispell on the selected word; **M-x ispell-region**, which runs ispell on all words in the marked region in the current buffer; and **M-x ispell-buffer**, which runs ispell on the entire buffer. When an ispell command finds a word that is not in its dictionary, it responds with a message that says, Enter letter to replace word; space to flush. If there are "near misses" in the dictionary, ispell also displays the incorrect word and a list of alternatives (if any exist). At this point, you can type one of the following characters:

Command	Action
r	Replace Word; Ispell asks you for a replacement word.
#	Replace Word with Suggested Word; The suggested alternative spellings are numbered. If you use a word's number, ispell uses this word as a replacement.
Space	Skip Word; Leave this word as is, but stop at the next occurrence and prompt for an action.
a	Accept Word; Accept this word as correct for remainder of Emacs session.
i	Add Word; Add this word to your personal dictionary. This is a file in your home directory called .ispell.words which tells ispell to accept the word as correct every time you use ispell (useful for things like your name).
C-g	Quit Ispell.

You can also use ispell outside of Emacs; type **man ispell** at the Athena prompt for instructions on how to use it.

Filling and Line Wrapping within Margins

If you are inserting text into a paragraph, you can use the **M-q fill-paragraph** command to fix the margins on the paragraph, or you can wrap your lines automatically with auto-fill by typing **M-x auto-fill-mode**. The **auto-fill-mode** command is a toggle, so typing it again turns auto-fill back off. You can tell whether it's on or off by looking in the bar; the word Fill appears next to the major mode (e.g., Text or TeX) when auto-fill is on. Here are some commands related to filling text:

Command	Action
M-x auto-fill	Automatically break lines (at spaces) when they are longer than the desired width.
M-q, M-x fill-paragraph	Fill paragraph at or after point. With a numeric argument (i.e., preceded by C-u #), justify as well.
M-x fill-region	Fill each of the paragraphs in the region.
C-x f, M-x set-fill-column	With C-u as an argument, sets fill-column to the current horizontal position of point. With a numeric argument, uses that as the new fill column.
C-x ., M-x set-fill-prefix	Make the text to the left of the cursor a prefix for each line in a filled paragraph.

For example, if fill-column is 40 and you set the fill prefix to `;; ', then M-q in the following text

```
;; This is an
;; example of a paragraph
;; inside a Lisp-style comment.
```

produces this:

```
;; This is an example of a paragraph
;; inside a Lisp-style comment.
```

Other Text Manipulation Commands

In addition to the commands for checking the spelling of a file and for basic formatting of the text, Emacs offers other useful text-manipulation

commands. The following table summarizes some of these:

Command	Action
M-u, M-x upcase-word	Make word uppercase
M-l, M-x downcase-word	Make word lowercase
M-c, M-x capitalize word	Capitalize word
M-x capitalize-region	Capitalize every word in region
C-t, M-x transpose-chars	Transpose preceding two characters
M-t, M-x transpose-words	Transpose words around cursor
C-x C-t, M-x transpose-lines	Transpose current line and previous line
M-x sort-lines	Alphabetize the region by line

When changing the case of words, Emacs finds the first word on or after the cursor, ignoring any characters before the cursor and ignoring white space when necessary, operates on that word, and moves the cursor to the end of the word.

When transposing characters, Emacs exchanges the character before the cursor with the character on the cursor, moving the cursor to the position after both the characters. For example, to fix the typo teh, position the cursor on the h, then press **C-t**. The h and e switch places, and the cursor moves to the position after the.

Deleting, Moving, and Copying Text

Deleting and Killing Text

In Emacs parlance, *deleting* something **removes it forever**. *Killing* something means that it disappears from the buffer, but it **can be yanked back**. Here are some common commands related to eliminating text:

Command	Action
Delete, M-x delete-backward-char, BACKSPACE	Delete the character before the cursor.
C-d, M-x delete-char	Delete the character the cursor is on.
M-Delete, M-x backward-kill-word	Kill back to the beginning of a word.
M-d, M-x kill-word	Kill up to the end of a word.
C-k, M-x kill-line	Kill the line forward from the cursor.
C-k C-k	Kill the line and its trailing Return.
C-u 0 C-k	Kill the line to the left of the cursor.
M-k, M-x kill-sentence	Kill forward to the end of the sentence.
C-x Delete, M-x backward-kill-sentence	Kill back to the beginning of the sentence.

To yank back the last thing that you killed, enter **C-y**. (See Restoring Text: The Yank Commands for more information.) One way to make a copy of something is to kill it, then yank it back where it originally was, then yank again where you want the copy. (There are other ways to copy a piece of text.)

You can kill, delete, and restore large blocks of text by using the region-marking capabilities of Emacs. For more information, refer to the section on Killing Regions.

Marking Regions

A *region* is the area between *point* and *mark*. Point is the current cursor location. Mark is an invisible place marker that Emacs sets up. When you jump large distances in Emacs, such as with **M->** or **C-s**, you usually leave a mark behind. You can manually set a mark with the **set-mark** command, **C-@** (**Ctrl-Shift-2**). On many workstations, **C-Space** works as well. Alternatively, you can specify the region by simply dragging the mouse and highlighting the appropriate text.

Unfortunately, Emacs does not highlight the mark. The easiest way to find out its location is to use **C-x C-x**, which is **exchange-point-and-mark**. Typing the same thing again returns you to your initial position. (A few other commands for moving the mark are listed in Chapter 12 of the **GNU**

Emacs Manual.)

Killing Regions

Emacs has a number of commands for deleting, killing, and yanking individual characters, words, and sentences. Often, however, you want to delete, kill, or yank arbitrary regions of text.

To kill text with the mouse, press Mouse-1 at one end and Mouse-3 twice at the other end.

Here are a few of the more murderous commands, for killing almost anything you want:

Command	Action
C-w, M-x kill-region	Remove region and place in kill ring.
M-w, M-x copy-region-as-kill, Drag-Mouse-1	Copy region to kill ring without removing from this buffer.
C-M-w, M-x append-next-kill	Append the next region killed to the contents of the current kill ring element.

Whenever you kill text, it goes into the *kill ring.* This is a special type of buffer that can hold up to 30 (by default) different killed pieces. (There is only one kill ring within each Emacs process, so you can use it to carry text between buffers, where it can be yanked again.)

Normally, consecutive kills are part of the same segment of the kill ring. For example, if you use **C-k** several times in sequence, all those lines are kept together as one element in the kill ring, and **C-y** yanks them together. If you move the cursor or do something else between kills, but you want the pieces linked, use **C-M-w.** Editing commands from the X Windows system also work when on a workstation running X Windows.

Restoring Text: The Yank Commands

Once you have killed text into the kill ring, you can restore it any number of times. There are three commands for manipulating the kill ring in this manner: **C-y**, **M-x yank**, or **Mouse-2** inserts the text from current segment of kill ring into the buffer at the cursor; and **M-y**, or **M-x yank-pop**, replaces the last yanked text with text from the previous segment of kill ring.

When you type **C-y**, the last block of text that you killed is copied to the cursor location. It remains in the kill ring as well, so you can copy it back out as many times as you like.

M-y can only be used immediately after **C-y** or another **M-y**. It replaces the yanked text on the screen with text from the previous positions on the kill ring (i.e., earlier and earlier deletions).

For example, suppose you want to copy the following paragraph from the beginning of your file to replace another paragraph at the end:

```
The quick brown fox jumped over the lazy dog. However, it neglected to take into account the amount of rain that falls on plains in Spain.
```

To do this, you could:

1. Set the mark at the beginning with **C-**@, use **M-}** (or other cursor commands) to move the cursor to the end of the paragraph, then *copy* the whole region into the kill-ring using **M-w**. (See the sections on Moving the Cursor and Killing Regions for more on this.)
2. Move to the end of the file (with **M->**), to find the paragraph to delete:

   ```
   Foxes rarely venture into the drier, mountainous regions.
   This is despite the fact that hurricanes hardly ever happen in Hartford, Hereford, and Hampshire.
   ```

3. Delete this paragraph using the same method as before, except use **M-x delete-region** to *delete* this region.
4. Now type **C-y** to get back the paragraph about hurricanes (the most recent kill). Since we don't want that, we type **M-y**. This replaces that paragraph with the original paragraph about lazy dogs, as desired.

Working with Rectangles

You can kill and yank rectangular regions as well as arbitrary regions between the mark and cursor. To mark a rectangle, indicate its opposite corners with the mark and cursor. Rectangular cut text is stored in a different buffer from the normally cut text, so you use a special command to retrieve cut text. Here are some of the commands for working with rectangular text regions (there are others):

Command	Action
C-x r k, M-x kill-rectangle	Deletes a rectangular block of text between mark and point and places it in a special kill ring (you can remove columns from a larger block of text easily).
C-x r d, M-x delete-rectangle	Deletes a rectangular block of text between mark and point without storing it in a kill ring (this text is not recoverable).

C-x r y, M-x yank-rectangle	Restores last killed rectangle at cursor location, pushing text to the right of that column further over to the right.

Note that "killing" a rectangle is not killing in the usual sense; the rectangle is not stored in the kill ring, but in a special place that can only record the most recent rectangle killed. This is because yanking a rectangle is so different from yanking linear text that different yank commands have to be used and yank-popping is hard to make sense of.

Searching For and Replacing Text

Using Search Commands

When editing large documents, it becomes difficult to keep track of where everything is or to go to a specific place in the document. Emacs provides a variety of options for locating specific strings and substrings without having to scroll through the entire file.

There are two ways to search for a string in a buffer:

- *Incremental searches* prompt from the minibuffer asking what characters you wish to match. Each time you type in a character, Emacs moves forward or backward to the nearest match in the file.
- *Search-replace* asks you for two strings of characters (the *from-string* and the *to-string*), then interactively replaces selected occurrences of the *from-string* with the *to-string*.

For searching more than one buffer, or for searching long buffers quickly, you can use the **grep** and **tags** commands. The Emacs **grep** command is like the operating system **grep** command: it searches through a number of buffers for a string. The **tags** command is useful for finding locations of variables, subroutines, subroutine calls, and similar structures in C or Lisp programs.

Searching in a Buffer

Incremental searches prompt from the minibuffer asking what characters to match. Each time you type in a character, Emacs moves the point forward or backward to the nearest match in the file. The two search commands are C-s, or M-x isearch-forward, which starts or continues a forward search; and C-r, M-x isearch-backward, which starts or continues a backward search. When you use one of these commands, Emacs prompts you in the minibuffer for the characters you want. If you type **C-s** as the first character, Emacs searches for the string it searched for previously. (**C-s** may not work over dialup.)

Commands available in search mode are:

Command	Action
Delete	Undo the last keystroke. (Usually go to previous match.)
Return	Terminate the search and leave the cursor at the current location. Most other simple commands (**C-a**, **C-t**, etc.) exit the search and perform their function.
C-s	Go to next occurrence.
C-r	Go to previous occurrence.
C-q	Quote a character to be searched for.
C-w	Grab the word after the cursor as part of the search string.
C-y	Grab the line after the cursor as part of the search string.
C-g	Quit the search.

When the incremental search can't find any more examples of your string, it displays the line Failing I-search: <string>. Typing **C-s** again executes a *wrapped* search, looking through the file from the beginning to the end; **C-r** searches from the end to the beginning in the same manner.

Sometimes, **C-g** does not take you out of an incremental search completely. If you give the search command a string that Emacs can't find in your file, **C-g** changes the string to the largest substring it could find, but *keeps you in the search mode.* If you are in a wrapped search, **C-g** takes you back to a normal search. In any event, multiple uses of **C-g** quit the search, returning you to the point where you *started* the search. You can also use most of the cursor movement commands, such as the arrows, to exit a search procedure and stay at the point where you exited it.

Here is an example of an incremental search. Suppose you are editing the file input.c, and you wish to find all the instances of **scanf**:

```
/* small portion of input.c  */

  printf("Please enter your first name:  ");
  scanf("%f", &fname);

  printf("Please enter your last name:  ");
  scanf("%s",&lname);
```

Suppose the cursor is currently at the beginning of the *second* **printf** statement. You type **C-s** to start the search, then enter the letter **s**. The first thing found is the letter s in the word please. Since that isn't what you're looking for, try to enter **c**. If you type **v** by mistake (for example), use **Delete** to delete the **v**, and then enter **c**. Now you arrive at scanf.

Instead of typing the entire word for your search object, use **C-w** (once you're in search mode) to grab the word scanf into the search buffer. This copies it to the minibuffer without affecting the main buffer in any way. Use **C-r** to move to the earlier occurrences of scanf in that file. When you are where you want to be, hit **Return** to leave search mode and leave the cursor at that point.

Searching and Replacing (query-replace)

Sometimes you want to find a string so that you can replace it with a different one. Say you decide to change the name of a variable in a certain module, or perhaps you want to fix a typographical error you are used to making. Rather than going through "by hand" and possibly missing some, you can have Emacs do the work for you. The command for this is M-%, or M-x query-replace, which goes through all occurrences of from-string, and see if the user wants to change any of them to to-string. **M-%** asks you about each string you want replaced. If you want everything changed, type **!** at the resulting prompt. (Alternatively, you can use the M-x replace string command, which will do the same thing without querying.) You can also respond with any of the following:

Command	Action
Space, y	Replace the string and continue to next occurrence.
.	Replace this string, but then quit.
Delete, n	Don't replace, but continue to next occurrence.
Return, q	Don't replace, just quit (abort the query-replace).
!	Replace all remaining occurrences without asking.
^	Back up to previous match, but do not select it
C-h	Display these and all other options.

These might seem cryptic, but they are pretty straightforward. **Esc** aborts the **query-replace**, just like it does for search. **Delete** and **Space** both affect only the current choice. **C-h** lists your options at any level of the search.

You can do many other wonderful things with query-replace searches. See Section 13.7 of the **GNU Emacs Manual.**

Searching in Several Files Using grep

Sometimes you need to find which files contain a certain variable or word. The Emacs **grep** command searches more than one file or buffer, creating a second window called *compile* and running the operating system **grep** command. It behaves the same as the Emacs **compile** command, described in the section Compiling within Emacs. (For example, you can use **C-x `** to find the various matches.)

Grep stands for *General Regular Expression Pattern* matching, and searches for regular expressions or simple strings. To use it, type M-x grep. (The grep command is not limited exclusively to Emacs; you can use it anywhere on Athena as well.) The minibuffer prompts you for a pattern and one or more filenames (the files don't have to be loaded into Emacs):

```
Run grep (like this): grep -n
```

Specify the pattern followed by the filename(s) to search (wildcard characters are allowed in filenames):

```
pattern
file1
file2 ...
```

For instance, suppose you want to know which module contains variable indata. Use *.c to specify all the C source code files (providing you are in the proper directory). To do so, type:

```
M-x grep
Run grep (like this): grep -n
```

```
indata *.c
```

The -n argument shows matches and the line numbers on which they are found. You can use other arguments to **grep**, such as ~~**i to consider upper**~~ and lowercase letters indistinguishable. For more information on the **grep** command, type **man grep** at the Athena prompt.

Searching in Programs: Using Tag Tables

You can use *tags* to find all incidences of a variable or a string in a long program without taking the time to do an incremental search. Tags are a powerful tool for finding variables in C and Lisp programs quickly and efficiently. Tags are references telling Emacs where your variables, subroutines, and functions are defined, so Emacs can go directly to those locations without searching through the entire file.

To use tags, create a *tag table,* then use the Emacs tag commands to use it.

1. To create the table, use the command **etags** at the Athena prompt to include the source code files. This creates a file called TAGS containing the new tag table. For a C program, you could type something like:

   ```
   athena% etags *.c
   ```

   The next time you edit the file, the positions of the tagged items in the file will probably change. However, Emacs is intelligent enough to search out the new location, so you only really need to do this after major revisions, or when you add new functions.
2. Once you have the TAGS file, you can use it from within Emacs. The default file is TAGS in the current directory. To specify a different file, use the **M-x visit-tags-table** command. Once Emacs knows where to look, use the tag commands to locate various functions. Commands for this purpose are based on the period (**.**) and include:

Command	Action
M-., M-x find-tag	Asks for a tag, then moves you to its location. With a numeric argument, (i.e., when prefixed with **C-u** #) finds the next definition that fits the same substring.
C-x 4 ., M-x find-tag-other-window	Finds tag and places it in the other window.

Tags are covered more fully in Section 21.11 of the **GNU Emacs Manual.**

Multiple Buffers and Windows

Working with Multiple Buffers

Each time you load a file into Emacs, you create a buffer. There is one buffer for each file. Emacs also has its own buffers, like **scratch** and **Help**. You can use multiple buffers to work on several files simultaneously, which is very convenient for multifile programs and documents. Some of the commands for working with buffers are:

Command	Action
C-x C-f, M-x find-file	Read the specified file into a buffer of the same name.
C-x C-v, M-x find-alternate-file	Read the specified file into a buffer of the same name, and kill the current buffer.
C-x C-s, M-x save-buffer	Write the contents of this buffer back to the file.
C-x s, M-x save-some-buffers	Interactively save each modified buffer.
C-x C-w, M-x write file	Write the current buffer into a different file.
C-x C-b, M-x list-buffers	Show the name and status of the various buffers.
C-x b, M-x switch-to-buffer	Switch to a new buffer (default=previous).
C-x k, M-x kill-buffer	Delete a buffer (default=current). If the buffer has been modified, Emacs asks for confirmation before killing.
M-x bury-buffer	Place the current buffer on the bottom of the "stack," displaying the buffer underneath it (useful for cycling between many different buffers quickly).
M-~, M-x not-modified	Forget that the current buffer has been changed.

Use **C-x C-f** to read a file into Emacs.

- If a buffer doesn't exist with that name, a buffer with the same name as the file is created, and the file is read into that buffer.
- If a buffer already exists, you are moved into that buffer.
- If a different buffer with the same name as the file already exists (usually a file with the same name, but in a different directory), the new buffer is given the same name with a <2>, <3>, etc, appended to it. The actual file name is not changed.

Save the file using **C-x C-s** periodically and when you are finished. You can leave the buffer version of the file contents in Emacs when you move on to other tasks, or you can use **C-x k** to clear it out.

Use **C-x C-b** to show all the buffers currently in Emacs. The display looks something like this:

```
MR Buffer         Size  Mode      File
   -- ------         ----  ----      ----
   .* area.c         1403  C         /afs/athena/user/j/r/jruser/Cprogs/area.c
      intro.c        4063  C         /afs/athena/user/j/r/jruser/Cprogs/intro.c
    % Makefile       198   Text      /afs/athena/user/j/r/jruser/Cprogs/Makefile
    % Makefile<2>    201   Text      /afs/athena/user/j/r/jruser/Fortran/Makefile
      *scratch*      0     Fundamental
      *Buffer List*  193   Buffer Menu
```

For each buffer:

- If the buffer is the current buffer, it is marked by a period in the first column.
- An asterisk (*) under the R means the buffer has been modified, and a percent sign (%) under the R means the buffer is read-only. (These symbols also appear in the bar of the buffer itself.)
- The following two columns tell the name and size (in characters) of the buffer.
- The next column lists the major mode for that buffer. (Major modes are discussed in the section Major Modes, and also in Chapter 23 of the **GNU Emacs Manual**.)
- The final column is the name of the actual file the buffer corresponds to, if any. If the file name would make the column wider than your Emacs window, the end of the filename is replaced by a dollar sign ($).

In the example, the first four buffers are user files, **scratch** is a buffer kept by Emacs, and **Buffer List** is the buffer containing this listing. The buffer area.c has been modified, but none of the others have.

To go to a buffer from the buffer list, move the cursor to the line for that buffer and type **f** to go to that buffer. You can also use **C-x b** to switch to another buffer. Hit **Return** to go to the default buffer (usually the previous buffer) or type in the name of the buffer. You can use completion commands as usual, except that **Return** doesn't complete; it goes to whatever buffer is explicitly named, creating it if necessary.

For more information about buffers, consult Chapter 19 of the **GNU Emacs Manual.**

Working with Multiple Windows

Emacs *windows* are subdivisions of the actual X window that the Emacs process creates. Emacs windows are not the same as X windows. X windows create a whole new pseudo-terminal, and are created from the Athena prompt. They only work on workstations. Emacs windows are created from inside Emacs and simply partition the existing Emacs window. They work on all terminals and split the current Emacs window (screen), so you can view more than one buffer at time. You can create varying sizes and numbers of Emacs windows, and even split them horizontally so that you have side-by-side Emacs windows (though this is rarely done). Emacs often creates a second window when you use the **C-h** Help facility. Here are the basic commands for working with Emacs windows:

Command	Action
left mouse button	Place Emacs cursor at location of mouse (in whatever Emacs window that happens to be).
C-x o ("oh"), M-x other-window	Move cursor to other Emacs window.
C-x 0 (zero), M-x delete-window	Remove this Emacs window from screen (make the current Emacs window "nothing") and move to other Emacs window.
C-x 1 (one), M-x delete-other-windows	
Remove the other Emacs window from screen (make current Emacs window the only "one").	
C-x 2, M-x split-window-vertically	Split the Emacs window the cursor is in into two Emacs windows, one above the other.
C-x 3, M-x split-window-horizontally	Split the Emacs window the cursor is in into two Emacs windows, beside each other.
C-x 4 b, M-x switch-to-buffer-other-window	Create an Emacs window, then place specified buffer there.

C-x 4 f, M-x find-file-other-window	Create an Emacs window then do a find-file there.

The **C-x 4** commands select another window, splitting the current Emacs window into two windows if there is only one, prompt you for the file or buffer to put in the new window, and place the cursor there. Most help commands create a new Emacs window but leave the cursor where it was. The cursor location determines where text is entered and commands executed. You can switch between Emacs windows by using **C-x o** ("oh"). On a workstation, clicking the LEFT button moves the input cursor to the point that the mouse cursor is over, placing you in that Emacs window.

For more information about Emacs windows, see Chapter 20 of the **GNU Emacs Manual**.

Working with Surprise Buffers and Windows

One of the most powerful aspects of Emacs is that you can use it to edit more than one buffer at a time. Unfortunately, this aspect of the editor sometimes makes Emacs a very confusing program to work with. Sometimes the editor creates a second buffer for you without your consent.

For instance, when Emacs wants to show you something that is too long to fit in the single-line minibuffer, it puts the text into a new buffer and displays the new buffer by dividing the screen into two Emacs windows. Many Emacs commands do this, most notably the help command **C-h.**

Sometimes Emacs won't even bother to split the window into two buffer areas. Instead, the editor *overlays* your editing buffer area with the new buffer area. When this happens, it's natural to think that you have lost your file.

However, you haven't. Emacs never "throws away" a buffer without explicit instructions to do so. Your original editing buffer is still there, and the minibuffer displays a message about how to restore the previous buffer. This can always be accomplished by using the appropriate Emacs buffer or window command. These are covered in the sections Working with Multiple Buffers and Working with Multiple Windows.

The most important thing to notice is where the Emacs input cursor is. Return to your working buffer as follows:

- If your working buffer is showing on the screen, just make sure the input cursor is in that buffer window. If it isn't, then move it there using **C-x o** ("oh"), repeatedly if necessary. Then type **C-x 1** to return to one-window mode.
- If your working buffer isn't showing on the screen, you must get it on the screen. Type **C-x b** for **switch-to-buffer**. Emacs prompts:

  `switch to buffer: (default` *`buffername`*`)`

  If *buffername* is the name of your working buffer, press **Return**. If it isn't, type the name of your working buffer and press **Return**.
- If you can't remember the name of your working file, **C-x C-b** shows you a list of all the buffers. (This command splits the Emacs window in two and displays the list in the lower area.) Now type **C-x b** to switch to your working buffer (or type **f** or **Return** on the line in the buffer list).

Buffer Modes

Major Modes

The *major mode* of a buffer indicates the nature of the file it is editing. Each buffer has a single major mode associated with it. The mode is determined by the suffix on the filename, with the default mode being Fundamental. Fundamental mode has no mode-specific redefinitions or variable settings; each Emacs command and option uses the default behavior. Editing text in a human language (as opposed to a programming language) should be done in Text mode.

You can also invoke a major mode by typing **M-x *mode-name***. Some modes, like Buffer Menu, only work on special buffers. The current major mode is listed in parentheses in the bar.

Major modes can serve a wide variety of purposes. The mail mode allows you to read mail and use MH commands within Emacs. The picture mode lets you move to arbitrary points on the screen using the mouse and arrow keys. Each mode redefines certain keys and commands to behave in an appropriate manner. You can obtain a complete description of the current major mode by typing **C-h m**, which lists the mode-specific commands for that buffer.

There are far more major modes than can be discussed here. The special ones most often used are TeX mode and the various language modes. They provide convenient abbreviations for often-used commands and ways to format text more easily. Two of these are described below.

TeX Mode

You are placed in TeX mode anytime you have a filename ending in .tex. TeX mode is much like Text mode, with numerous time-saving features.

Some of the special features include the following:

- Emacs informs you of matches on the following pairs:

  `$ $ { } [ ] ( )`

- The **M-x validate-tex-buffer** command checks a buffer for paragraphs containing mismatched $'s or braces.

- Typing a double quotation mark, **"**, inserts two single left quotes, ``, when it seems to be the beginning of a quotation, and two single right quotes, **''**, when it appears to be the end. (Emacs inserts an actual double quotation mark only after a backslash.)
- The **C-c {** command inserts both the left and right curly braces, leaving your cursor between them.

For a complete listing of the special features of TeX major mode, type **C-h m** from an Emacs window in which you are editing a .tex file.

Editing Directories: Dired Mode

You can use your Emacs window to rename, copy, move, delete, and look at files in a directory by using the **C-x d** command. **C-x d** works similarly to **C-x C-f** (the **find-file** command) except that you give it the name of a directory. Emacs displays the current listing of all files in that directory, and enters Dired mode.

(You can also use **C-x C-f** to enter Dired mode, by trying to "find" a directory. If this doesn't work when you are trying to find the directory you are working in, delete the final slash in the file name.)

You can type **C-h m** to get a list of the special commands available in this mode. Some of the most useful include the following:

Command	Action
f	Load the file the cursor is on.
d	Flag the file the cursor is on for deletion.
~	Flag all backup (~) files for deletion.
R	Prompt for a new filename and rename the file.
C	Prompt for a new filename and copy the file to that name.
x	Remove all files flagged for deletion.
g	Update the displayed directory, removing all flags in the process.

Minor Modes

Unlike major modes, you can have as many or as few active *minor modes* as you want in each buffer, and each minor mode is active in the buffer only if you explicitly activate it. You can toggle each minor mode by giving its name as an argument to **M-x**. Active minor modes are listed on the mode line immediately after the current major mode. Some of the more useful minor modes are auto-fill-mode and abbrev-mode.

In auto-fill-mode, after you pass a certain column, inserting a space breaks the line at a previous space, "wrapping" the text. This is very useful in text and Fortran to prevent long lines. Abbrev-mode enables expansion of abbreviations (see the section Using Abbreviations).

For example, in a buffer that has C mode, auto-fill-mode, and abbrev-mode, the mode line looks something like this:

```
--**-Emacs: area.c (C Fill Abbrev)------20%-----------
```

A full description of minor modes is given in Section 28.1 of the **GNU Emacs Manual**.

Using Abbreviations: abbrev-mode

When you are programming, you should use long, descriptive names for your variables, so others can understand them; however, such names can be difficult to type. There are also a few words that are frequently misspelled or mistyped. To solve these, as well as other related problems, you can use *word abbreviations*.

Word abbreviations work by defining a short word or group of letters to stand for a much larger string. For example, you can set "mit" to stand for "the Massachusetts Institute of Technology." From then on, whenever you type **mit**, Emacs automatically replaces it with the longer string.

Defining Abbreviations

There are several ways to define abbreviations. Here are two:

- **Identify the expansion first, then name the abbreviation**. For instance, in the file calc.c, the variable surface_area_of_object is used a lot. To enter saoo as an abbreviation for surface_area_of_object, place the cursor after the word "object" in the buffer and type:

  C-u 4 C-x a g saoo

  C-u 4 indicates that the abbreviation refers to the previous four words (ignoring punctuation).
- **Enter the abbreviation first, then then define the expansion**. This avoids the necessity of having to count the number of words, and helps when you are abbreviating words with punctuation inside of them. To do the same thing as in our previous example, place the

cursor after the abbreviation in the buffer and type:
C-x a i g surface_area_of_object

Using Abbreviations

Once you have defined your abbreviations, you must be in abbrev-mode to use them; enter the mode by typing **M-x abbrev-mode**. In abbrev-mode, Emacs automatically expands your abbreviations when you type them. (You can use the **C-x '** command to expand the abbreviation before the input cursor if you're not in abbrev-mode, or to expand an abbreviation you typed before entering abbrev-mode.) If you have no further need of abbreviations at all, type **M-x abbrev-mode** again to turn automatic expansion off.

Abbreviations are very useful to automatically correct common misspellings. Let's say you commonly mistype "the" as "teh." You can fix this by entering abbrev-mode, and defining "teh" as an abbreviation for "the." Now whenever you type **teh** Emacs will automatically correct it. (Be careful about doing this with common letter clusters, or you could end up in more trouble than you started with.)

Use **M-x unexpand-abbrev** to undo the most recently defined abbreviation. For example, if you define "mit" as the longer string given above, Emacs dutifully turns the letters "mit" in words like "admit" and "submit" into their expanded versions, which may not be your intention.

To list what abbreviations you have defined, use **M-x edit-abbrevs**. This produces a listing of the various abbreviations in a new buffer, and allows you to modify them. Use **C-c C-c** to save your modifications, then **C-x b** to return to the buffer you were previously editing.

To change the definition of an abbrev, just define a new definition. When the abbrev has a prior definition, the abbrev definition commands ask for confirmation for replacing it. To remove an abbrev definition, give a negative argument to the abbrev definition command:

C-u - C-x a g

M-x kill-all-abbrevs removes all the abbrev definitions there are. Here is a listing of the most useful abbreviation commands:

Command	Action
C-x a g, M-x add-global-abbrev	Define an abbreviation for word before the cursor location. Numeric argument specifies how many words to be abbreviated; you are prompted for the abbreviation.
C-x a i g, M-x inverse-add-global-abbrev	This prompts you to specify an expansion for the abbreviation before the cursor location.
M-x unexpand-abbrev	Undo last abbreviation expansion.
C-x ', M-x expand-abbrev	Expand an abbreviation even if not in abbrev-mode.
M-x edit-abbrevs	Show abbreviation table, and allow you to modify it.
M-x read-abbrev-file	Read in a file of abbreviations.

The sample ~/.emacs file in the section Making Changes Permanent Using Your .emacs File has an example of how to read in abbreviation files and set abbrev-mode when in text-mode. For more information about abbreviations, see Chapter 28 of the **GNU Emacs Manual**.

Customizing Your Emacs Environment

Emacs Variables

Human nature being what it is, you can never please all the people all of the time. That's why Emacs has variables that you can change to suit your own tastes. There are really only two commands you need to know for using variables, with a third for the curious:

Command	Action
M-x list-options	Show all user-accessible variables, with value and description.
M-x set-variable	Enter the name of a variable and set it to specified value (for this Emacs session only).
C-h v, M-x describe-variable	Give the documentation on specified variable.

There are many uses for variables. The **list-options** command has a complete list of anything you are likely to use, so it is rather long. Here are just a few, with the default value shown in brackets:

Variable	Description
c-indent-level [2]	For each open brace in C mode, number of additional spaces to indent next line.
fill-column [70]	In auto-fill-mode, the column after which a word is broken.

kill-ring-max [30]	The number of different regions saved in the kill ring buffer.
lpr-command ["lpr"]	Shell command for printing a file.

There are too many variables to list here. To find a variable relating to a specific task, use the **C-h a** command. For more information about how to manipulate variables, see Section 28.2 of the **GNU Emacs Manual**.

Making Changes Permanent Using Your .emacs File

With Emacs variables and keyboard macros (covered under Advanced Techniques), there are many alterations you can make in your Emacs environment. Instead of trying to type in all those variables and definitions each time you need them, you can create a file called ~/.emacs in your home directory that is read each time you start Emacs.

This file is quite powerful, so be careful not to do anything that radically changes your Emacs. If something goes wrong, use the **-q** command line option with the **emacs** command to enter Emacs without reading the ~/.emacs file.

The ~/.emacs file is written in Emacs Lisp, which makes it simple to customize your Emacs if you know Lisp. Looking at other people's ~/.emacs files is probably the best way to learn more about them. Here is a sample ~/.emacs file:

```
;;; Read in the file of word abbreviations
  ;;; If the file does not exist, this command will not work!
  (read-abbrev-file abbrev-file-name)     ;Standard Abbreviations
                                          ;Default=~/.abbrev-defs

  ;;; Access one emacs window from multiple programs (use emacsclient)
  (server-start)

  ;;; Prevent appearance of copyleft on Emacs startup
  (setq inhibit-startup-message t)

   ;;; Don't need to check completions with these endings
  ;;; in find-file and similar commands
  (setq completion-ignored-extensions
        (append completion-ignored-extensions
                (quote
                 (".dvi"  ".PS" ".aux" ".otl" ".bak" ".err"  ".o"))))

  ;;;       Set Major Mode Hooks

  ;;; Set mode according to filename extension
  (setq auto-mode-alist
                       '(("\\.text$" . text-mode)
                         ("\\.c$" . c-mode)
                         ("\\.h$" . c-mode)
                         ("\\.tex$" . TeX-mode)
                         ("\\.el$" . emacs-lisp-mode)
                         ("\\.scm$" . scheme-mode)
                         ("\\.l$" . lisp-mode)
                         ("\\.lisp$" . lisp-mode)
                         ("\\.f$" . fortran-mode)))

  ;;; If no other mode indicated, use Text as Major Mode
  (setq default-major-mode 'text-mode)    ;default mode

  ;;; Set auto-fill and abbreviation for Text Mode
  (setq text-mode-hook
        '(lambda nil
           (setq fill-column 72)
           (auto-fill-mode 1)
           (abbrev-mode 1)))

  ;;; Set abbreviations and customized indenting for C Mode
  (setq c-mode-hook
        '(lambda nil (progn
           (setq c-brace-offset 2)
           (setq c-auto-newline 1)
           (abbrev-mode 1))))
```

Emacs Lisp is very similar to Common Lisp, using parentheses as delimiters and semicolons (;) as comment indicators. There are a few other things you should note in this file:

- You use **setq** to set Emacs variables (see the section Emacs Variables):

  ```
  (setq
  variable
  value)
  ```

- Emacs allows you to specify *hooks* to customize a particular *major mode,* rather than all of Emacs or the buffer you are in when you start up. There is no hook for fundamental-mode. To make your initial buffer a mode other than fundamental-mode (so you can customize it), you should use:
  (setq default-major-mode 'text-mode)
  This makes your default mode text-mode. You can then hook a series of commands to activate whenever you enter that mode. The syntax is:

  ```
  (setq
  major-mode-hook '(lambda nil
  (
  progn
  command(s))))
  ```

  Thus, the example from the above ~/.emacs file to set auto-fill mode is:

```
(setq text-mode-hook '(lambda nil (setq fill-column 72) (auto-fill-mode 1)))
```

You don't have to know Emacs Lisp to use hooks. Just know what mode you are working with, what commands you want to execute or variables you want to set, and make sure you have all your parentheses balanced. The quote is necessary.

Advanced Techniques

Using an Existing Emacs (emacsclient)

The **emacsclient** program tells programs like **comp** to use an existing Emacs window rather than starting their own. Note that if you tell programs to do this, they *always* try to find an existing Emacs. They fail if you don't have an Emacs running, or if you are in a tty session; they will not work until you create an Emacs and try again. Make sure you know what you are doing before you effect the conversion. There are three steps:

1. Set the EDITOR and VISUAL environment variables in your ~/.environment file:

   ```
   setenv EDITOR emacsclient # Default editor
   setenv VISUAL emacsclient # Screen editor
   ```

   (If you don't have EDITOR and VISUAL set in your ~/.environment file, just insert these lines. If you don't have a ~/.environment file, create it.)
2. Modify your ~/.mh_profile so that **comp** and **repl** know where to look:
   `Editor: emacsclient`
3. Add the following line to your ~/.emacs file (create it if you don't have it):
   `(server-start)`

The next time you log in, everything is set up to use the Emacs server (you still type **emacs &** to call up the editor to begin with). Of course, you can change everything back to the way it was by substituting emacs for emacsclient in ~/.environment and ~/.mh_profile.

Issuing Sophisticated Commands

You can issue commands with arguments, or reissue a complex command you already issued. The following commands help you do this.

Command	Action
C-u # *command*, M-x universal-argument	Do *command* # times. The default value is 4, and typing **C-u** again quadruples the previous value of the argument. You type this before commands that require a numeric argument.
M-#	Same as **C-u #**, without the default.
C-x Esc Esc, M-x repeat-complex-command	Display the last command of more than two keystrokes, and allow you to edit and execute it. A numeric argument calls up a list of earlier commands. While inside this list, you can use **M-p** and **M-n** to move up and down.

For example, **C-u M-d** deletes the next four words; either **C-u 3 M-d** or **M-3 M-d** deletes the next three words.

Keyboard Macros

Using Keyboard Macros

Keyboard macros allow you to define a new command to execute a specific group of several commands, and even name and save the new command so that you can continue to use it.

For example, you might want to delete a word, move down one line, and replace the word there, all with one command – but unfortunately, Emacs doesn't have a **move-word-down-line** command.

Here are the basic commands for working with keyboard macros:

Command	Action
C-x (, M-x start-kbd-macro	Begin the definition process.
C-x), M-x end-kbd-macro	Conclude the definition process.
C-x e, M-x call-last-kbd-macro	Execute the last defined keyboard macro.
M-x name-last-kbd-macro	Give a name to the current macro for the duration of this session.
M-x insert-kbd-macro	Insert a named macro into the current file. (Useful for saving a macro definition in your ~/.emacs file.)
M-x global-set-key	Assign a keybinding to an Emacs command or named macro.
M-x global-unset-key	Remove a keybinding from the Emacs command or named macro to it is assigned

C-x (starts the macro process, and the word Def is added to the mode line. From then on, everything you type is executed normally *and* stored in the macro. This continues until you type **C-x)**. To have that exact sequence repeated over again, starting from the current location of the cursor, type **C-x e.** Whenever you define a new macro, the old one is lost unless it has a name. All macros, named or not, only last until you exit Emacs.

To make a macro more permanent, give it a name (using **name-last-kbd-macro**) and then add it to a file (using **insert-kbd-macro**) for later use. In particular, if you put macro definitions into your ~/.emacs file (see the section on Making Changes Permanent Using Your .emacs File), you can have those commands available to you whenever you enter Emacs.

Suppose you wanted (for some inexplicable reason) to delete the last word on every line that had the word "happy" in it. You need to start the macro (**C-x (*), find "happy" (*C-s happy**), exit the search, move to the end of the line (**C-e**), delete the word behind the cursor (**M-Delete**), and end the definition (**C-x)**). To do this, type:

```
C-x ( C-s happy C-e M-Delete C-x )
```

Don't forget that all commands are executed while you type them into the macro. To make this new command more permanent, use **name-last-kbd-macro** to assign it the cheerful sounding name of **destroy-happiness**. Having named it, you can now execute this macro by typing **M-x destroy-happiness**, and you can define another macro without losing this one, until you exit Emacs.

To preserve this macro and this name, go to (or create) your ~/.emacs file and type:

```
M-x insert-kbd-macro
```

Insert kbd macro (name): **destroy-happiness**

This prints into your ~/.emacs file the code for the commands listed above. (You don't have to know Emacs Lisp or understand this code, but that's okay, Emacs understands them.)

Of course, there are more constructive uses for keyboard macros as well; lots of fun things you can do with macros are explained in Chapter 35 of the **GNU Emacs Manual.**

Keyboard macros, like other commands, may be bound to keystrokes for convenient use. See the Customizing Command Key Bindings module for more information.

Customizing Command Key Bindings

There are many Emacs commands that are bound to particular keystrokes for easy access. Many of these are listed in these modules: **C-x C-s** runs the **save-buffer** command, **C-x i** runs the **insert-file** command, and so forth. Often, however, your favorite commands are not bound to keystrokes, making it necessary to type out the whole command name. For example, to get Emacs to automatically do a carriage return at the end of each line, you can use the **auto-fill-mode** command, but in each buffer you must type out the whole command: **M-x auto-fill-mode.**

Emacs commands are easy to customize, using the **global-set-key** command. You can also use this command to assign keystrokes to named keyboard macros (see Using Keyboard Macros).

Let's say, for example, that you have defined a macro to delete the last word in the next line wherein the word "happy" is found, and you have

named this macro "destroy-happiness" (see Using Keyboard Macros for an explanation of how to do this). You can execute this series of commands by typing **M-x destroy-happiness**.

Since this name is still a little tedious to type, and there is a lot of happiness that must be destroyed, you can assign it to a key. You decide to use the command **M-*** to run this command. You type:

```
M-x global-set-key
Set Key globally:
M- *
Set key M-* to command:
destroy-happiness
```

So, the next time you want to kill the last word on a line that has the word "happy" in it, type **M-***.

You can use the **global-unsetset-key** command to *undo* a binding that is especially annoying or awkward. For example, let's say you keep accidentally hitting **M-z** when you mean to hit **M-<.** (**M-z** is bound by default to the **zap-to-char** command, which deletes all text until the first incidence of a particular character.) Let's say, further, that you can conceive of no possible use for the **M-z** command. You can solve this problem by unsetting the key. (Notice that now, to use the **zap-to-char** command, you must type out its full name.)

Be careful not to over-write any useful commands by mistake. Use **C-h c** to find out what a key sequence is being used for by default. Remember, also, that many keystrokes in Emacs are prefixes, and binding those keystrokes to a command will keep you from being able to use all of the commands that are prefixed with that keystroke.

All bindings you define go away when you exit Emacs. If you want to save these bindings, put them into your ~/.emacs file (see Customizing Your Emacs Environment). The syntax for the command is:

```
(global-set-key "
keystrokes" '
command-name)
(global-unset-key "
keystrokes")
```

The quotes are all important. **Ctrl** and **Meta** keystrokes are written **\C-** and **\M-**. For example, to put the **M-*** command into your ~/.emacs file, you would type:

```
(global-set-key "\M-*" 'destroy-happiness)
```

Editing and Compiling Programs in Emacs

Language Modes

There are many programming language modes, covering a wide range of behavior. The command **M-x run-lisp**, for example, actually runs a Lisp interpreter inside an Emacs window. You can look at Section 22.2 of the **GNU Emacs Manual** if you are interested in some of the Lisp interpretation abilities.

The language mode chosen depends on the suffix of the filename: .c for C, .f for FORTRAN, and .l for LISP. Each mode has its own peculiarities. LISP balances parentheses. C works with braces. FORTRAN has many abbreviations for common commands. All of them, however, share facilities for doing *indentation* and *commenting*.

In language mode, the **Tab** key, instead of just indenting eight spaces, moves the line to its proper indentation based on the number of open brackets and the indentation of previous lines. **Delete** is modified so that when it it tries to delete a tab, it treats it like an equivalent number of spaces. The indentation may not be perfect, but it is pretty good.

The key bindings for the commenting commands are based on the semicolon, since that is the comment character used in Lisp. Each language has its own particular comment character (which Emacs' language modes know about), but the commands are the same. The behavior is a little different in FORTRAN, of course. Here are a few of the most useful commenting commands:

Command	Action
M-;, M-x indent-for-comment	Start a comment on this line.
M-j, M-x indent-new-comment-line	Continue a comment on the next line if currently on a comment line.
C-x ;, M-x set-comment-column	Set column where comments should begin. Use either the cursor position or a numeric argument.

As with any mode, you can type **C-h m** to get all the mode-specific commands and variables. For detailed descriptions of these and other language mode commands, read Chapter 26 of the **GNU Emacs Manual**.

Compiling Within Emacs

People often use Emacs inefficiently when writing a program. They edit the file, exit Emacs, compile the file, re-enter Emacs, search for errors, fix

them, then start the whole cycle over again – unnecessarily. There are many Emacs facilities to speed up the whole process of compilation and debugging for programming languages.

The Emacs **compile** command allows you to compile files without ever leaving Emacs. When executed, it asks you what command you want to execute, then it asks you to save each buffer. The default command is **make -k** for programs. The **make** command uses a file called a Makefile to help compile multifile programs. (For more information, type **man make** at your Athena prompt.) If you don't have (or don't wish to use) a makefile, you can backspace over the default command and specify any command you like: **cc**, **f77**, **lint,** etc.

Emacs then splits the screen, creating a second window, and runs the specified command in the buffer **compile**. You can continue working in the first window, if you like. To abort the compilation, use the command **kill-compilation**:
`M-x kill-compilation`
If you start up a new compilation while an old one is still running, Emacs offers to kill off the old one. Otherwise, be sure to stay in two-window mode so you can use the error-finding routines. (See the section Finding Compilation Errors.)

Finding Compilation Errors

You've just written your program, and have used the **compile** command to compile it in Emacs. Unless you are lucky, you now have a buffer full of errors. Each error has a file name and line number associated with it. Emacs can use this information to move directly to those errors. The command to do that is **C-x `**, which is **next-error**.

To start, type **C-x `**. Notice that ` is the backquote, or grave accent, not the apostrophe. This moves you to the occurrence of the first error listed in **compile**. Emacs moves backwards and forwards, and even loads in new files if necessary. Once there, you can move around, insert and delete, and do whatever is necessary to fix the error. When you are ready, just type **C-x `** again to move to the next error. The second window scrolls to follow which error you are on. A numeric argument starts over from the first error.

You can stop at any time. For more information about compiling and debugging files within Emacs, consult Chapter 27 of the **GNU Emacs Manual**.

Version Control

Version control systems are packages that record multiple versions of a source file and record history infomation about the creation of these different versions. The version control features of Emacs works with multipme version control systemsincluding RCS, CVS, SVN, GIT, and the less-recommended SCCS. Type **man rcs**, **man cvs**, or **man sccs** at the athena prompt for more information about each of these control systms. The rest of this document deals with version control features within Emacs, regardless of which version control system you use.

The most useful VC command is an all-purpose command that performs either locking or check-in, depending on the situation. **C-x C-q**, **C-x v v**, and **M-x vc-next-action** all perform the the next logical version control operation on the file, be it checking in a new version or checking out a version to you. However, the precise action of the command depends on the state of the file and the version control system you are using.

When you check in changes with **C-x v v** a buffer called '**VC-Log**' pops up for you to enter the log entry. When you are finished, type **C-c C-c** to actually check in the file.

With version control, you can look at and compare old versions of files. Versions are numbered and the version number appears in the status bar of the Emacs window.

Command	Action
C-x v ~ *version* RET	Examine version *version* of the visited file, in a buffer of its own
C-x v =, M-x vc-diff	Compare the current buffer contents with the latest checked-in version of the file
C-u C-x v = *file* RET *oldvers* RET *newvers* RET	Compare the specified two versions of *file*

Comparing versions of a file causes a window containing the buffer ****vc-diff**** to pop up. This buffer shows the differences between the two versions by displaying both the old and new text found in changed sections.

Secondary Commands of VC

The above commands are the ones you primarly deal with when using version control. The following commands are used less often but are still useful.

Command	Action
C-x v i, M-x vc-register	Register the visited file for version control
C-x v l, M-x vc-print-log	Display version control state and change history
C-x v u, M-x vc-revert-buffer	Revert the buffer and the file to the last checked-in version
C-x v c, M-x vc-cancel-version	Remove the last-entered change from the master for the visited file. This undoes your last check-in.

For a file to be under version control, it must be registered with the system. Typing **C-x v i** will register the file in the current buffer. Once a file is registered, it cannot be edited except through the version control system.

As mentioned before, version control keeps logs of changes which include version number, date and time changed, who changed it, how much was changed, and a log message entered by the user who checked in that version. **C-x v l** will open a buffer containing all the logs associated with the current file.

M-x vc-revert-buffer and **M-x vc-cancel-version** are used when you make mistakes. Revert-buffer is used *before* you check in changes. If you're editing a file and decide you want the original (latest version) back (perhaps because you've badly mangled whatever was in the file), use **C-x v u**. If, however, you've just checked in a file and found a small mistake you need to correct, use **C-x v c** to undo the check-in just performed. You can then fix the mistake and check it back in, without having separate revisions for the major changes and the typo correction.

For more information on version control, see Chapter 18 of the **GNU Emacs Manual**.

Using Shell Commands from Within Emacs

Emacs is an extremely versatile program that lets you run a great many other processes through it. However, there are some things that you can't do directly from an Emacs window.

It is possible to create a buffer within an Emacs window that behaves like a normal X window, giving you an Athena prompt and accepting many commands that you would normally type in another window. The command **M-x shell** creates this buffer. You can switch between the shell buffer and other buffers like you switch between files, with **C-x b**.

The shell buffer is slower than the standard command line, and there are still some programs that can't be run through it, but you can try it. This is a way to use Emacs as your "window" manager when you're logged in via dialup on a non-workstation terminal.

More Things to Do Within Emacs

Emacs is more than just a word processor. In addition to the features described in this documentation, there are many other useful features. There are also many fun, but less-than-useful features we don't have room to describe here. Here is a brief, arbitrary listing of some of these:

Command	Action
M-x mh-rmail	Run mh-e, Emacs' interface to MH. For more information, see the section on Using the Emacs-based MH Mail Handler (http://web.mit.edu/olh/Email/email.html#mh) in the document Electronic Mail on Athena (http://web.mit.edu/olh/Email/email.html).
M-x discuss	Run the Emacs interface to Discuss, Athena's electronic conferencing system. For more information, see the section Emacs Interface to Discuss (http://web.mit.edu/olh/Discuss/Discuss.html#emacs_head) in the document Discuss on Athena (http://web.mit.edu/olh/Discuss/Discuss.html).
M-x gnus	Run gnus, the Emacs interface to network news, after doing initial setup.
M-x yow	Generates a randomly-chosen Zippy quote in the minibuffer.
M-x doctor	Creates a buffer and interactively runs an "Eliza" style computer psychotherapist program.
M-x psychoanalyze-pinhead	Runs **M-x doctor** with input from **M-x yow**.
M-x dissociated-press	Dissociate the text of the current buffer (changes the appearance on the screen, but doesn't change any text).
M-x dunnet	Text adventure game
M-x hanoi	Tower of Hanoi. Argument is number of rings.

Error Recovery

Dealing with System Failures

When you log in to an Athena public workstation, the workstation gives you access to your files by attaching your file system to the workstation's file system. Your file system is actually on another machine on the network, your file server machine. When you edit one of your files with Emacs, you are editing the file over the network. This system has three failure points:

- your workstation
- the network
- your file server machine

This section provides information about recovering your work in the event of any of these system failures.

Workstation Failure

The workstation is the most dependable part of the system; workstations do not fail very often. Workstations fail by either "freezing" (typing at the keyboard does nothing on the screen), or by crashing – your windows disappear and the machine begins rebooting itself.

When the workstation fails, you have few options. Your best bet is to log out and recover your work later from an auto-save file (you must be on the same workstation you were working on before for this). See the section on Recovering Lost Files for how to do this.

Network and File Server Failures

Network and file server failures are almost indistinguishable. What happens is this: you try to write your file and the workstation begins flooding the console window with messages like the following:

```
afs: Lost contact with file server 18.145.0.6 in cell athena.mit.edu
```

If you have the time, wait and see if the file server comes back online. Often the failure will correct itself in five or ten minutes. When the server comes back you can write the file to disk.
If you must leave before the system comes back, you can mail the file to yourself and save it that way. Proceed as follows:

1. Use the **C-x C-w** command to write the file. When you enter this command, Emacs prompts you to specify the name of the file you want to write:

   ```
   Write file: /mit/
   username/
   ```

2. Backspace over the pathname, then type **/usr/tmp/** ***filename*** and **Return** (*filename* specifies the name of the file you want to save). You have now written the file to the workstation's hard disk.
3. Move to an xterm window and send yourself the file in the form of a mail message. Substitute your username for *username*, and the filename you created above for *filename*:

   ```
   mhmail
   username < /usr/tmp/
   filename
   ```

   The **mhmail** command sends a copy of the file you created above to your Athena mailbox. Next time you log into Athena, you will receive the file when you incorporate your mail.

Recovering Lost Files

If the auto-save version of the file you are working on is newer than the current version (this may happen if the workstation crashes and you later log in to the same workstation), Emacs displays the message:
`Auto-save file is newer - consider M-x recover-file.`
You can recover an auto-save file by doing the following:

1. Login to the workstation you were using at the time of the crash. Use the **ls** command to see if your file is still in the /usr/tmp directory:

   ```
   athena% ls /usr/tmp
   ```

   Auto-save files have funny names. Say your username is jruser and you are editing the file /mit/jruser/paper.tex. Within /usr/tmp, the file's auto-save file has a name something like this:

   ```
   #4863.@afs@athena.mit.edu@user@j@r@jruser@paper.tex#
   ```

2. Start Emacs and enter this command:

   ```
   M-x recover-file
   ```

3. Emacs prompts you to enter the name of the file you want to recover. Enter the filename and press **Return**.
4. Emacs splits its window in two and asks if you want to recover the auto-save file. (Note that for long filenames, you won't be able to see the entire question.)
5. If you want to recover the file, type **yes** followed by **Return**.
6. Emacs creates a new buffer for the file, and copies the auto-save file's contents into it. The editor also informs you that it has turned off the buffer's auto-save-mode. This is done so that you won't inadvertently overwrite the auto-save file.
7. If you want to continue editing the recovered file, turn on auto-save-mode by entering **M-x auto-save-mode** followed by **Return.** Otherwise, save it and exit Emacs.

Getting Help and More Information

Emacs is full of hundreds of features, commands, variables, and neat tricks including many not described here. To find out about Emacs features, consult the following information resources:

1. The Emacs help command **C-h** provides extensive assistance on-line. The general form of the command and useful subcommands are listed here:

Command	Action
C-h C-h, M-x help-for-help	List the various characters that give further information about different parts of Emacs.
C-h a, M-x command-apropos	List all Emacs functions that contain a given word or string. (e.g., **C-h a write** lists all commands with "write" in their names. Emacs commands have descriptive names, so you're likely to find what you want.)
C-h c, M-x describe-key-briefly	Give name of any keystroke command.
C-h f, M-x describe-function	Describe any Emacs function.
C-h i, M-x info	Read documentation.
C-h k, M-x describe-key	Describe any keystroke command.
C-h m, M-x describe-mode	Describe the commands unique to the current mode.
C-h v, M-x describe-variable	Describe any variable.

   Warning: Do not try **C-h** until you have read Working with Multiple Buffers, or the extra buffers created by help may confuse you.

2. There is an interactive tutorial that you can use to teach yourself Emacs. To use this tutorial, enter the **C-h t** command in Emacs, then follow directions. You can get out of the tutorial at any time by entering **C-x C-c**, just as you would to exit any normal Emacs session. If it asks you to save TUTORIAL, just say no. Decide for yourself if you want to save any other buffers it asks about.
3. The main GNU Emacs Manual describes all of Emacs in detail.
4. You can view the Emacs manual page by typing **man emacs** at the Athena prompt.
5. Emacs Stock Answers in Hermes
6. Ask Athena Consulting a Question

Related Links

Athena Help
Athena at MIT