

Working With Editors

One of the most fundamental tools required for application development on any platform is a text editor; and the Linux operating system offers programmers a wide variety to choose from. These editors offer a wide variety of functionality from simple editing features to syntax highlighting and reading e-mail.

In this chapter, we will focus on a couple of editors that offer features that will be of interest to developers, *Jed*, *vim*, and *Emacs*. All of these editors offer extended features beyond simple text editing.

2.1 What to Look for in an Editor

While editors like *pico* or even *ed* may be useful for editing system files or writing quick notes, programming editors have certain functions that permit the programmer to focus on the act of creating code and helps to manage the process and keep an eye on syntax.

2.1.1 Extensibility

One useful feature of programming editors is the ability to extend the out-of-the-box functionality with custom programming.

Emacs utilizes a full-featured Lisp language called *Elisp* to provide users with the ability to add just about any functionality they might require to the editor. The original Lisp language was written in the late 1950s as part of MIT's research into artificial intelligence. *Elisp* is derived from the original Lisp and provides surprising flexibility in extending *Emacs* for those who take the time to learn it.

Jed extensibility is based on the *s-lang* libraries (www.s-lang.org) that were developed by John E. Davis as a generic macro language that would become the basis for a number of different programs. *S-lang* programs resemble C in syntax and layout.

2.1.2 Understanding Syntax

By understanding the syntax of the programming, the editor can perform a number of functions that make it easier for the programmer to move through the code, locate bugs more quickly and perform other common functions.

Such functions include jumping to the beginning of a stanza in the code, highlighting that stanza, automatic indenting, highlighting syntax, and quickly inserting comments and commenting on entire sections of code.

2.1.3 Tag Support

The *ctags* and *etags* utilities that come with Linux are able to build a list of the various functions, classes, fragments, data blocks and other information within the various files that make up the application. Not all information is available for all languages. For example, while only subroutines are supported for the Perl language, support for the C/C++ languages includes:

- macros (names defined/undefined by `#define / #undef`)
- enumerators (enumerated values)
- function definitions, prototypes, and declarations
- class, enum, struct, and union names
- namespaces
- typedefs
- variables (definitions and declarations)
- class, struct, and union members

Both Emacs and vim provide the ability for the editor to understand standard tag files and help the programmer quickly locate and edit the portion of code that he/she is working on. Emacs uses the output from *etags*, while vi uses *ctags*.

2.1.4 Folding Code

Folding code refers to the ability of the editor to hide portions of code while they are not needed. For example, all of the functions or subroutines in the code can be folded so that only the names of the functions are seen until the programmer unfolds that routine to work on it.

2.2 Emacs

Emacs is a lisp-based editor that has been around in one form or another since the late 1970s; despite its long history, it remains one of the most up-to-date and widely used editing environ-

ments today. From within the editor a user can read and send mail, perform research on the Internet, and even send out for coffee to RFC2324 compliant coffee makers.

Configuration of Emacs is accomplished via the `.emacs` file that can be created in each user's `$HOME` directory. If the configuration does not exist, Emacs will use its built-in defaults. This file consists of a series of elisp expressions that the editor evaluates when the application runs.

2.2.1 Using Emacs

Navigating and using Emacs may appear confusing and counter-intuitive for those just learning how to use the editor, however the time taken to master this editor can prove well-spent when one considers the time savings that such an integrated and flexible development environment can provide.

If Emacs is started from within X-Windows, it will automatically open a new window in the desktop. To prevent this behavior, you can use the `-nw` option from the command line. Alternately, you can tell Emacs how large to make the window by specifying the size of the window, in characters, that you wish to create. Examples are:

```
$ emacs -nw main.c
$ emacs -geometry 80x24 main.c &
```

Figure 2-1 shows the initial Emacs screen if it is invoked without an initial file name.

```

Buffers Files Tools Edit Search Mule Help
Welcome to GNU Emacs, one component of a Linux-based GNU system.
The menu bar and scroll bar are sufficient for basic editing with the mouse.

Useful Files menu items:
Exit Emacs                (or type Control-x followed by Control-c)
Recover Session           recover files you were editing before a crash

Important Help menu items:
Emacs Tutorial            Learn-by-doing tutorial for using Emacs efficiently.
(Non)Warranty             GNU Emacs comes with ABSOLUTELY NO WARRANTY
Copying Conditions        Conditions for redistributing and changing Emacs.
Getting New Versions      How to obtain the latest version of Emacs.

GNU Emacs 20.7.1 (i386-redhat-linux-gnu, X toolkit)
of Fri Mar 16 2001 on porky.devel.redhat.com
Copyright (C) 1999 Free Software Foundation, Inc.

-1:-- *scratch*          (Lisp Interaction)--L1--All-----
For information about the GNU Project and its goals, type C-h C-p.
```

Figure 2-1 The initial Emacs screen.

For a complete tutorial in using Emacs, from within the application itself, type **^H-t**¹. This tutorial covers many of the functions that are available within Emacs and takes you step-by-step through them.

2.2.2 Basic Emacs Concepts

Emacs uses several keyboard keys to enter commands into the application. The primary one is the Meta key. The Meta key is signified by M-. The Meta key is generally the ALT key on the keyboard, although it may be the ESC key in certain circumstances. If the ALT key does not work, try the ESC key. What works will depend on if you are logged in locally, accessing the console directly or using X-Windows. The ALT key is used in the same manner as the CTRL key. When using the ESC key, press and release the ESC key and then press the next indicated key. In all cases, typing ^U may be used in place of the Meta key. Just type and release ^U and then type the rest of the key sequence as you would for the ESC key.

Entering text into the buffer is accomplished in the default mode simply by typing on the keyboard. To abort a command that's currently running or asking for input, type ^G. This will return you to the current buffer.

Simple commands can be repeated by prefacing them with the key sequence **ESC #**. By pressing the escape key and typing any number, the next command that is issued will be repeated that number of times. For example, typing **ESC 75=** is the same as pressing the equal key 75 times.

To exit Emacs, use the command sequence **^X^C**.

Moving around

In addition to the basic functionality provided by the arrow keys on the keyboard, the key combinations shown in Table 2-1 may be used to move the pointer one character at a time in a given direction.

Table 2-1 Simple Movement Commands.

Arrow Key	Alternate Combination
Left Arrow	^F
Right Arrow	^B
Up Arrow	^P
Down Arrow	^N

1. The caret symbol denotes a control character. To enter the key combination ^H-t, press and hold the CTRL key, and then press the 'H' key. Release both keys and then press the 't' key.

Movement can also be accomplished a word or a page at a time. This is accomplished by referring to Table 2-2.

Table 2-2 Movement Commands

Action	Command
M-b	Word Left
M-f	Word Right
^A	Beginning of Line
^E	End of Line
M-c or <i>Page-Up</i>	Page Up
^V or <i>Page-Down</i>	Page Down
M-<	Beginning of Buffer
M->	End of Buffer

Deleting

Just as there are several ways to move around in the buffer, there are several ways to quickly delete unwanted information. The <BACKSPACE> key or ^H can be used to delete the character before the pointer. By using ^D you can delete the character at the pointer.

In addition, there are ways to delete words, lines and portions of lines. There are also methods for blocking text and deleting entire sections of the buffer. For more information on working with blocks of text, see *Cutting and Pasting* below.

Table 2-3 shows the various commands used to perform these functions.

Table 2-3 Movement Commands

Action	Command
<BACKSPACE> or ^H	Delete character to the left
^D	Delete character to the right
M-DEL	Delete word to the left
M-d	Delete from pointer to end of current word
^A^K	Delete current line excluding the EOL character
^K	Delete from pointer to end of line
^X u	Undo previous command – may be used multiple times

File Operations

Editing within Emacs is done through a series of buffers. Each buffer may be edited separately and the changes saved back to the disk. Files may also be opened into new or existing buffers as shown in Table 2-4.

Table 2-4 File Commands

Action	Command
^X ^F	Open File
^X k	Close File
^X i	Insert File
^X ^S	Save File
^X w	Save File As
^X b	Change Buffer

Search and Replace

There are two search and replace functions within Emacs. The first one simply does a simple search (case insensitive by default) for the character string provided by the user. The second, more complex search function does a search and replace on regular expressions. See Table 2-5 for a list of these commands.

Table 2-5 Search and Replace Commands

Action	Command
^S ENTER	Search
^S	Continue Forward Search
^R ENTER	Search Backwards
^R	Continue Backwards Search
M-%	Search & Replace
M-X query-replace-regexp	Regular Expression (regex) Search & Replace
M-X occur	Find occurrences of a string in the current buffer

By typing `^S <ENTER>` you will be prompted to enter the string to search for. With the search string still active, typing `^S` again to search for the next occurrence of that string in the current buffer.

Typing `M-%` (usually accomplished by `ESC-%` rather than `ALT-%` under Linux) will bring up the same search function, but when you press `RETURN` after entering the search key, you will be prompted for a replacement string. Type in the string that you wish to replace the search key with and press `RETURN`. If the search string is found in the current buffer, you will be presented with the options shown in Table 2-6.

Table 2-6 Search and Replace Options

Action	Command
y or SPACE	Replace the string at the pointer with the replacement string and search for the next occurrence.
n or DEL	Leave the string at the pointer as is and search for the next occurrence.
!	Replace globally from the pointer forward in the buffer.
.	Replace the string at the pointer and then exit search and replace mode.
^	Move point back to previous match.
u	Undo the previous replacement.
q or ENTER	Exit search and replace mode.
?	Display help.

The more complex search and replace feature is available by default, only from the prompt and is not bound to a particular key combination. To access this feature, you need to type in the name of the mode, which in this case is “`query-match-regex`”. The complete key sequence for this is:

```
M-X query-replace-regex <RETURN>
```

This command brings up a similar series of prompts that allows you to search for regular expressions in the current buffer and, using the same options shown in Table 2-5, replace them.

Emacs has an auto-completion option that you can use instead of typing the entire command shown above. By typing:

```
M-X que<ESC>
```

Emacs will search through its listing of modes and complete as much of the request as it can. If there is a conflict and there are one or more modes that begin with the phrase that you

have typed in, pressing the <SPACEBAR> will cycle through the names. You can press <RETURN> to select the one currently displayed.

Emacs supports the use of parenthetical statements in regex search and replace commands. A portion of the search string may be used as part of the replacement string. The contents of the first set of parenthesis in the search string may be referenced as \1 in the replacement string. The second set would be referred to by \2.

For example:

```
Original string:The Dodo and the Griffin
Search string:\([Tt]h\)e \([a-zA-Z]*\)
Replacement string:\1ose \2s
New string:Those Dodos and those Griffins
```

Cutting and Pasting

Sections of the buffer may be marked and certain commands may be applied to those regions. These commands include copying, cutting and pasting text. To select a region of text, move the pointer to the beginning of the sections that you wish to select and press ^<SPACEBAR>. When the pointer is moved, the text from the marked point to the current pointer position will be highlighted. Once the region is selected, issue the cut or copy command. To deselect the text without issuing the copy or paste command, simply press ^<SPACEBAR> again.

Table 2-7 shows a list of the various selection commands.

Table 2-7 Cut and Paste Commands

Action	Command
^<SPACEBAR>	Begin selection
^W	Cut
M-W	Copy
^Y	Paste

2.2.3 Using Buffers and Windows

As mentioned, Emacs has multiple buffers that can be opened and edited simultaneously. To navigate between buffers, press ^**Xb** and type the name of the buffer that you wish to switch to. Buffers may be opened and closed by using the File Commands listed earlier in this chapter. To see a list of buffers, use ^**X^B**.

As shown in Figure 2-2, each window may be split into two windows by using the ^**X2** command. This will create a horizontal divide in the middle of the current window. The same file will be present in each pane, but they may be scrolled separately. To move between windows, press ^**Xo**. Windows may be split multiple times. To close all other buffer, use ^**X1**. The current buffer may be closed with ^**X0**.

See Table 2-8 for a complete list of window commands and Figure 2-2 for an example of using multiple buffers. These buffers are 'main.c' and 'temp.c'.

Table 2-8 Window Commands

Action	Command
^Xb	Switch to buffer
^X^B	List buffers
^X2	Split current window
^Xo	Move to other window
^X0	Delete current window
^X1	Delete all over windows

```

Buffers Files Tools Edit Search Mule C Help
void main ( int argc, char **argv) /*{{{ Main Function */
{
  while ( $a != 10 ) /*{{{ Inner Loop 1 */
  {
    $a++;
  } /*{{{ */

  while ( $a != 0 ) /*{{{ Inner Loop 2 */
  {
    $a--;
  }
}

--:-- temp.c (C)--L1--Top-----
#include <stdio.h>

int main ()
{
  printf('Hello World\n');
}

--:-- main.c (C)--L1--All-----
main.c has auto save data; consider M-x recover-file

```

Figure 2-2 Using multiple buffers in Emacs.

2.2.4 Language Modes

Emacs recognizes a number of different programming language files based on their extensions. When you load a recognized source code file, Emacs will assume certain defaults and enter the appropriate mode for editing that file.

For example, when you load a file with a .c extension for editing, Emacs sets the appropriate mode and enables commands used to automatically and manually indent code, quickly move through functions and insert comments.

When a language mode is on, Emacs can automatically indent code as you type. To turn this mode on, type `^C^A`. The same command is used to turn this mode back off.

With this mode active, auto-indenting takes place when certain characters are entered from the keyboard. These characters are the semi-colon, curly braces, and under certain circumstances, the comma and colon.

For example, if auto-indent (or technically **c-toggle-auto-state**) is on and the following code is typed into the buffer:

```
void main ( int argc, char **argv) { while (
```

it will be formatted by Emacs as follows:

```
void main ( int argc, char **argv)
{
    while (
```

Table 2-9 shows some of the common C-mode commands.

Table 2-9 C-mode Commands

Action	Command
ESC ;	Insert comment
ESC ^A	Go to top of function
ESC ^E	Go to bottom of function
ESC ^H	Mark function
{	Insert bracket and return
}	Return and insert bracket
^C^A	Toggle Auto-indent mode
^\ ^_	Auto-indent selected region

2.2.5 Using Tags

As an application grows in size, it also grows in complexity. As the number of subroutines, variables, functions and files increases, it becomes much more difficult to keep track of every piece and to quickly find the portion of code that one needs to work on. To address this issue, Emacs has the ability to read a file that contains a table of tags that reference various parts of an application.

These tags are stored in a TAGS file that is created by the `etags` command. Once this file is built, Emacs can read this table and use it to quickly locate a specific portion of code, regardless of whether it is in a currently open file or not.

From the command line the `etags` command is issued with a list of file names to be read:

```
$ etags *.ch]
```

This command will read all files in the current directory that have a `.c` or `.h` extension and build a tags table file in the current directory. The output file is, by default, called TAGS.

To build a single TAGS table for a project that has source code spread through any number of subdirectories, the following command may be used:

```
$ find . -name \*.ch | xargs etags -
```

Just be aware if there are too many files, then this command may need to be run several times with varying arguments in order to build a complete table. To do this, you must use the `--append` option to the `etags` command as shown:

```
$ find . -name \*.c | xargs etags -
$ find . -name \*.h | xargs etags --append -
```

Any of these commands may be issued from within Emacs itself by using the `M-!` command to issue a shell command. Simply type `ESC ! <command name>` and press return.

Once you have built a TAGS table, it must first be read into Emacs before you can use it to search. To accomplish this, type `M-x visit-tags-table`, specify the location of the TAGS file to be read in, and then the name of the TAGS file. The default value for the location is the current working directory, and the default tags file name is "TAGS".

Once the TAGS file has been read in, you may issue search commands against the table. There are several commands that can be used. The one most commonly used is `ESC .` which searches for a tag that matches the search parameter. The default search parameter is the word at the current pointer position. For example, if the pointer were on the character string `search_function`, the default search string that Emacs presents would be `search_function`.

If you are not sure of the name of the function that you are looking for, you can type the first few characters of the function name and press the `TAB` key. This invokes the completion function of Emacs and, if the function is unique, it will display that function name. If the function name is not unique, Emacs will complete as much of the function name as it can and then prompt you for the rest. Pressing `TAB` again after Emacs has completed as much of the function

name as it can and will display the matching functions in a new buffer. If, for example, you wanted to edit the `close_files` function, Figure 2-3 shows the results of typing `ESC . c<TAB><TAB>`.

If Emacs finds a function that matches your search string, it will replace the current buffer with the contents of the first file in which it found the search string and the pointer will be positioned at the first line of that function. In the above example, completing the file name and pressing `ENTER` results in the file `exit.c` being opened and the pointer being positioned on the first line of the `close_files` function. This is shown in Figure 2-4.

Alternatively, you can initiate the search with the command `ESC x find-tag-other-window` and rather than replacing the current buffer with the found function, a new buffer will be opened instead. Remember that Emacs has a completion function, so after typing the first few characters of a function, you can press the `TAB` key and Emacs will fill in the rest of the command if it is unique. If you are sure that a command is unique, pressing the `ENTER` key will execute that command.

Rather than searching for functions by name, you can also search all of the files referenced in the tags file for a regular expression. To accomplish this, type `ESC x tags-search` and Emacs will prompt you for a regular expression to search for. If it finds a match, the first occurrence of the string found will be displayed in the current buffer. You can search for the next occurrence of the string by issuing the `ESC ,` command.

```

Buffers Files Tools Edit Search Mule Minibuf Help
}

/* Given the mask, find the first available signal that should be serviced. */
static int
next_signal(struct task_struct *tsk, sigset_t *mask)
{
    unsigned long i, *s, *m, x;
    int sig = 0;
--1;***F1 signal.c (C)--L50-- 4%-----
In this buffer, type RET to select the completion near point.

Possible completions are:
calc_load                call_usermodehelper
can_schedule              cap_bset
cap_emulate_setxuid      cap_set_all
cap_set_pg                cascade_timers
check_free_space          check_resource
child_reaper              close_files
collect_signal            console_cmdline
--11;--F1 *Completions* (Completion List)--L1--Top-----
Find tag: (default int) c

```

Figure 2-3 Emacs tags-search function.

```

Buffers Files Tools Edit Search Mule C Help
/* We dont want people slaying init */
p->exit_signal = SIGCHLD;
p->self_exec_id++;
p->p_opptr = reaper;
if (p->pdeath_signal) send_sig(p->pdeath_signal, p, 0);
    }
}
read_unlock(&tasklist_lock);
}
static inline void close_files(struct files_struct * files)
{
    int i, j;

    j = 0;
    for (;;) {
        unsigned long set;
        i = j * __NFDBITS;
        if (i >= files->max_fdset || i >= files->max_fds)
            break;
        set = files->open_fds->fds_bits[j++];
--1:--F1 exit.c (C)--L177--27%-----
Mark set

```

Figure 2-4 Finding the function.

Instead of searching for a specific regular expression, the command **ESC tags-apropos** will create a new buffer entitled **Tags List** that contains a listing of all of the tags that match that regular expression. By moving the pointer around this buffer, you can place it on the function that you are looking for and use the **ESC .** command to open that function in the current buffer. A list of TAGS commands is shown in Table 2-10.

Table 2-10 Emacs commands related to TAGS

Action	Command
M-x visit-tags-table	Load a tags file into Emacs
M-.	Search for a function
M-x find-tag-other-window	Search for a function and load the file in a new buffer
M-x tags-search	Search for a regular expression in the files represented by the current tag list
M-,	Repeat regular expression search
M-x tags-apropos	Load a list of all tags into a new buffer

2.2.6 Compiling

Emacs has the facility to call external compilers, parse their output and display the results in a buffer. As the results are displayed, the programmer can then move forward and backward through any error or warnings. As each error or warning is displayed, the appropriate line in the code is displayed and may be edited.

To invoke the compiler from Emacs, type `M-X compile`; in response, Emacs will ask you for the command to use to compile the program for application. You can either specify `make` or the command line compiler with all the options. The default is to invoke `make` with the `-k` option in order to continue as far into the `make` as it can when encountering errors.

For example, assume that the following (broken) bit of code is in a buffer entitled 'main.c'.

```
#include <stdio.h>

int main ()
{
    printf('Hello World\n');
}
```

The compiler may be invoked by typing:

```
M-X compile
Compile command: gcc -o main main.c
```

If the buffer being compiled has not been saved, the editor will prompt you to save it. The results of the compilation will appear in the `*compilation*` buffer as seen in Figure 2-5.

```
Buffers Files Tools Edit Search Mule C Help
#include <stdio.h>

int main ()
{
    printf('Hello World\n');
}

--1:--F1 main.c (C)--L1--All-----
cd ~/
gcc -o main main.c
main.c: In function `main':
main.c:5: character constant too long
main.c:5: warning: passing arg 1 of `printf' makes pointer from integer without
a cast

Compilation exited abnormally with code 1 at Sun Jun 30 15:56:30

--1:**F1 *compilation* (Compilation:exit [1])--L1--All-----
(No files need saving)
```

Figure 2-5 The results of `M-X compile`.

The main.c buffer is still the active buffer, but it is linked to the *compilation* buffer. As indicated by Figure 2-6, typing `^X-`` the first error in the bottom buffer is highlighted and the pointer in the top buffer is positioned in the line of code that caused the error. Repeating that command moves the pointer forward in the code to the next error.

```

Buffers  Files  Tools  Edit  Search  Mule  C  Help
#include <stdio.h>

int main ()
{
  printf('Hello World\n');
}

--:-- main.c (C)--L5--A11-----
main.c:5: character constant too long
main.c:5: warning: passing arg 1 of `printf' makes pointer from integer without\
a cast

Compilation exited abnormally with code 1 at Sun Jun 30 16:51:21

-1:** *compilation* (Compilation:exit [1])--L4--Bot-----
Parsing error messages...done.

```

Figure 2-6 Using the built-in functions to debug code.

While navigating through the *compilation* buffer, typing `^C^C` will move the code in the window to the line referenced by the error message. Table 2-11 lists the commands used to aid in debugging compiled code.

Table 2-11 Compiling with Emacs

Action	Command
M-X compile	Compile a file or an entire project
<code>^X-`</code>	Find the next error
<code>^C^C</code>	Examine source code that created the error

2.2.7 Xemacs

Xemacs is a project that was spawned from the original source tree of the Emacs project. In many ways it still resembles its ancestor, but when it is run it includes additional features that make use of the X-Window interface. These include toolbars, font faces, image embedding and editing, and similar features.

As you can see in Figure 2-7, the Xemacs interface has taken many commonly used functions and created a tool bar across the top of the display that makes them accessible to the mouse. In addition, most of the user interface is customizable.

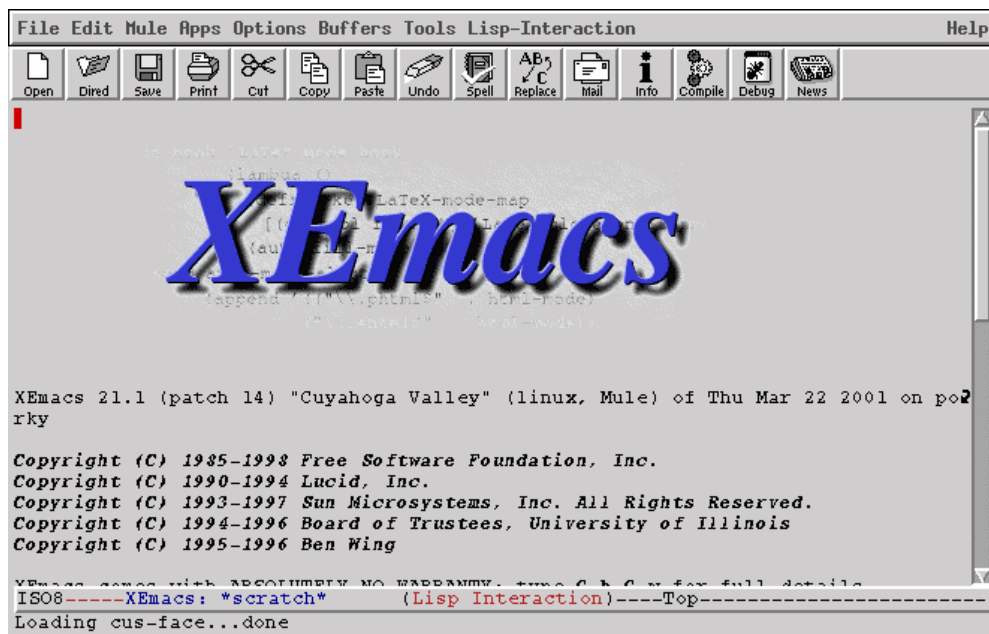


Figure 2-7 The Xemacs interface.

2.3 Jed

Jed was designed as a small, lightweight editor. It has many of the functions required by application programmers and can be set up with one of several different key bindings to aid people transitioning from other editors.

The main Jed configuration file is in JEDROOT/lib/jed.rc (/usr/share/jed/lib/jed.rc if the RedHat RPM was used to install the application). This file contains the default values for all users, but it is only read if the user does not have a local configuration file present in the home directory.

2.3.1 Configuring Jed

Individual users may change their default setting by copying `JEDROOT/lib/jed.rc` to `.jedrc` in their home directory:

```
$ cp /usr/share/jed/lib/jed.rc ~/.jedrc
```

This file may be edited to change the initial values and behavior of Jed. Lines beginning with a percent symbol (%) are comments and are ignored when the file is read. Other than conditional statements, all entries must end in a semi-colon (;).

One of the first options that a user may wish to change is the emulation mode of the editor. By default Jed uses Emacs-like key bindings for entering commands. Some other emulation modes available are IDE, CUA and even WordStar. To select a new emulation, edit the `.jedrc` in the user's home directory, comment out the current emulation and uncomment the one that you wish to use.

Below, the user has changed the application to use the IDE mode instead of the default. These key bindings resemble those used by in Borland's IDE.

```
if (BATCH == 0)
{
% () = evalfile("emacs");    % Emacs-like bindings
% () = evalfile("edt");      % EDT emulation
  () = evalfile ("ide");      % Borland IDE
% () = evalfile ("brief");    % Brief Keybindings
% () = evalfile("wordstar"); % Wordstar (use ide instead)
% () = evalfile ("cua");      % CUA-like key bindings
...
}
```

You will also notice that there is a conditional statement in the example above. This is because Jed may also be run in batch mode for processing files unattended. Statements within this block will only be processed if the application is run in interactive mode, not when run in batch mode. While having the statements outside of this block would not effect the application when run in batch mode, having them separated speeds up the load time of the application when they are not needed.

There are many other configuration options available in the `.jedrc` file that control how the program operates. Some of them are generic to all modes and others are used in only one mode. For example, the variable `CASE_SEARCH` may be set to force case sensitivity in searches, or `C_BRA_NEWLINE` may be set to force a newline character to be inserted prior to a curly-bracket when in C-mode.

Jed also has the capability of calling a compiler directly and examining the output. The standard compiler is assumed to be `gcc`. If you are using a different compiler, you will need to set the `Compile_Default_Compiler` variable in the `.jedrc` file.

2.3.2 Using Jed

Jed is called from the command line with an argument telling it which file you would like to edit. If Jed is called without an argument, it will prompt you for the name of the file before you may do any editing. This behavior may be changed by modifying the `Startup_With_File` variable in `.jedrc` to have a value of 0.

Jed may be called with one or more command line arguments. A `-n` argument forces Jed to ignore the users' local `.jedrc` file as well as the `jedrc` file. There is also an X version of Jed that allows you to use the mouse to select text and options from the menu. To start Jed, simply type:

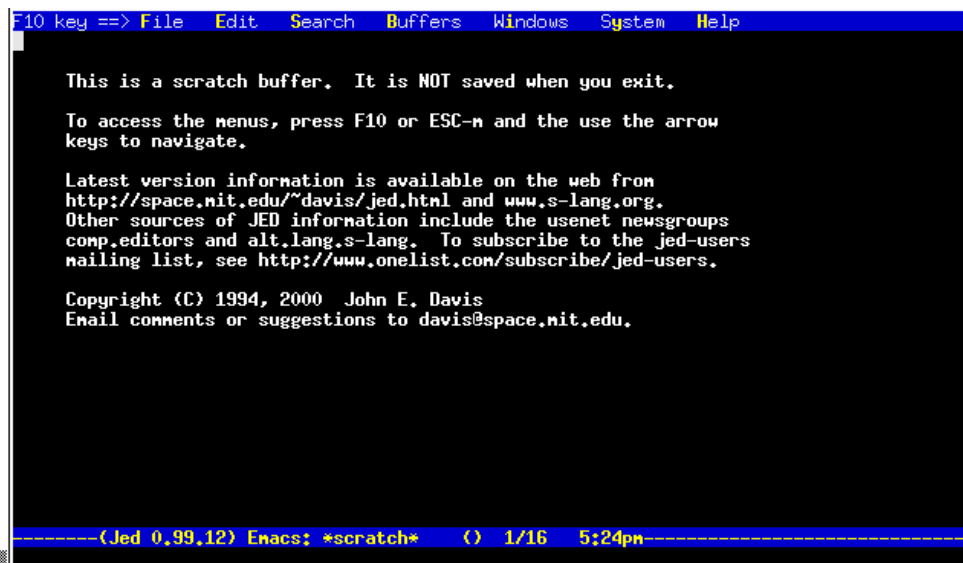
```
$ jed <filename>
```

or

```
$ xjed <filename>
```

Figure 2-8 shows the initial text-based Jed display when no command-line argument is given. The menu across the top is accessed by pressing the F-10 key. This scratch buffer will disappear when you begin to type and if a filename is given on the command line you will be taken immediately to that buffer to begin editing.

The basic editing features of Jed will be dependent upon the emulation mode that is selected. In this section, it is assumed that you will be using the Emacs emulation mode.



```
F10 key ==> File Edit Search Buffers Windows System Help
This is a scratch buffer. It is NOT saved when you exit.
To access the menus, press F10 or ESC-n and the use the arrow
keys to navigate.
Latest version information is available on the web from
http://space.mit.edu/~davis/jed.html and www.s-lang.org.
Other sources of JED information include the usenet newsgroups
comp.editors and alt.lang.s-lang. To subscribe to the jed-users
mailing list, see http://www.onelist.com/subscribe/jed-users.
Copyright (C) 1994, 2000 John E. Davis
Email comments or suggestions to davis@space.mit.edu.
----- (Jed 0.99.12) Emacs: *scratch* () 1/16 5:24pm -----
```

Figure 2-8 The Jed text interface.

2.3.3 Folding Code

The ability to fold buffers in order to view only the parts of a file that are necessary at the time is quite useful when trying to understand program flow and how the application is put together at a very high level.

In order for folding to be used, special markers must be put in the file at the appropriate points. There are two markers, one to denote the beginning of a section to be folded and one to denote the end. In plain text files, these markers must be in the leftmost column; but when an application language that is recognized by Jed is used, they may be inserted in comments.

The opening marker is three left curly braces “{{{” and the closing is three of the right “}}”. To insert marker into a C-language program, place the markers directly after the /* that begins the comment:

```
/*{{{ Deep magic begins here. */
{
    x = n[i];
    a = ((a<<19)^(a>>13)) + n[(i+128)&255];
    n[i] = y = n[x&255] + a + b;
    r[i] = d = n[(y>>8)&255] + x;
}
/*}}} */
```

When the file is folded up, the above section of code will appear as follows:

```
/*{{{ Deep magic begins here. */...
```

The ellipsis at the end of the line indicates that this line may be unfolded.

Figure 2-9 shows a section of Perl code with the fold markers in place. The comments are placed before and after each section of code and a description of the code included in that fold has been added. Figure 2-10 shows the same file folded.

```
F10 key => File Edit Search Buffers Windows System Help
#i=1;
#}}}
#{{{ Read in source.conf configuration file and parse it
open(SOURCE, "./source.conf") || die 'Can not open input file';
chop(@source = <SOURCE>);
close(SOURCE);

@defs=grep(/^#define/, @source);
foreach (@defs) {
    ($junk, $var, $val) = split(/\s/, $_, 3);
    $DEF{$var} = $val;
}
#}}}

#{{{ Print out the HTML HEADER
HEADER: {
    print "<HTML>\n<HEAD><TITLE>$DEF{title}</TITLE>\n</HEAD>\n";
    print "<BODY BACKGROUND=\"$bg$DEF{background}>\n";
    print "<center>\n";
    print "<table cellpadding=$DEF{cellpadding} border=$DEF{border}>\n";
}
#}}}

#{{{ Main body of the app
**-----+(Jed 0.99.12) Emacs: gallery.pl (perl) 14/94 11:04pm
```

Figure 2-9 The file with the folding markers in place.

```

F10 key ==> File Edit Search Buffers Windows System Help
#!/usr/bin/perl

#### Static Variable Definitions...
#### Read in source.conf configuration file and parse it...
#### Print out the HTML HEADER...
#### Main body of the app...
#### Print out the HTML FOOTER...

sub makepage {
#### ...
}

**-----*(Jed 0.99.12) Emacs: gallery.pl (perl) 3/94 11:07pm-----
Folding buffer...done

```

Figure 2-10 The same file folded.

The entire program can be folded and unfolded at once, or individual sections of the file may be unfolded. Several sections from different parts of the program may be unfolded at once.

Jed treats each folded section as a subroutine and it can be edited as such. By moving the pointer to a folded section and pressing `^C>` only that section of the program is displayed for editing.

In order to use the folding mode, it must be activated from within Jed. In order to do this, type `M-X folding-mode <RETURN>`. This will turn the folding mode on and immediately fold up the current buffer.

See Table 2-12 for a list of the available commands in this mode.

Table 2-12 Folding Mode Commands

Action	Command
<code>^C^W</code>	Fold current buffer
<code>^C^O</code>	Unfold current buffer
<code>^C^X</code>	Fold current marked section
<code>^C^S</code>	Unfold current marked section
<code>^C^F</code>	Fold highlighted section
<code>^C></code>	Edit folded section
<code>^C<</code>	Exit current section being edited

2.4 VIM

VIM stands for Vi IMproved and was developed by Bram Moolenaar. It is based on the functionality of the original, non-open source vi editor. While most open source software is also free-ware, VIM is distributed as Charityware. In exchange for using the program, the authors request that users consider donating to the Kibaale Children's Center (KCC), a charity providing food, health care and education for the children in the area. For further information regarding this donation program and the KCC, within VIM, type **:help ifcc** or visit <http://www.vim.org/ifcc>.

2.4.1 Using VIM

VIM is available in both text-based and graphical modes. The graphical version, shown in Figure 2-11, has the same functionality as the text-based version, but also provides easy access to many functions through pull-down menus and a button bar. To start the text-based version, use the command **vim**. The graphical version is started by typing **gvim**. For example, to start VIM and edit the file main.c, type the following:

```
$ vim main.c
```

Or, for the graphical version, type:

```
$ gvim main.c
```

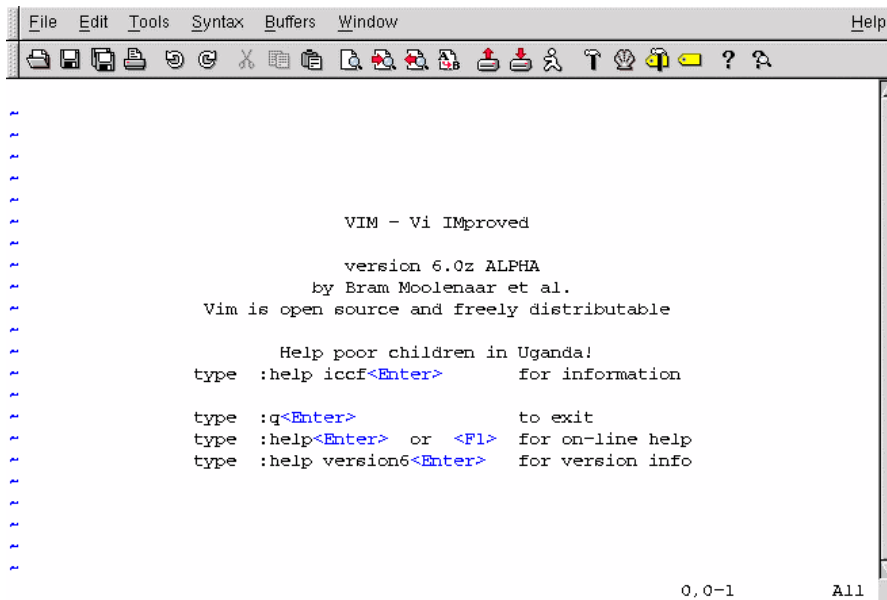


Figure 2-11 gvim.

2.4.1 VIM Concepts

There are two basic modes within VIM that determine VIM's behavior. These two modes are Normal and Insert. In Normal mode, the editor expects the user to enter commands that perform a number of functions. In Insert mode, the editor inserts typed characters into the current document.

The editor starts in Normal mode. To return to that mode from Insert mode, press the **ESC** key.

2.4.2 Basic Editing

As previously noted, VIM is based on the vi editor and anyone familiar with vi's editing keys will immediately be at home with VIM.

VIM uses two methods of entering commands into the application. Simple commands such as those used for navigation and entering Insert mode are one- or two-letter commands that are case sensitive and are not echoed to the screen as they are typed. More complex commands such as those used to perform searches and utilize the tags created by the `ctags` program are entered and echoed into the last line of the application screen. A colon, slash or question mark character is used to activate the last line mode and enter the command. The character used will depend upon which command is being issued.

Most simple commands can be preceded by a number. When this occurs, the command entered is treated as if it had been entered that many times. For example, the command to delete a line in a buffer is `dd`. If this command preceded by the number 15, as in `15dd`, the next 15 lines in the buffer will be deleted.

To exit VIM, type `:q` if the text has not been changed, `:q!` to abandon all changes made to the file, or `:wq!` to save all changes and exit.

For help on any function, type `:help <name>` where `<name>` is the name of the function. Typing

```
:help tutor
```

will bring up information about VIMs built in `tutor` file. This file will walk you through the basics of using VIM.

Navigation

If the terminal that is being used is set up correctly, the arrow keys on the keyboard will often work for simple navigation commands that move the cursor around the screen. If the terminal type is not properly configured, alternate keys may be used to navigate through the text.

The "h", "j", "k", and "l" keys can be used in place of the left, down, up and right arrow keys respectively. Additionally, the cursor may be moved one word, sentence or paragraph at a time by using the keys listed in Table 2-13.

Table 2-13 Navigating VIM

Action	Command
Hjkl	Left, down, up, right
W	Move one word forward
E	Move one word backward
()	Move to previous, next sentence
{ }	Move to previous, next paragraph

It is also possible to scroll the text on the screen without moving the cursor from its current position. Table 2-14 lists some of the more common commands.

Table 2-14 Folding Mode Commands

Action	Command
^F ^B	Scroll one screen forward, backward
^U ^D	Scroll one half screen up, down
^E ^Y	Scroll one line up, down

Insert Mode

There are numerous ways to enter Insert mode depending on where in the buffer you wish to insert text and how much of the text you wish to change. For example, it is possible to insert text at the beginning of the current line, at the end of the line, to change only the letter or word at the cursor, or to change text from the cursor to the end of the line. All of these commands are accomplished by simple one- and two-letter commands. When you press the “i” key, VIM enters Insert mode and starts inserting the next typed characters at the current cursor position. If “a” (append) is used to enter Insert mode, the insertion point is the character following the cursor.

Pressing “I” will insert characters at the beginning of the line, and “A” will append characters to the end of the current line. The “o” character opens the line after the current one for inserting text, and the “O” key creates a new line above the current one and positions the cursor to enter new text.

Table 2-15 lists these commands.

Table 2-15 Entering Insert mode

Action	Command
i a	Begin inserting characters at the cursor, after the cursor
I A	Begin inserting characters at the beginning, end of the line
o O	Open a new line after, before the current line

In addition to simply entering text into the buffer, vim can change text already in the document. Table 2-16 lists the commands for changing text and their function.

Table 2-16 Commands to change text

Action	Command
cw	Change from the current cursor position to the end of the word
cc	Change the current line
r	Replace the letter at the cursor
R	Replace from cursor on

Also of note are the deletion commands. They are entered from Normal mode and may be used to delete a character, a word or a line at a time. These commands are listed in Table 2-17.

Table 2-17 Deletion commands

Action	Command
x X	Delete the character at, before the cursor
dw	Delete from the cursor to the end of the current word
dd	Delete the current line
D	Delete from cursor to end of the current line

Automatic Indentation

There are several indenting options available with VIM. They are `autoindent`, `smartindent` and `cindent`. The first, `autoindent`, simply copies the indentation from the previous line and uses that as the default for all new lines entered into the buffer. `Smartindent` and `cindent` are similar, but `cindent` is stricter in the way that it handles the formatting and may not be suitable for programming languages other than C.

To turn any of these indent modes on, with the application in Normal mode, type `:set <indent mode>`. To turn off an indent mode, simply preface the mode name with the word “no”. For example, to turn on `cindent`, type:

```
:set cindent
```

To turn `cindent` back off, type:

```
:set nocindent
```

2.4.3 Using Tags with VIM

The `ctags` program can be used to build a tags file that is readable by VIM and can be used to quickly navigate among multiple source code files. Building the tags file is done in the same way as described earlier in the chapter for `Jed`, but you must use the `ctags` program instead of `etags`.

```
$ ctags *.ch
```

This builds the `tags` file in the current directory.

To build a tags file for an entire project that spans many subdirectories, from the main project directory, issue the `ctags` command with the `-R` option to recurse the directories.

```
$ ctags -R
```

From within VIM, if you need to jump to a particular function that is in the current file, place the cursor on the appropriate function and press `^]`. This command uses the tags file to locate the function and reads that file into the buffer.

To open a file and locate a function that is not present in the current buffer, type:

```
:tag <tagname>
```

where `<tagname>` is the name of the function that you are looking for. This will load the file into the buffer and position the cursor at that appropriate tag.

To list the tags that you have jumped to in this editing session, type `:tags`. The output from this command is shown below.

```
:tags
# TO tag          FROM line  in file/text
  1  1 main                1  ./arch/alpha/boot/tools/mkbb.c
  2  1 perror             116 ./arch/alpha/boot/tools/mkbb.c
>
```

By pressing the number listed in the left column, you can return to previously accessed tags.

In large projects, it would not be uncommon for one function to be referenced in several places. In the event that multiple tags are created with the same name, you can select from the list of similar tags by using the `tselect` command. To select from a list functions, type `:tselect <tagname>`. When the `tselect` command completes, you will be presented

with a list of matching functions from which to choose. When you type in the appropriate number from the left-hand column, that function will be opened in the buffer.

2.4.4 Folding Code

VIM can use several methods for folding code. The most useful of these for application programming is the indent method. The variable `foldmethod` determines how the folding will take place. To indent set the mode by typing:

```
:set foldmethod=indent
```

This command can also be set as a default by entering it in the `~/.vimrc` configuration file.

When this option is set and a file is opened, VIM will parse the file looking for initial tab sequences in order to determine the various indent levels and fold the file automatically. In order to open or unfold a section, use **zo** and to close or refold a section, use **zc**.

The commands **zm** and **zr** can also be used to increase and decrease the amount of folding currently being done to a file. By issuing the **zr** command, the amount of folding being done is reduced by one *shiftwidth* level. That is to say, one level of tabbing is revealed in the code. The **zm** command reverses this and folds up one level of indentation every time that it is used.

2.5 References and Resources

1. Learning GNU Emacs, Second Edition, Debra Cameron, Bill Rosenblatt & Eric Raymond, O'Reilly & Associates, Inc., ISBN:1-56592-152-6.
2. GNU Emacs home page, <http://www.gnu.org/software/emacs/emacs.html>
3. Jed home page, <http://space.mit.edu/~davis/jed/>
4. GNU Emacs Lisp Reference Manual, http://www.gnu.org/manual/elisp-manual-21-2.8/html_chapter/elisp.html
5. Coffee.el, a fanciful elisp program to have Emacs make coffee, <http://www.chez.com/emarsden/downloads/coffee.el>
6. Xemacs home page. <http://www.xemacs.org/>
7. VIM Home page. <http://www.vim.org>