# AN UNDERGROUND GUIDE TO TEXT EDITING ON PROJECT ATHENA

An introduction containing useful information

for processing text on Athena

by the Student Information Processing Board

MIT Room 11-205 -- (617) 253-7788

Copyright (c) 1987 by the Student Information Processing Board

February 28, 1987

## 1. Before You Begin

This document should give you a basic idea of how to do your text processing on Athena. Remember, however, that **the best way to learn is by example**. This document, for instance, was produced entirely with Athena facilities. The source file for this document is located in the file:

**`/usr/unsupported/sipb/Scribe/underground-guide.mss`**

on all Athena machines. Look at it to see how we made this document. As an example of the capabilities of this system, this entire document, *An Underground Guide to Text Editing on Project Athena*, was produced using Athena text formatting facilities.

If you have any questions about text processing or anything having do to with computers, you can call the SIPB office at 3-7788 or drop by (room 11-205, above the building 11 terminal room). Another resource is Athena's student consultants, who are often on duty in the terminal rooms.

Note: This is *not* an official Athena document. If you have any comments, suggestions or complaints, please send them to the SIPB. You can either leave a note in the SIPB office or send computer mail to ''uus'' on any Athena machine.

## 2. Introduction to Word Processing on Athena

The Athena computer system offers a wide range of commands for editing, correcting, and formatting text. This facility is useful for producing project reports, program documentation, and written papers of all kinds. The major available facilities that are the most useful to new users are outlined below:

- **Text Editors** (such as Emacs) enable the user to interactively enter, modify, and save text in ''files.'' The text can then be printed directly on the user's terminal or by a high-speed line printer. The editor can be used at any time to change the text file in order to make further additions, deletions, or corrections.

- **Text Processors** (such as Scribe) can take text files and produce a more professional looking version of the text than that typed in at the terminal. Text formatters can ''justify'' pages so that the right-hand margin is straight, place headers and footers on each page, center titles or equations, print footnotes and page numbers in a uniform manner, hyphenate words at the end of a line, and generally enhance the appearance of a document. Experienced users can use text formatters to create tables of contents or indices, format tabular lists of data and perform many more advanced forms of text processing. The basic concepts of text formatters, including a description of how to use them, are presented in the section ''Concepts in Text Formatting.''

- **Printer** facilities make it possible to get text files printed on paper of various different sizes and weights. Text files can always be printed directly in any terminal room on the local line printer, or your can get high quality output by sending them to the Xerox 9700 printer run by IS.

## 3. Typing in a Document

When you write a paper on Athena, you will use a text editor and a text formatter. A text editor is used to enter your text into the computer and make changes to it -- it's like a correcting typewriter, only better. A text formatter takes that ''raw'' text and makes it look nicer -- it can straighten out both the left and right margins, insert and number footnotes, number pages and do other things to make your paper look more professional. The Athena text editor is called ''Emacs''. You will use it for everything from writing papers to writing computer mail messages to friends.

Generally, the advantage of computer text preparation is only having to type anything once. Once a document is ‘‘in’’ the computer (and *saved* into a ‘‘file’’), any corrections or changes are made by giving commands to replace only the desired parts, without having to retype the entire document. The computer can be told to permanently remember any text typed in. The place the computer remembers the text in is called a ‘‘file’’. You can have several files. Names can be anything you want as long as they do not have the ‘‘/’’ character in them. (The slash is used for specifying files in another directory). An uppercase character is different from a lower case character in a name. (i.e. ‘‘MyFile’’ is not the same name as ‘‘myfile.’’) By convention certain portions of a name are used to mean certain things. For example, having a suffix of ‘‘.mss’’ in a name means that the file is meant to be processed by the Scribe text formatting program. See the section on ‘‘Concepts in Text Formatting’’, page (11).

## 4. Terminology and Abbreviations

Certain terms and operations will be referred to frequently in this document. To simplify notation, standard abbreviations for these phrases have been developed. A listing of these abbreviations follows.

| | |
|---|---|
| SIPB | Student Information Processing Board |
| RETURN | carriage return key, sometimes labelled ‘‘RETURN’’ or ‘‘RET’’ |
| M- | ‘‘ESCAPE’’ key, sometimes called ‘‘ESC’’ |
| C- | ‘‘CONTROL’’ key, sometimes labelled ‘‘CTRL’’ or called ‘‘^’’ |
| DELETE | ‘‘DELETE’’ key, sometimes labelled ‘‘DEL’’ or ‘‘RUBOUT’’ |
| LINEFEED | ‘‘LINEFEED’’ key, sometimes labelled ‘‘LF’’ |

**examples**:

| | |
|---|---|
| foobar RETURN | means press the carriage RETURN key after typing ‘‘foobar’’. |
| C-X | means *hold down the CONTROL key while typing the character ‘‘x’’*. This document (and most others) uses capital letters (e.g. C-X) to signify this, but that does not mean you need to type a capital X. |
| M-X | means *press and release the ESCAPE key, then type the character ‘‘x’’*. Neither the dash nor the space is typed; they are merely the written representation of the sequence just described. |
| | Note: unlike C-X, whenever you use the ESCAPE key you release it before hitting the next key. |

## 5. The Emacs Editor

The Emacs editor is a program which allows you to create text and enter it in the computer’s memory. It allows you to type, change and move words and phrases, and to store them in the computer for further use. Text you create will appear as if the screen were one infinite piece of paper. The advantage is that you can make changes in the text quickly and easily, without using an eraser or getting more paper.

Use Emacs to create a file you eventually want printed. Various features in Athena enable you to produce thesis quality papers using Emacs and Scribe, the associated text formatter.

### 5.1. Getting Started

First log into your Athena account. If you do not know how to do so, stop reading this document and read *Essential Athena* or ask a consultant.

One of the most important things about the terminal that you need to know is where the cursor is. The cursor is a blinking box or underscore on most terminals. The cursor is said to be ON the character which is at the same place as the cursor is. The cursor is important because all changes take place at the cursor.

After you log in and give your password, Athena will print various messages and current information, followed by a ''prompt''. Unless you have changed it, your prompt will probably look like:

```
teela%
```

except that instead of ''teela'', you will see the name of the machine on which you have your account. At that point, you are at ''unix command level.'' You can learn more about this level in *Essential Athena* and *Essential Unix*, as well as the SIPB's *Inessential Guide to Athena*. Whenever Athena finishes executing a program or operation, it returns to command level, prints a prompt and waits for you to type what to do next.

The commands you must use in Emacs differ from the commands you use at command level (before entering Emacs, for example). You must always be aware of whether you are in or out of Emacs, so that you know which commands are appropriate.

The first thing you should do is to type (at unix command level)

```
emacs -f help-with-tutorial RETURN
```

This will put you in a very good Emacs tutorial. Just read your screen and follow directions. You can stop whenever you want to.

By the time you finish using the online tutorial, you will have a good idea of how to use Emacs. At that point, you will be ready to start text editing. While at command level, enter the editor. The simplest way to enter Emacs is to type

```
emacs filename RETURN
```

The word ''filename'' is whatever name you give the text you will be writing. Thus, if you want to write about farm techniques for growing apples, you might type

```
emacs apples RETURN
```

If this is the first time you have written in the file specified (e.g. apples), Emacs will tell you after a moment that ''apples'' is a new file, and create a file by that name. Your file is kept in a ''buffer'', which you can think of as a long scroll onto which you will type. A buffer is a temporary storage area. On the screen, you will see a large blank section called the ''main window'', and some writing at the very bottom of the screen, in the ''minibuffer''. The main window is where you will type your text. It really is a sort of window, through which you can view various sections of the buffer. The writing at the bottom of the screen tells you the name, ''filename'', of the buffer containing your text. Many commands you may use in Emacs also cause writing to appear in the minibuffer.

If, while using Emacs, you get into a state you don't understand (such as Emacs asking you for input at the bottom of the screen), type C-g C-g (two CONTROL g's -- that is, hold down the CONTROL key and type g twice). If you wish to get out of Emacs completely, type C-x C-c (C-x followed by C-c).

*Most of the following is in the emacs tutorial. It is included here for reference*.

**WARNING**: Commands involving C-s and C-q may not work on many terminals. C-s may freeze the screen so that you cannot type anything else. Typing C-q will unfreeze the screen. For commands which use C-s and C-q, you will find equivalents posted on the terminal itself.

## 5.2. Moving Around in Emacs

The first thing you need to know is how to move from place to place in the file. You move forward a screen by typing C-v. To move backward a screen, type M-v (strike the ESCAPE key and then type v). There is a small overlap when going from screen to screen; this provides some continuity when moving through the file.

If you accidentally let go of the CONTROL key before typing the v, you will not advance to the next screen. Instead, the CONTROL will be ignored and the letter v will be put in the file, at the cursor. To get rid of it, simply press the DELETE key once, and you will be ready to type C-v correctly.

The command C-l is also useful for arranging the text on the screen. Typing C-l will cause the screen to clear, then all the text will be redisplayed, with the line on which the cursor was located centered on the screen.

## 5.3. Basic Cursor Control

Getting from screenful to screenful is useful, but how do you reposition yourself within a given screen to a specific place? There are several ways you can do this. One way is to use the commands next-line, previous-line, backward-char, and forward-char. These commands are given to Emacs by the characters C-n, C-p, C-b, and C-f respectively. As you can imagine these commands move the cursor from where it currently is to a new place in the given direction. Here, in a more graphical form, are the commands.

```
                Previous line, C-p
                     :
                     :
 Backward, C-b .... Current cursor position .... Forward, C-f
                     :
                     :
                Next line, C-n
```

You'll probably find it easy to think of these by letter. p is for previous, n for next, b for backward and f for forward. These are the basic cursor positioning commands. You'll be using them ALL the time so it will be of great benefit if you learn them now.

Lines are separated by special characters, called NEWLINE's. When the cursor is at the end of a line, it is on the NEWLINE. When you go off the top or bottom of the screen, the text beyond the edge is shifted onto the screen so that your instructions can be carried out while keeping the cursor on the screen. If moving by characters is too slow, you can move by words. M-f moves forward a word and M-b moves back a word.

Notice the parallel between C-f and C-b on one hand, and M-f and M-b on the other. Very often ESCAPE characters are used for operations related to English text (words, sentences, etc.) whereas CONTROL characters operate on the basic textual units that are independent of what you are editing (characters, lines, etc). There is a similar parallel between lines and sentences: C-a and C-e move to the beginning or end of a line, and M-a and M-e move to the beginning or end of a sentence.

Two other simple cursor motion commands are M-<, which moves to the beginning of the file, and M-> which moves to the end of the file. If you need the shift key to type a ‘‘<’’, then you must also use the shift key to type it when you type M-<. To type M-< press ESCAPE then release it, then press the comma while holding down the shift key.

The commands mentioned so far, and most others in Emacs, can be given arguments which cause them to be executed repeatedly. The way you give a command a repeat count is by typing ESCAPE and then the digits before you type the command. If the command also contains an ESCAPE (as in M-f) you must type ESC a second time for the command itself. For instance, M-8 C-f moves forward eight characters, and M-3 M-b moves backward three words.

Here is a summary of simple moving operations including the word and sentence moving commands:

| | |
|---|---|
| C-f | Move forward a character |
| C-b | Move backward a character |
| M-f | Move forward a word |
| M-b | Move backward a word |
| C-n | Move to next line |
| C-p | Move to previous line |
| C-a | Move to beginning of line |
| C-e | Move to end of line |
| M-a | Move back to beginning of sentence |
| M-e | Move forward to end of sentence |
| M-< | Go to beginning of file |
| M-> | Go to end of file |

### 5.4. Cancelling Commands

If you begin typing a command that takes more than one keystroke, such as M-a, and decide that you don’t wish to execute that command, you can cancel it by typing C-g. This can also be used to cancel any numeric arguments that have been entered. The terminal bell will ring to let you know that the command has been cancelled.

### 5.5. Inserting and Deleting

If you want to type text, just do it. Characters which you can see, such as A, 7, or * are taken by Emacs as text and inserted immediately at the cursor location. Type RETURN to insert a NEWLINE.

You can delete the last character you typed by typing DELETE. DELETE deletes the character immediately to the left of the current cursor position.

This is the most basic way of correcting errors in Emacs. You can delete by words or lines as well. Here is a summary of the delete operations:

| | |
|---|---|
| DELETE | delete the character to the left of the cursor |
| C-d | delete the character under the cursor |
| M-DELETE | kill the word immediately to the left of the cursor |
| M-d | kill the word after the cursor |
| C-k | kill from the cursor position to end of line |
| M-k | kill to the end of the current sentence |
| C-x DELETE | kill to the beginning of the sentence |

(C-x commands are typed by typing a C-x and then the following character.)

Notice that DELETE and C-d versus M-DELETE and M-d extend the parallel started by C-f and M-f (well, DELETE isn't really a control character, but let's not worry about that). C-k and M-l are like C-e and M-e, sort of, in that lines are opposite sentences.

Now, what if you kill something, and then you decide that you want to get it back? Well, whenever you kill something bigger than a character, Emacs saves it for you. To yank it back, use C-y. Note that you don't have to be in the same place to do C-y. This is a good way to move text around. Also note that the difference between ''killing'' and ''deleting'' something: ''killed'' things can be yanked back, but ''deleted'' things cannot. Generally, the commands that can destroy a lot of text save it, while the ones that attack only one character, or nothing but blank lines and spaces, do not save.

A single C-k kills the contents of the line, and a second C-k kills the now-blank line, and make all the other lines move up. If you give C-k a repeat count, it kills that many lines AND their contents. You can think of C-y as if you were yanking something back that someone took away from you. If you do several C-k's in a row the text that is killed is all saved together so that one C-y will yank all of the lines.

What do you do if you have some text you want to yank back, and then you kill something else? C-y would yank the more recent kill. But the previous text is not lost. You can get back to it using the M-y command. After you have done C-y to get the most recently killed stuff, typing M-y replaces that yanked text with the previously killed text. Typing M-y again and again brings in earlier and earlier kills. When you have reached the text you are looking for, you can just go away and leave it there. If you M-Y enough times, you come back to the starting point (the most recent kill).

### 5.6. Extending the Command Set

There are many, many more Emacs commands than could possibly be put on all the control and escape characters. Emacs gets around this with ''extended'' commands - commands which need more than one character to invoke them. These have two forms:

| | |
|---|---|
| C-x | Character extend. Followed by one character. |
| M-x | eXtended-command. Followed by a long name. |

There are many C-x commands. The ones you need immediately are:

| | |
|---|---|
| C-x C-f | find-file. |
| C-x C-s | save-same-file. |

C-x C-c quit-the-editor. This does NOT necessarily save your file! To prevent you from accidentally quitting Emacs without saving, if your buffer is modified you will be asked if you really wish to quit. The standard way to save and exit is C-x C-s C-x C-c.

Extended commands are commands which are used even less frequently, or commands which are used only in certain modes. Consider, for example, the function ''replace-string'', which globally replaces one string with another. When you type M-x, Emacs prompts you in the minibuffer with ''M-x'' and you should type the name of the function you wish to call -- in this case, ''replace-string'' and RETURN. You type the string that you want to replace, RETURN, then the string you want to replace it with, and another RETURN.

## 5.7. Mode Line

At the bottom line of the screen, you will see a line like:

```
--**-Emacs: filename.mss  (Text)  --22%----------
```

This is a very useful information line. In this example, ''filename.mss'' is the name of the file you are editing. You are currently editing at a position 22% of the way through the file. The ''**'' before the ''Emacs'' means that you have made changes to the text since the last time you saved it.

The part of the mode line inside the parentheses tells what modes you are in. The default (what the computer assumes when you don't specify something different) mode is Fundamental. It is an example of a ''major mode''. There are several major modes in Emacs for editing different languages and text, such as C mode, LISP mode, etc. At any time one and only one major mode is active, and its name can always be found in the mode line. Each major mode makes a few commands behave differently. For example, there are commands for creating comments in a program, and since each programming language has a different idea of what a comment should look like, each major mode has to insert comments differently. Each major mode is the name of an extended command, which is how you get into the mode. For example, ''M-x text-mode'' is how to get into text mode, and ''M-x c-mode'' is the command to get into C Mode.

A very useful minor mode is ''fill'' mode, which can be set by ''M-x auto-fill''. Normally you have to type RETURN at the end of each line in Emacs. If you are in fill mode, Emacs will automatically put in a carriage return when you reach the margin.

The margin is usually set at 70 characters, but you can change it with the C-x f command. You can give the margin setting you want as a numeric argument, or leave out the argument to cause the margin to be set to the column the cursor is in when you give the command.

## 5.8. Searching

The Emacs editor can do searches for strings (these are groups of contiguous characters or words) either forward through the file or backward (reverse) through it. To search for the string means that you are trying to locate it somewhere in the file and have Emacs show you where the occurrences of the string exist. The command to initiate a forward search is C-s. When you type C-s the string ''I-Search:'' appears as a prompt in the minibuffer (at the bottom of the screen), along with the cursor. Right there, type whatever string you want to search for, and as you type it, Emacs will search for the first occurence of that string, putting the cursor at the end of the first occurence. Typing C-s again will cause Emacs to search for the next occurence, and so on.

If no such occurrence exists Emacs beeps and tells you that it is a failing search. c-g would also cancel the search if and put the cursor back where it was when you started the command. (as it would for any command that prompts for an argument, such as C-x C-f).

The C-s starts a search that looks for any occurrence of the search string *after* the current cursor position. But what if you want to search for something earlier in the text? To do this, you should type C-r for reverse-string-search. C-r is just like C-s except that the direction of the search is reversed.

## 5.9. Getting More Help

Just enough information is supplied here to get you started using Emacs. There is so much available in Emacs that it would be impossible to explain it all here. However, you may want to learn more about Emacs since it has numerous desirable features that you don't know about yet. Athena supplies an Emacs wallchart, which is a concise listing of many Emacs commands. It is useful to have on hand while you edit.

The Emacs editor has a great deal of internal documentation. All of these commands can be accessed through the help command, which is gotten using the C-h character.

To use the HELP features, type the C-h character followed by a character saying what kind of help you want. If you are REALLY lost, type C-h ? and Emacs will tell you what kinds of help it can give. If you have typed the C-h character and decide you don't want any help, just type C-g to abort that command.

If the answer to your question is more than one line long, Emacs will split your window into two pieces. One has your original buffer; the other holds the contents of the help buffer. To view more than the first screen of the help buffer, use the command C-x o (for other-window). Then you can use C-v and M-v to move around in the help buffer. C-x o will change windows again. C-x 1 will delete the 'other' window -- the one your cursor is *not* currently in.

The most basic HELP feature is C-h c. Type C-h, then a c followed by a command character, and Emacs prints a description of the command.

For example, if you type C-h c C-p the screen will display

```
Control-P runs the command previous-line
```

in the minibuffer (at the bottom of the window).

The name of the ''function'' is important for people who are customizing Emacs. It is what appears in the Emacs wallchart as the documentation for the command character. For now you can ignore it.

Multi-character commands such as C-x C-f and M-v are also allowed after C-h C.

Here are some other useful C-h options:

| | |
|---|---|
| C-h d | Describe a function. You type in the name of the function. |
| C-h a | Apropos. Type in a word and Emacs will list all the commands whose names contain that word. |

For example, if you type C-h a file RETURN, you will see a list of all commands with ‘‘file’’ in their names, such as C-x C-f and C-x C-s, listed under the corresponding function names.

## 5.10. Saving Files and Leaving Emacs

Your text file will not be held in the computer’s permanent memory (written on disk) until you save it. This means that if the computer crashes (stops working), your text will be lost unless it has been saved.

## 5.11. Saving Named Files

After typing a paragraph or two, type C-x C-s. This tells Emacs to save your file. A message will appear at the bottom of the screen telling you first that Emacs is saving the file, then that the file is saved. Once that is done, chances are very slight that a system crash will cause your file to disappear.

Emacs tries to help you out by auto-saving your file every 100 keystrokes or so. This file has the name #<your.filename.goes.here>. However, it is best NOT to count on this mechanism, since sometimes a crash will leave you without an autosave file.

If you save (C-x C-s) every time you type a few paragraphs, the most you will ever lose is those paragraphs you have added since the last time you saved the file. You will regret it if you type five pages without saving and then the computer crashes.

## 5.12. Saving Unnamed Files

It is possible, although not a good idea, to enter Emacs without specifying a file name. This would happen if you entered Emacs by typing to Unix command level

```
emacs RETURN
```

where you neglected to type the name of your file before you hit the carriage return key. To save the file you type into the unnamed buffer, type C-x C-w. The words ‘‘Write File:’’ will appear at the bottom of the screen, along with the cursor. To save your work in a file called ‘‘alpha’’, for example, type

```
alpha RETURN
```

This stores your text in the file ‘‘alpha’’. Do NOT choose an existing file name unless you really do intend to replace the former contents of that file with the new text. Subsequent saves to the same file may be done by typing C-x C-s.

## 5.13. Saving Changes in Alternate Files

You may want two versions of a file. If you edit the file ‘‘apples’’ but do not want to destroy the current version of apples, write the altered file into another file. Type C-x C-w and specify another file name, perhaps ‘‘apples_new’’. The altered version will be saved in apples_new, and the original file (apples) will be intact providing you do not save the changes (C-x C-s). You are now working in the new file (apples_new), and if you make any changes now and save with C-x C-s, the changes will be saved in apples_new, still leaving apples intact. You can see a copy of apples by re-reading (C-x C-r) apples. ‘‘Re-reading’’ tells the computer to look at what is stored in a file and to redisplay it on the screen. Once you re-read apples, any changes you save with C-x C-s will be made in apples.

## 5.14. Leaving Emacs

When you are done editing a file and wish to edit Emacs (returning to Unix command level), type C-x C-s to save the last changes you made, then type C-x C-c to leave Emacs, If you have not saved your file, Emacs will warn you and ask if you really want to exit. If you do not save it, then any work that hasn't been saved will be lost. What you see in your buffer is only a copy of your file. Until you save the buffer (C-x C-s), the changes are not made to the permanent file.

## 5.15. Finding Files

To begin work on an already existing file, type C-x C-f. The words ''Find File:'' and the cursor will appear at the bottom of the screen. Type

```
filename RETURN
```

specifying the name of the file you wish to edit. This file will be read into a buffer, ready for editing.

## 5.16. Inserting Files

It is often useful to combine files. Suppose you decide to insert an appendix into a history paper you wrote. To insert the text of ''appendix'' into the text of ''history.mss'' for instance, use the C-x i command. While you have history.mss displayed on the screen, type C-x i (Note: this is not the same as C-x C-i). The cursor, and the words ''Insert File:'' will appear at the bottom of the screen. If you type:

```
appendix RETURN
```

then the contents of the file ''appendix'' will be inserted in history.mss, wherever your cursor was located in history.mss. Either save this combined file in history.mss, or write it into another file, e.g. improved-history.mss (see ''Saving Changes in Alternate Files''). The file appendix will still be intact. If you are writing the combined file into improved-history.mss, you will also leave history.mss intact.

## 5.17. Suspending and Re-entering Emacs

You may want to leave Emacs temporarily, then return to editing in it. Rather than killing (C-x C-c) your Emacs and making the computer start up a new one, use C-x C-z and the ''fg'' command. This makes life on an overloaded machine much nicer for you and all those around you.

When you want to leave Emacs, save your file (C-x C-s), then type C-x C-z. This takes you out of Emacs but leaves your Emacs suspended. Now you are at Unix command level.

Do whatever you wish at command level, then type:

```
fg %emacs RETURN
```

to re-enter your Emacs. You will re-enter exactly where you left. This means that the file will be precisely as it was when you typed C-x C-z, with the same text displayed on the screen. Note that it IS possible to suspend an Emacs, type ''emacs filename'' and have *another* Emacs. We recommend that you do NOT do this, as it can cause problems (in particular if you are editing the same file with both Emacs's).

# 6. Concepts in Text Formatting

Who among us has not marvelled at the nifty way the left and right margins in books seem to form straight lines or at least never run all the way off the edge or into the binding? This is done by the magic of text filling and adjusting. Filling is the process that makes sure that just the right amount of text winds up on each line. This is done by chopping off sentences in the middle of the line or scooping up words from the next line until the current line is full. Adjusting then takes the filled line and spaces things out neatly so the last character on the line winds up in the last column.

a) before anything

```
This is some sloppy
looking input text
obviously typed in by a normal human type of person
who
has trouble with the carriage return key.
```

b) after filling

```
          This is some more sloppy looking input text obviously typed
in by a  normal human type of person who has trouble with the carriage
return  key.
```

c) after adjusting

```
          This is some more sloppy looking input text obviously typed
in by a  normal human type of person who has trouble with the carriage
return  key.
```

The program called ‘‘scribe’’ normally fills and adjusts your text. This means that you can type in your text without paying any attention to details such as line length or margin size and the Scribe program will format it neatly to your specifications. Furthermore, since the text is stored in the computer’s memory, it is easy to update and then use the Scribe program again. This eliminates such familiar problems as ‘‘inserting one line and having to slide everything down somehow’’ or the exasperating ‘‘having to renumber all of the footnotes’’.

*Without any special commands, Scribe will generate formatted output*. It will single space your paper, filling and adjusting all lines so that they are 65 characters long.

The Scribe program does not center lines, put up page numbers, manage your footnotes or generate a table of contents or perform any advanced formatting without being told to do so. These can only be done by means of special commands to Scribe.

## 6.1. Document Types

Scribe has several *Document Types*. There are document types for normal text, letters, resumes and many other standard documents. Scribe ‘‘knows’’ about these different document types and what they look like. **To specify a certain document type, you insert the line**:

```
@make(DocumentType)
```

at the beginning of your text.

If you don’t specify a document type, Scribe will use the ‘‘text’’ document type, which is a generic document style. Below are descriptions of some of the most commonly used document types:

**Text** This is the default document type for Scribe. Paragraphs will not be indented, but

| | |
|---|---|
| | text will be *justified* and *filled*. Lines will be *single spaced* and pages will be *numbered* |
| **Article** | We recommend this document type for writing ordinary papers. It will give you a title page and double spacing as well as justifying and filling your text. |
| **Letter** | This document type will produce a letter in standard business form |
| **Resume** | This document type will produce a resume in a standard form |

For example, if you are writing a paper, you would type:[1]

```
@make(Article)
```

There are many more document types, but these are the ones which you will probably use most often. For a more complete list, look in the Scribe reference manual.

The SIPB has written a number of *templates* which are skeletons of standard Scribe files. We have filled in the important commands. You simply fill in your text. You will find these in ***/usr/unsupported/sipb/Scribe*** in files called ***letter_template.mss*** and ***paper_template.mss***. Feel free to copy them to your own directory and use them to write your paper.

There are also some unsupported templates in /usr/unsupported: the directory ***thesis***, which contains templates and instructions for doing your thesis in Scribe, and ***resume.sample***, which is the resume template.

## 6.2. Scribe Environments

In order to customize your formatting (ie, do anything interesting at all), Scribe provides *environments*. Scribe sets up environments in the following way:

Normally when Scribe reads text from your input file it assumes that it is simply text, unless it finds the character ‘‘@’’. *All Scribe commands begin with ‘‘@’’* Scribe commands may appear anywhere on a line -- they do not have to be at the beginning.

For example, to emphasize a word, use the Scribe command ‘‘@i’’ (for Italics). Thus, the Scribe command:

```
@i(This text is
emphasized.)
```

will produce: <u>This</u> <u>text</u> <u>is</u> <u>emphasized.</u>

As with most Scribe commands, *there are two ways to invoke the emphasize environment*. You can do it as above or, if you need to underline a larger region of text, by a sequence of Scribe commands like:

```
@begin(i)
This is a large region of text.  You really
wouldn't like to enclose this whole thing
in parentheses.
@end(i)
```

[1]Note that this is different from the method suggested in *Essential Scribe*. We think this works better, with less work

Which results in the document:

*This is a large region of text. You really wouldn't like to enclose this whole thing in parentheses.*

*Centering* is done by the *@center* command which puts you in a centering environment. To center the title of your paper:

```
@begin(center)
The Determination of Minor Quadratic Razors
in the Context of the Major Polemic Poets

by
Mildron A Quuxstuff
@end(center)
```

would produce:

```
               The Determination of Minor Quadratic Razors
                in the Context of the Major Polemic Poets

                                   by
                           Mildron A Quuxstuff
```

To insert a certain number of lines verbatim there is the @verbatim request which takes input lines and inserts them into the output file as is (*except* for "@'"s).

The Scribe program also provides many other useful commands:

Centering — The *@center* command centers text on a line. The center environment does not fill or adjust, for obvious reasons. It may be invoked with *@center(text) or @begin(center) and @end(center)*

Emphasizing — Use the *@i* command to emphasize text. It will produce italics if your printer is capable of doing so; otherwise, it will underline. Again, it may be invoked with *@i(text) or @begin(i) and @end(i)*.

Subscripts — To do subscripts, use the @- command. For instance,

```
This is text@-(subscript)
```

results in:

This is text$_{\text{subscript}}$

Superscripts — Use the @+ command, identical in all other respects to the @- command.

Footnotes — To put a footnote at the bottom of the page, use *@foot(text-of-footnote)*. This will put text-of-footnote in a footnote at the bottom of the page. Footnotes are sequentially numbered from the beginning of the document. As usual, if you have a long footnote, you may use @begin(foot) and @end(foot). For example, the following line:

```
@Foot(This is an example of a footnote)
```

will produce a footnote like this.[2]

Quotations — To do a long quotation, use @begin(quotation) and @end(quotation). This will single single space and change the indentation in the standard way.

Verbatim — The verbatim environment, invoked with @begin(verbatim), does not do any processing of your text. What you type is what you get.

[2]This is an example of a footnote

Blank lines — To get blank lines, use the @blankspace() command. It has several formats. For example:

```
@blankspace(2 lines)
```

will insert two blank lines into your text. and

```
@blankspace(1 inch)
```

will insert two inches of blankspace.

Program Example — For display program text, use @begin(ProgramExample) and @end(ProgramExample)

## 6.3. The style command

It is also possible to customize certain global aspects of your Scribe document. This is done with the @style command. In general, the @style command has to be used at the beginning of the file. For example, to change the width of a line from the default of 65 characters, you would use the command:

```
@Style(linewidth
72)
```

Here is a list of some of the more important attributes which can be set with @style:

Margins — BottomMargin is the Vertical distance between the last line of text and the bottom of the page. *TopMargin* is the distance between the top of the paper to the first line on the page. *LeftMargin* and *RightMargin* are the distance to the edge of the paper of the left and right margins, respectively. For example,

```
@style(BottomMargin 5lines)
```

would give you a bottom margin of 5 lines and

```
@style(LeftMargin 1inch)
```

would give you a left margin of 1 inch.

Spacing — The spacing between lines. For example:

```
@style(spacing
= 2)
```

will give you double spacing in a document type which would otherwise single space your text. (Note that on the 9700 printer, you can set fractional spacing)

Justification — Whether or not Scribe justifies text can be set with:

```
@style(Justification off)
```

or

```
@style(Justification on)
```

Linewidth — Adjusts the width of each line, as above.

## 6.4. A Few More Things To Remember

1. <u>Always</u> remember to <u>close an @begin() with and @end()</u>, or Scribe will not know when to stop the environment.
2. If you want to insert an <u>@</u> into your text, type ‘‘@@’’.
3. If you want to nest Scribe commands, you can use any of the symbols (), [], "", {}, not just (). If you have one Scribe command inside another, ie.

```
@center[@u(This output is centered and
underlined)]
```

make sure you use a different set of symbols for each command so that Scribe doesn't get confused.

### 6.5. Suggestions for Text Formatting

Probably the most important thing to remember in using Scribe commands is that you DON'T have to use all of them! It's better to leave it simple at first and use the fancy stuff after a little practice.

For the final word, see the Scribe User Manual, which can be found in any Athena terminal room. It is actually fairly well-written and easy to understand.

Although it does not really matter when the commands are inserted into the text, it is probably more convenient to insert the Scribe commands while you are writing your text. This will probably seem strange at first, but you will soon find yourself almost automatically inputting Scribe commands. In fact, you'll find yourself putting them in even when you're not writing Scribe files, @i(like this).

One extremely good way to learn Scribe is to look at documents which have already been written and see what Scribe commands were used to create various document results. For example, the source for this document is located in the directory /usr/unsupported/sipb/Scribe on most Athena machines.

## 7. Proofing Your Document

To avoid wasting lots of your time and perhaps money, we suggest that you always do the following before making a ''final'' copy.

### 7.1. A Speller: ISPELL

To correct spelling mistakes in your document, Athena has an interactive spelling checker called **ispell**. To use it, simply type:

```
ispell filename
```

where ''filename'' is the name of your file. Ispell goes through your file, checking your words against its own dictionary. When it comes to a word it doesn't know, it displays a list of possible spellings of the word and prompts you for a response. The important responses are:

| | |
|---|---|
| ? | Display help |
| 0-9 | Replace with one of the suggested words |
| <space> | Accept the word this time only |
| a | Accept the word for the rest of this file |
| r | Replace the misspelled word completely |
| q | Write the rest of this file, ignoring misspellings |
| x | Exit immediately. Asks for confirmation, leaves file unchanged |

Ispell also has other commands to maintain your own private dictionary. This is particularly useful, because it doesn't recognize Scribe commands. Type:

```
man ispell
```

for more information.

### 7.2. Getting Draft Copies

If you intend to print your final copy on the 9700 laser printer, remember that it costs 5 cents per page and takes a fair amount of time to get output. We strongly suggest that you print out your document on the local Athena line printer first. Certain commands will look different on the line printer. Italics, for instance, will come out underlined. Other things, such as greek characters, will not come out at all, but at least you can get a feel for what your document will look like.

## 8. Getting a Document Printed

To actually get your Scribe output, you need to go through two steps. The first step is to format the document. Doing this is very simple. First of all, you must make sure the name of your file ends in the suffix ‘‘.mss’’. If it does not, Scribe will not work properly. If you accidentally create a file which does not have this suffix, you can rename it using the ‘‘mv’’ command: You should type the following command at the terminal:

```
mv oldfilename newfilename.mss
```

and it will change the name.

To format the document, simply type:

```
scribe test.mss
```

Scribe will go to work on your file, producing output looking something like:

```
Scribe 4(1400)  Copyright (C) 1981, 1984  UNILOGIC, Ltd.
Welcome to Athena Scribe v4
*test.mss
[Processing test.mss
    [Device ‘‘LPTLN’’]
    [Document type ‘‘Text’’]
]
1.

**test.lpt for device LPTLN has 1 page.
```

Be prepared to wait a long time (maybe 20 seconds per page) because Scribe runs very slowly. If your file is called ‘‘filename.mss’’, Scribe will create a new file called ‘‘filename.lpt’’, ie, one which is exactly the same except that ‘‘.lpt’’ will be substituted for ‘‘.mss’’.

It is possible that you will have made a mistake in creating your file, and Scribe will come across a statement it doesn’t understand or notice that something is wrong. If this happens, Scribe will continue to process the file, and will put a list of errors in the file ‘‘filename.err’’. If this happens, try to figure out what the error message means, and go back and look at the documentation on Scribe to see if you can fix the mistake.

You can look at the ‘‘.lpt’’ file in the same way you look at any file. It contains your formatted text. To see it, type:

```
more
filename.lpt
```

The only step remaining now is to get it printed out. To do this, simply type:

```
lpr filename.lpt
```

This command will send the file to the laser printer in the Athena cluster where you are logged in. The file file should be printed out immediately. If it is not, typing the command:

```
lpq
```

will give you information about the current status of the printer. It might print out something like:

```
ln is ready and printing
Rank   Owner      Job  Files            Total Size
active wcs        111  nets.lpt         18819 bytes
1st    lerman     112  newplan          30456 bytes
2nd    doug       113  resume.lpt         287 bytes
3rd    yourname   114  myfile.lpt       11293 bytes
ln is ready and printing
```

This particular example would indicate that everything is working fine but that there is a backlog of requests. You are third in line with the filename ‘‘myfile.lpt’’ and as soon as the other files are printed out, your file will be. If you see the message ‘‘ready and printing’’ and the laser printer is not printing, it is probably out of paper. Check the error message on the top of the printer. If it is indeed out of paper, just refill the printer from the paper which is on the shelf next to it, or ask a consultant to do this. If the printer is not out of paper but still doesn’t function, contact a consultant.

### 8.1. Cancelling printouts

It is possible to cancel your pending print request. To cancel a request, simply type ‘‘lprm <job-number>’’, where <job-number> is the number that was printed by the lpq command. For example:

```
lprm 114
```

Will cancel the request in the previous example.

NOTE: Please do not use Athena printers to make multiple copies. They are fragile and break down often. If you need more than one copy, use a Xerox machine.

## 9. Notes on using the Xerox 9700 printer

Software has been distributed which will allow you to print your files on the Xerox 9700 printer. This is a very high quality laser printer which will let you use proportional fonts, greek characters and eventually mathematical output.

### 9.1. Formatting your file

To format your Scribe file, use the command:

```
scribe
filename.mss -device x9700
```

This will create a file called ‘‘filename.x9700’’ which is not readable by humans but which tells the 9700 which fonts to use, etc.

## 9.2. Printing your file

You should **not** use the lpr command to print your file on the 9700. To print your file, type:

```
/usr/unsupported/sipb/print_9700
```

The program will ask you several questions, including whether you want printing on one or two sides, paper with holes or no holes, etc. Make sure you print the ‘‘.x9700’’ file and not the ‘‘.lpt’’ file. Unfortunately, the 9700 is an expensive machine to run, and you will be charged **5 cents per printed side of output**. You may pick up your output on the second floor of building 11 at the end of the hall. Operators should be there from 8 until 11:30 weekdays and 10 until 5:30 on Saturdays. Note that you will not be able to get output on Sunday. If there is enough demand, there is a chance that the service will be made available on Sundays. These hours are generally extended near the end of the term. Never try to print the ‘‘.x9700’’ file on a normal printer. It will cause the printer to crash and people will get mad at you.

## 9.3. Special 9700 abilities

Greek Letters — The @g command will print output in greek characters. For instance

```
@g(abd)
```

would produce the Greek characters alpha, beta, delta. (Lowercase and uppercase work the way you would expect them to.)

Boldface — The @b command will print output in bold type.

Italics — The @i command will print output in an italic font.

Bold Italics — The @p command will print output in a bold italic font.

In general, you will format your file for the 9700 the same way you would format it for another device. Scribe will automatically choose the correct fonts.

## 9.4. Available fonts

Although most document types have use a default font which has been determined to ‘‘look good’’, it is possible to manually choose which font to use. At the moment, there are two sets of fonts available -- TimesRoman and Helvetica. These are available in both 10 and 12 point sizes. The 10 point is good for most things. You probably want the 12 point font for your papers so that they aren’t too short. The names of the fonts are:

- **TimesRoman10**
- **TimesRoman12**
- **Helvetica10**
- **Helvetica12**

The command to set the font manually must appear at the beginning of your file. It looks like:

```
@style(fontfamily =
TimesRoman10)
```

### 9.5. Limitations

At the moment, the Scribe mathematics library will not work with the 9700.

Due to the small size of the memory of the 9700, the font sets are not very complete -- ie, there is only one header font, and the small body font (used for footnotes) does not have italics or boldface. For normal use, this should not be a problem. If you desperately need various header sizes or extra typefaces, come to the SIPB office and we will look into it.

On rare occasions Scribe will not be able to format a file if there are too many fonts on one page.

A few document types do not look the same on the 9700 as they do on the ln01 and ln03. To avoid problems, we suggest that you use templates found in

**`/usr/unsupported/sipb/Scribe`**

to format your file.

## 10. Getting More Information

There are several places to go for more information:

As we said before, **the best way to learn is by example**. If you see something in this document that you do not know how to do, look at:

**`/usr/unsupported/sipb/Scribe/underground-guide.mss`**

.

Ask an Athena consultant. If the question is about using the 9700, come by the SIPB office.

Finally, there is the **Scribe User Manual**, available in documentation racks in all the terminal rooms.

## Table of Contents