CS 114 - Fall 2004

Lecture 3 - Friday, October 1, 2004

Editors: Comparative Religion 101

In this lecture, we finally create and modify files, a process known as *editing*.

A file is just a sequence of bytes. Text files are just sequences of characters split into lines.

There are two very popular editors used on Unix systems: vi and emacs. Many people have strong opinions on which editor is best. People will pontificate on the relative merits (or lack thereof) of each editor with an almost religious fervor. I'll just say that which editor you use should be a matter of personal taste.

Early Unix editors were line editors. Files are loaded into memory, edited, then saved back to the filesystem. Editing involved giving commands like: "display lines x to y", "replace line x with the following", or "delete line x". To use a line editor you have to have a mental picture of the layout of the file.

In contrast, vi and emacs display the text of the file on screen and allows you to modify it directly by moving a cursor to where you want to make changes. This is way of editing should be immediately familiar to everyone.

On most Unix systems, vi is the default editor: programs that need you to edit some text—a mail client, for example—will invoke the default editor to do so. In addition, nearly every Unix system ships with a copy of vi, and it may be the only editor available. Consequently, it's worth learning enough about vi to get around, even if you don't use vi on a daily basis. We'll discuss how to change the default editor in the next lecture. Vi has many variants, the most popular of which today is vim (vi improved).

Vi has a steep learning curve, but is small and fast (once you know what you're doing) and ubiquitous. Emacs, another popular editor, is more powerful and easier for new users to pick up, but is slower, and some would say bloated. For instance, emacs has it's own built-in email client, it's own newsreader, it's own web browser, and even it's own built-in psychotherapist.

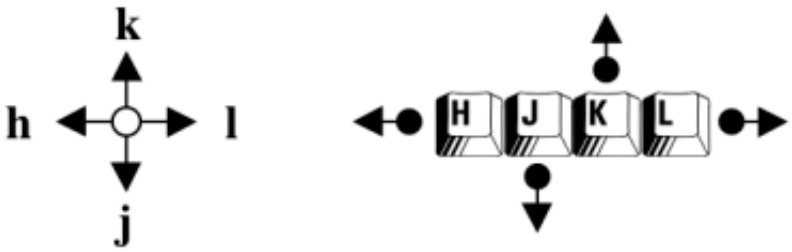
vi

Vi is a *modal* editor: the editor is always in one of two modes. In *insert mode*, you can type in text and have it appear on the screen and in the document. In *command mode* (sometimes called *beep mode* by vi's detractors), you can cut and paste text, delete text, even run the text through an external program.

```
vim.txt [unix][]                                              1/1,22 All
"vim.txt" [New] 1L, 38C written
```

Movement

Vi, by default, starts in command mode. In command mode you can navigate through the text using the following keys:



Notice that all four keys are on the home row of the keyboard, making movement fast. On more recent versions of vi, such as vim, you can use the arrow keys to move around. The following keys are also useful:

- `Ctrl-f` – page down (forward)
- `Ctrl-b` – page up (backward)
- `^` – move to the beginning of the line
- `$` – move to the end of the line
- `w` – move to the beginning of the next word
- `b` – move to the beginning of the word
- `e` – move to the end of the word
- `/`*string*`<Enter>` – search forward for *string*
- `?`*string*`<Enter>` – search backward for *string*
- `n` – repeat the last search

Insert mode

To insert text, you need to enter insert mode. Position the cursor where you want to insert text and type one the following:

- `a` – *append* after the cursor
- `i` – *insert* before the cursor
- `A` – *append* at the end of the line
- `I` – *insert* at the beginning of the line
- `o` – *open* a new line below the cursor
- `O` – *open* a new line above the cursor
- `R` – *replace* the text under the cursor

Then type the text you wish to insert. To exit insert mode and return to command mode, hit the `<Esc>` key.

Cut and paste

In command mode, you can also cut, copy, and paste text:

- `d`*movement* – delete with a movement for example, `de` deletes to the end of the word; `db` deletes to the beginning of the word; `d/`*string* deletes all text up to the string *string*, etc.,
- `dd` – delete the current line
- `D` or `d$` – delete to the end of line
- `x` – delete the character under the cursor
- `X` – delete the character before the cursor

- `y`*movement* – yank (copy) text to a buffer
- `yy` or `Y` – yank the current line
- `p` – paste after the cursor
- `P` – paste before the cursor

You can prepend any command with a number indicating how many times to repeat the command. For example, `3dd` deletes 3 lines; `5yw` yanks the next 5 words. You can also repeat insertions. For instance, `72i-<Esc>` will insert 72 dashes.

Searching and replace

You can also search and replace text with the `:s`. For example:

- `:s/`*string1*`/`*string2*`/` – replace the first occurrence of *string1* on the current line with *string2*
- `:s/`*string1*`/`*string2*`/g` – replace all occurrences of *string1* on the current line with *string2*
- `:3,12 s/`*string1*`/`*string2*`/g` – replace every occurrence of *string1* on lines 3 through 12 with *string2*
- `:1,$ s/`*string1*`/`*string2*`/g` – replace every occurrence of *string1* on every line *string2*
- `:% s/`*string1*`/`*string2*`/g` – replace every occurrence of *string1* on every line *string2*

The strings to replace are actually *regular expressions*. We'll discuss these next week.

Undo

If you make a mistake, type `u` to undo the last change. In the original vi, hitting `u` twice will undo the undo—redoing the original command. In vim, hitting `u` again will undo the command performed before the command just undone; use `Ctrl-r` to redo the command.

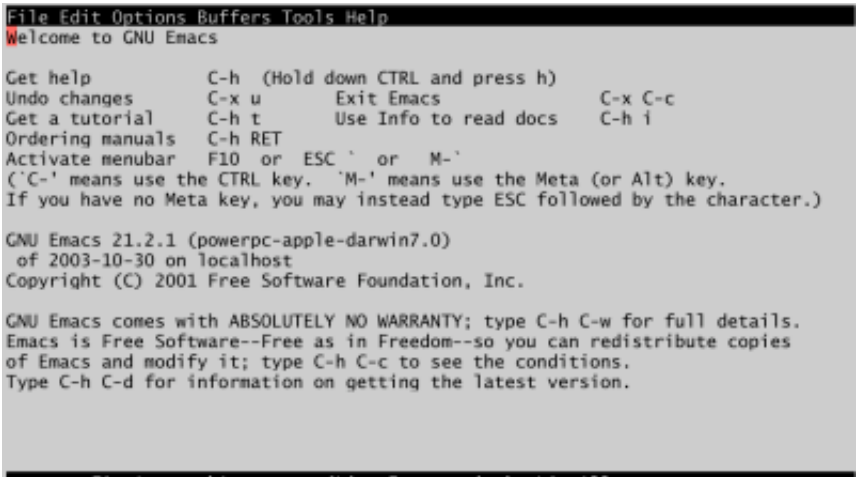
Files

The following commands deal with files:

- `:w` – writes the file
- `:w` *filename* – writes the file as *filename*
- `:w!` *filename* – writes to *filename* even if it already exists
- `:wq` – writes the file and quits
- `:q!` – quits, discarding any changes
- `:e` *filename* – opens *filename* for editing

Emacs

When you start emacs on a file, you get full screen display split into a large *edit area* where the contents of the file appears, a separator line, called the *mode line* containing information such the file name, current line number, etc., and a one-line area at the bottom of the screen called the *minibuffer*.



If you type a "normal" key such as a letter, punctuation, a number, it gets inserted at the position of the cursor in the edit area. Other editor commands are performed using various key combinations. The notation for these combinations is:

- `C-x` means pressing the `CONTROL` key and the `x` key.
- `M-x` means pressing the `META` key and the `x` key.

What's the `META` key? Some keyboards actually have a `META` key, but most do not. `META` is sometimes mapped to the `ALT` key, but more often is mapped to `<Esc>`. If `META` is mapped to `<Esc>`, to do `M-x`, press `<Esc>` and release, *then* press `x`.

To move the cursor around, you can sometimes use the arrow keys, but it may depend on your system. Instead, the following keys are used:

- `C-f` (forward) – move the cursor right one character
- `C-b` (back) – move the cursor left one character
- `C-p` (previous) – move the cursor up one line
- `C-n` (next) – move the cursor down one line
- `C-v` – move the cursor one page down
- `M-v` – move the cursor one page up

Other key combinations that are useful are:

- `C-l` – refreshes the display and centers the text on the cursor position.
- `C-s` – starts a forward search from the cursor position for a piece of text. The minibuffer will ask you to enter the text to search for.
- `C-r` – searches backward from the current position.
- `C-g` – interrupts the current operation.

There are also double key combinations. If you press a key prefix such as `C-x`, emacs will wait for you to press another key to execute the corresponding command. If you change your mind, or you mistype, you can press `C-g` to get out.

The following key combinations are useful:

- `C-x C-s` – saves the file to disk using the same name as it was loaded
- `C-x C-w` – prompts for a file name in the minibuffer and saves with that name.
- `C-x C-c` – quits emacs. It will prompt you to save any unsaved changes.

Emacs buffers

In emacs, you can edit more than one file at a time. This introduces the notion of a *buffer*. A buffer is typically associated with a file you are editing, although there are other ways to get buffers.

Once emacs is started, you will be editing in the *scratch buffer* You edit a new file by doing:

- `C-x C-f` – loads a file from disk, or creates if it doesn't exist

This creates a new buffer to edit that file. The previous file you were editing is still loaded, but it is hidden. Every buffer has a name. When the buffer is associated with a file, the buffer name is just the file name. To change to another buffer, the following commands are useful:

- `C-x b` – changes to another buffer. Emacs will ask you in the minibuffer for the name of the new buffer. You can just hit `<Enter>` to accept the default choice.
- `C-x C-b` – creates a new buffer containing the list of all the buffers emacs currently has.
- `C-x o` – when more than one buffer is on screen, this key moves the cursor to the next buffer.
- `C-x 1` – when more than one buffer is on screen, this key makes the buffer containing the cursor

a full-screen buffer.
- `C-x k` - kills a buffer. If the buffer contains unsaved data, emacs will ask if you want to save it to a file.

Functions

Everything in emacs is ultimately done by commands, or *functions*. At startup, keys are bound to a function. For example, the key `C-f` is bound to the function `forward-char`; the key `C-x C-w` is bound to the function `write-file`. There are many more functions than there are key combinations, and it is possible to change the function to which any given key is bound.

You can invoke a function by name:

- `M-x` - asks in the minibuffer for the name of a function to call

Since this is so common, we use the notation `M-x function-name` to represent the call of a function function-name from emacs. Note that emacs has function name completion: if you start to type a function name in the minibuffer and then hit TAB, emacs will try to complete the function name or give you a list of possible completions in the buffer.

Some useful functions to know about are:

- `M-x describe-key` - asks for a key (by pressing it) and creates a new buffer containing information on that key, including the function it is bound to, if any.
- `M-x describe-function` - asks for a function name and creates a new buffer containing information on that function.
- `M-x apropos` - asks for text to each for in all the function descriptions and returns the function names that match.
- `M-x help-with-tutorial` - starts an useful tutorial on the use of emacs that I advise you to go through at least once.
- `M-x doctor` - starts the emacs psychotherapist.

You can also write your own commands. Unfortunately, you'll have to learn the programming langauge Emacs LISP to do so.

Minor religions

Unix has several other editors available. One popular choice is Pico. Pico is the editor used by the email client Pine. It's easy to use, but not particularly powerful.