Emacs: The Software Engineer's "Swiss Army Knife" ©N.S. Matloff, 1992

1 Overview

The **emacs** editor is arguably the "editor of choice" among good software engineers. You will be required to use it in our course (and be responsible for answering questions on it in exams), but you should quickly find that it is so nice that you would want to use it even if it were not required.[1]

What is so nice about **emacs**? Here are some examples:

- **Emacs** has a really nice windowing capability, e.g. allowing you to look at two different parts of a file at the same time.
- **Emacs** has an excellent interface to the **gdb** debugger, which we will be using. This will save you a lot of debugging time.
- **Emacs** has its own programming language, **emacs-lisp**, which you can use to customize **emacs** to your own personal usage patterns.
- **Emacs** has a lot of miscellaneous tools which are very helpful, e.g. a built-in calculator.

How much new learning will be required? The answer turns out to be "Very little," because **emacs** includes its own copy of **vi**! And this is not just a lazy way to avoid learning—in my opinion, it is actually the preferable thing to do. I myself use **vi** within **emacs**. The reason for this is that the basic set of **emacs** commands usually require more keystrokes than their **vi** counterparts. For example, to move the cursor left a distance of one character, one must type control-b in **emacs** (two strokes), but type only h in **vi** (one stroke). In other words, this way you get the best of both worlds—the power of **emacs** but the typing convenience of **vi**.

2 Before You Begin

Copy the .emacs file, called DOTemacs in the directory from which you are reading this, to your home directory. This is not absolutely necessary, but when you start up **emacs**, it will read the .emacs file in your home directory. By copying my version of that file, you will be able to take advantage of **vi** more easily.

3 Notation

Most of the **emacs** commands use either the control key or the escape key, which are denoted in **emacs** circles as C- and M-. For example, C-x b would mean to type control-x and then b; C-x C-b would mean to type control-x and then control-b; M-x would mean to type ESC then x (but release ESC before typing x).

When in vip-mode, the **emacs** mode in which **vi** commands are available, the ESC key can't be used for **emacs** commands, since it has special meaning for **vi**. For this and other reasons, **emacs** provides an alternate, the underscore key, i.e. '_'. So, for example, any M-x can be done as _x

[1]At one school I know of, the students are taught **emacs** in their first programming course, and then for variety, are told to use **vi** in their second course. A TA at that school told me that all the students in that second course secretly use **emacs**!

4 Functions and Key Binding

To illustrate the concepts of **functions** and **key bindings**, consider the action, common in any editor, of moving the cursor one character to the left (e.g. this is done using the h key in **vi**). In **emacs**, the formal name for this function is 'backward-char'. As with the other **emacs** functions, we could invoke this function using M-x, i.e. by typing

```
M-x backward-char
```

Of course, this would be far too much typing for such a small action. **Emacs** allows key-binding to take care of this problem. In this case, for example, we say that **emacs** has "bound C-b to the function backward-char," meaning that whenever we type C-b, that function will be invoked.

We can change key bindings, and add new ones, by using various **emacs** functions, e.g. 'global-set-key'. In my .emacs file, for example, I have the line

```
(global-set-key "\Cx-v" 'vip-mode)
```

which binds C-x v to the function 'vip-mode'.

5 Getting In and Out of Emacs, and Out of Trouble

To start **emacs**, just type

```
emacs
```

To temporarily suspend **emacs**, type C-x C-z.[2]. To revive it, just type fg, or better yet, %emacs (fg would only revive the most recently suspended or "backgrounded" job, which might not be **emacs**).

To exit **emacs** just type C-x C-c.

Emacs is so useful that most people start up an **emacs** session as soon as they log on to the machine, even though they may not need it right away. They just want to have it ready when the need arises.[3]

You should do this too. If you are at a workstation console, then reserve one window for **emacs**.[4] Otherwise, start up **emacs** when you first log on, and then immediately suspend it; then you can instantly revive it whenever you need it.

If you find yourself in a mess while in **emacs**, say by having typed the wrong thing, you can kill the current command by typing C-g.[5] Remember this command!

6 Buffers

In a typical **emacs** session, you will be editing several "files" at once. Some of these will be real files, while others will be temporary files[6] which are created as the result of **emacs** commands.

[2] C-z also works, but this would conflict with **emacs**'s **vi** capability.

[3] And since it generally does take a few seconds to start up, it's better to just do this once, at the beginning of one's Unix session.

[4] Better yet, include a line 'emacs &' in your .xinitrc file, which will automatically create an **emacs** window for you when you start up X11.

[5] If you are in vip-mode at the time, you may have to type C-g more than once.

[6] Since they are temporary, they will not be on the disk, though you have the option of saving them to disk if it becomes useful to do so

For example, suppose I wish to edit the file WC.c. I type C-x C-f to load the file, and then type WC.c. The screen will then look like this:

```
/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
   but of course a real file can be read by using the Unix '<' redirection
   feature */


#define MaxLine 200


char Line[MaxLine];  /* one line from the file */


int NChars = 0,  /* total number of characters in the file */
    NWords = 0,  /* total number of words in the file */
-----Vi:****WC.c****************11:36pm*0.00*Mail***(C)----Top------------------
```

This looks just like regular **vi**, except for the bottom line, in which **emacs** gives some general information, like the time of day. (Again, the bottom line is in reverse video on the screen, but that doesn't show up here.)

Suppose this is my first **emacs** command. Then WC.c will be my first buffer. Now suppose I type

```
M-x calendar
```

in order to use **emacs**'s built-in calendar facility.[7] The screen would now look like this:

[7]Actually I must type '_-x calendar', since I am in vip-mode and thus the ESC key isn't available for **emacs** commands.

```
/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
   but of course a real file can be read by using the Unix '<' redirection
   feature */

-----Vi:****WC.c****************11:42pm*0.05*Mail***(C)----Top------------------
        November 1992            December 1992            January 1993
      S  M Tu  W Th  F  S      S  M Tu  W Th  F  S      S  M Tu  W Th  F  S
      1  2  3  4  5  6  7            1  2  3  4  5                     1  2
      8  9 10 11 12 13 14      6  7  8  9 10 11 12      3  4  5  6  7  8  9
     15 16 17 18 19 20 21     13 14 15 16 17 18 19     10 11 12 13 14 15 16
     22 23 24 25 26 27 28     20 21 22 23 24 25 26     17 18 19 20 21 22 23
     29 30                    27 28 29 30 31           24 25 26 27 28 29 30
                                                       31
C-x <      Calendar    ? help/o other/c current     Sun, Dec 6, 1992       C-x >
Loading calendar.elc...done****************************************************
```

What has happened is that **emacs** has created a new buffer, for the calendar, and displayed it along with my first buffer (for WC.c). The two buffers are delineated on the screen by reverse-video lines at the bottom edge of each buffer (which I have shown as long strings of asterisks here).

The calendar buffer is now a temporary file. I can edit it, e.g. delete November, and if I desire to do so, save it to a permanent file.

Here are some commands dealing with buffers:

```
C-x o    if two buffers appear together on the screen, move the cursor
         to the other buffer
C-x 1    if two buffers appear together on the screen, expand the one
         current containing the cursor to make that buffer fill the screen
C-x b    display another buffer which is not currently displayed; emacs
         is usually able to guess which buffer you want, and will ask you
         to confirm that its guess is right; otherwise, specify the buffer
         name yourself
C-x C-b  list all buffers (note: this list itself will be a new buffer!)
C-x C-f  read in a file and create a new buffer for it
C-x C-r  read in a file and create a new buffer for it, but make the
         buffer read-only (so that the file will not accidentally get
         changed)
C-x k    kill buffer; emacs will ask you to confirm that you mean
         the one currently containing the cursor; otherwise, specify
         which one
```

Again, remember that any buffer is a temporary file, and thus all editing commands apply. For example, suppose I type C-x C-b to get a list of all buffers. Again, this will create a new buffer, and suppose the list is so long that it doesn't fit in one screen. Then I can use the usual editor scrolling commands (for **vi**, C-f and C-b) to browse through the buffer.

6.1 The Minibuffer

When you issue an M-x or C-x command that requires some parameter, **emacs** will display the command in the **minibuffer** at the bottom of the screen, and move the cursor there, waiting for your input of the parameter. In some cases, **emacs** will offer a default value for that parameter. If the default is OK, just hit the Return key; otherwise, use the Delete key or the Backspace key (might be different on different terminals; try it out) to partially or completely erase the default, and type in the parameter.

6.2 Loading Files

As mentioned earlier, the command C-x C-f will prompt you for a file name, create a new buffer, and load the file into that new buffer. Here are some more details, illustrated with an actual example.

I typed C-x C-f. **Emacs** responded by displaying the following message in the minibuffer:

```
Find file: ~/tmp/tmp/
```

I was currently in the directory indicated, i.e. tmp/tmp. I knew that the file I wanted began with the string 'Sum', but didn't remember the rest, so I took advantage of **emacs**'s **name completion** feature: I hit the Space bar, resulting in **emacs** creating and displaying a buffer named *Completions*,[8] with contents

```
Possible completions are:
Sum                                 Sum.c
Sum.lst                             Sum.mas
Sum.out                             Sum.sym
```

which was a list of all files in that directory whose names began with 'Sum'. I saw that the file I wanted was Sum.mas, so I typed '.m' and then hit the Space bar, so that the minibuffer now looked like

```
Find file: ~/tmp/tmp/Sum.mas
```

That is what I wanted, so I hit the Return key, and **emacs** then loaded the file.

If I had wanted to switch to another directory, I could have backspaced in the minibuffer, erasing part or all of the path displayed there.

Note that this is useful not just if you have forgotten a file name, but also to save typing.

7 Windows

As mentioned before, a very nice feature of **emacs** is the ability to split a given buffer into two or more windows, enabling one to view more than one piece of a file at a time. Here are the main commands:

```
C-x 2    split current window into two windows
C-x o    move cursor to the other window
C-x 1    unsplit, i.e. change back to having just one window on
```

[8] You may wish to move the cursor to this buffer, e.g. to scroll it (the list may be too large to fit the window) or maybe even to save it to a file. You cannot make this move by using C-x b, as that would need the minibuffer, which is already in use, but you can temporarily leave the minibuffer by using C-x o, to go to another window; just keep using this command until you get to the *Completions* window. By the way, the *Completions* buffer will be killed once you actually finish with the minibuffer.

```
         this buffer
C-x 0    move cursor to the other window and make that one fill
         the screen (same as C-x o followed by C-x 1)
C-x ^    make current window larger (and other window smaller)
```

For example, suppose I currently just have one window on my WC.c buffer in the example above. If I type C-x 2, the result will be:

```




/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
-----Vi:****WC.c****************12:14am*0.06*Mail***(C)----Top------------------




/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
   but of course a real file can be read by using the Unix `<' redirection
-----Vi:****WC.c****************12:14am*0.06*Mail***(C)----Top------------------
```

Now I can scroll the two windows independently. For example, suppose I want one window to show the function WordCount(), and the other window to show the call to that function.[9] What I can do is use the **vi** search command '/' to look for the string 'WordCount', and thus position the two windows as desired:

```
int WordCount()

{  int I,NBlanks = 0;

   for (I = 0; I < LineLength; I++)
      if (Line[I] == ' ') NBlanks++;

   /* ordinarily the number of words is NBlanks+1, except in the case
      in which the line is empty, i.e. consists only of the end-of-line
      character */
-----Vi:****WC.c****************12:17am*0.00*Mail***(C)----47%------------------
      Line[LineLength++] = C;
      if (C == 10) break;
   }

   NChars += LineLength;
   NWords += WordCount();
```

[9]This is a very common usage, e.g. to check whether the parameters match up correctly, though in this example there are no parameters.

```
   NLines++;
   return 0;
}


-----Vi:****WC.c***************12:17am*0.00*Mail***(C)----88%-----------------
```

8 Using Vi in Emacs

The two main commands are:

```
M-x vip-mode   change from emacs mode to vi mode
C-z            change from vi mode back to emacs mode
```

Both for convenience and for easy remembering, I have in my .emacs file (which you should copy, as mentioned earlier) these commands:

```
C-x v   change from emacs mode to vi mode
C-x e   change from vi mode back to emacs mode
```

Note that if there are two windows on a buffer, the **vi** commands work on the two windows as a whole; it is just one buffer, i.e. just one file. So, for instance, when I was searching for the string 'WordCount' in the example above, I first typed

```
/WordCount
```

using the standard **vi** search command '/', and then moved to the other window and simply typed 'n'. This is also a standard (though possibly new to you) **vi** command, meaning, "Repeat the last search."[10] So it did look for a string 'WordCount', even though I had changed windows.

Just as ordinary **vi** will automatically read a file .exrc from your home directory, and read in customization commands from it, one can do this here too. In my .emacs file in my home directory, I have put the two lines

```
(local-set-key "]" 'vip-scroll)
(local-set-key "[" 'vip-scroll-back)
```

after the line

```
(vip-mode)
```

The latter line has **emacs** start up in vip-mode, and then the "local" commands apply to that mode. Whenever I am using vip-mode, the] and [keys will have the same meaning as **vi**'s usual scroll-forward and scroll-backward commands, i.e. C-f and C-b.

[10] A summary of the major **vi** commands is given in Appendix B. Again, some of these are probably new to you. Learn them, because they can save you a lot of keystrokes, time and mental concentration.

9 Using Emacs for Compiling and Debugging Programs

Emacs can save you a large amount of time and frustration in your debugging. Use it!

The two main commands are:

```
M-x compile   compile a program
M-x gdb       debug a program
```

(of course, in vip-mode these are _-x compile and _-gdb).

Since in a debugging session one will compile many times, to save typing, I have in my .emacs file bound C-x c to M-x compile, so I need type only C-x c.

Here is what happens when I type the compile command:

```
/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
   but of course a real file can be read by using the Unix ‘<’ redirection
   feature */


#define MaxLine 200


char Line[MaxLine];  /* one line from the file */


int NChars = 0,  /* total number of characters in the file */
    NWords = 0,  /* total number of words in the file */
-----Vi:****WC.c****************11:46am*0.02*Mail***(C)----Top------------------
Compile command: make -k
```

Note the minibuffer here (‘Compile command: make -k’). Though you can't see it here, the cursor is right after the -k. **Emacs** is waiting for me to confirm that this is the compile command which I want. I want a different one, so I use the Delete key (or Backspace key, on some terminals) to erase the ‘make -k’, and then put in the compile command which I want, which in this instance is ‘cc -g WC.c’: The screen now looks like this:

```
/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
   but of course a real file can be read by using the Unix '<' redirection
   feature */


#define MaxLine 200


char Line[MaxLine];  /* one line from the file */


int NChars = 0,  /* total number of characters in the file */
    NWords = 0,  /* total number of words in the file */
-----Vi:****WC.c****************11:51am*0.09*Mail***(C)----Top------------------
Compile command: cc -g WC.c
```

I then hit the return key, and **emacs** creates a new buffer for the compile:

```
/* introductory C program */

/* implements (a subset of) the Unix wc command  --  reports character,
   word and line counts; in this version, the "file" is read from the
   standard input, since we have not covered C file manipulation yet,
-----Vi:****WC.c****************11:52am*0.32*Mail***(C)----Top------------------
cd /usr/home/matloff/Courses/40/Progs/
cc -g WC.c








-----Emacs:**compilation********11:52am*0.32*Mail***(Compilation:*run)----All---
(No files need saving)
```

Eventually, that new buffer will include a 'Compilation finished' message, and will display any syntax errors which it found. If there are any such errors, I can keep typing

```
C-x `
```

(C-x and then a backward apostrophe), and **emacs** will automatically go to the buffer for my source file, and position the cursor at the source of the error; this is a big convenience.

By the way, note the message 'No files need saving'. If I had made a modification to the source file, **emacs** would have asked me if I wanted to save the file before compiling it.

I then invoke **gdb**. **Emacs** will ask me what the executable file name is, in this case a.out. **Emacs** will create a new buffer for **gdb**. Then I give **gdb** a breakpoint command, to stop at the function WordCount(), and I type r to run. When the program reaches the breakpoint, my screen looks like

```
Breakpoint 1 at 0x40023c: file WC.c, line 43.
(gdb) r < z
Starting program: /usr/home/matloff/Courses/40/Progs/a.out < z

Breakpoint 1, WordCount () at WC.c:43
(gdb)




--**-Emacs:**gdb-a.out**********12:05pm*0.36*Mail***(Inferior*Gdb:*run)----Bot--

int WordCount()

{  int I,NBlanks = 0;

=> for (I = 0; I < LineLength; I++)
      if (Line[I] == ' ') NBlanks++;

   /* ordinarily the number of words is NBlanks+1, except in the case
      in which the line is empty, i.e. consists only of the end-of-line
      character */
-----Vi:****WC.c****************12:05pm*0.36*Mail***(C)----47%------------------
```

There are now two buffers displayed on the screen, my **gdb** buffer at the top and my WC.c buffer at the bottom. Note the marker '=>' in the latter; it shows that my next source-line statement is

```
for (I = 0; I < LineLength; I++)
   if (Line[I] == ' ') NBlanks++;
```

If for example I give **gdb** an n command now, it will execute that statement (and the '=>' marker will then advance to the next line).

Note again that if I want to restore the WC.c buffer to be displayed in full-screen instead of half-screen form, I merely need to type C-x o to switch to the WC.c buffer, and then type C-x 1.

10 Using Emacs to View Unix Man Pages

This is done with the M-x man command. A buffer is created for the man page, and then you can use the regular **emacs/vi** cursor-movement commands to move around the man page entry. This is much more convenient than simply using the ordinary **man** command in the shell. Also, the current directory is continuously displayed in the minibuffer, which is helpful.

11 Shell Use

The **emacs** command M-x shell will create a new buffer, and a regular shell in it. You can run any shell command there except those which use **curses**, e.g. **talk**.[11] One of the nice things about this is that this is a buffer; thus you can edit it, save it to a permanent file, etc.

You may wish to have more than one shell buffer, say each of them half-screen size, so that you can see both at once. To get a second shell, just run M-x shell again, but first you will have to change the name of the buffer of the first shell, from '*shell*' to some other name of your choice. Do this using the M-x rename-buffer command.

12 Miscellaneous Emacs Features

12.1 Name Completion

This was described earlier, for file names. The same holds for command names, buffer names, etc.

You can give just the first few letters of, say, an M-x command (e.g. M-x cal instead of M-x calendar). Then hit the Space bar, and **emacs** will do name completion, showing you which commands begin with the string you have so far.

Use of the Tab and Return keys in this context produces results like Space, but with some differences. Experiment with them, and you'll see the difference.

Once you get accustomed to this feature, you will find that you rarely have to spell out a full command name; the Space bar will do most of your typing for you! All you have to do is give enough characters for uniqueness, and hit the space bar.

12.2 Abbreviations

Suppose you are writing an article about the state of Washington. Instead of typing "Washington" again and again, you need type it only once. After you type it that first time, place the cursor one character to the right of the final letter in the word, in this case the second 'n', and type C-x +. You will then be prompted for an abbreviation, say 'wa'. From then on, whenever you type 'wa', followed by a space, **emacs** will expand your abbreviation to the full word 'Washington'.

To take effect, the **emacs** variable 'abbrev-mode' must have the value 't', i.e. true. You can check this by using the C-x v command, and if it has the value 'nil', i.e. false, you can either use the 'set-variable' function to set it, or simply 'M-x abbrev-mode'.

12.3 File Backup

Emacs periodically will make backup copies of all files you are editing. The copies are distinguished by a tilde at the end of their names.

12.4 Using Emacs Under X11

Emacs does allow you to make use of the mouse in certain ways. One of the most useful is as a speedier

[11]Note that this will mean, for example, that you will need to type carriage returns when using the Unix **more** command, whereas they are not needed ordinarily. The command M-x terminal can be used to get a curses-capable shell, but on the other hand it doesn't work very well as an bfsemacs buffer.

alternative to C-x o in switching windows. If you are in one window and wish to move to another, simply click the left mouse button in that window. And even better—the cursor will not only move to that window, but also move to the exact position within the window that the mouse pointed to when you clicked.

12.5 Customizing Emacs

Emacs allows you to make your own abbreviations of commands. Some you would use only in a given **emacs** session, or even only in a given **emacs** buffer. Others you would use all the time, and thus you would put them into your .emacs file. See the on-line help facilities described below for the commands global-set-key, local-set-key, add-global-abbrev, etc. My .emacs file shows some examples of their use.

You can also use the emacs-lisp language to write new commands.

13 Learning More About Emacs

13.1 On-Line Help

Emacs has an excellent on-line help facility, including the following commands:

```
C-h t     invoke an emacs tutorial
C-h f     get a short description of a function
C-h a     list all functions whose name includes the given string
C-h b     list all the bindings of keys
C-x v     get an explanation of a given emacs variable, and find out
          its current value
M-x info  get more extensive information on emacs features
```

The tutorial is mainly about cursor-movement commands, which is not very useful if you will (as I recommend) be using vip-mode for those actions.

C-h f is much more useful. If for example I wish to know about the various e-mail commands which **emacs** has, I can type C-h a and then when prompted in the minibuffer, type 'mail'. A buffer will then be created which lists all **emacs** commands whose name includes the string 'mail'. For example, one of them will be rmail; I can then learn more about it by typing C-h f and then answering 'rmail' when the prompt appears in the minibuffer.

The command M-x info is quite good. Just follow instructions (e.g. your first command in it will probably be 'm emacs').

13.2 Library Files

Various **emacs** commands are actually files in the lisp subdirectory of the emacs directory on your machine. You may wish to browse through this subdirectory. To find out where it is, type

```
whereis emacs
```

to the Unix shell. The .el files you see in that directory are written in the emacs-lisp language, so you can use these as examples to help learn from. Note that whenever you see a function or variable name which is new to you, you can find its definition by using C-h f.

13.3 Books

Several books on **emacs** are available. Some are better than others. I like *Learning GNU Emacs*, by Cameron and Rosenblatt, published by O'Reilly and Associates (this publisher offers a wide selection of Unix books).

14 Using Emacs on Non-Unix Systems

Emacs is available for VMS and DOS systems. For the latter, it is called DEMACS. Both are available from various **ftp** archive sites.

A My .emacs File

```
(display-time)
(calendar)

(global-set-key "\C-xc" 'compile)

(global-set-key "\C-xv" 'vip-mode)
(global-set-key "\C-xe" 'vip-change-mode-to-emacs)
(vip-mode)
(local-set-key "]" 'vip-scroll)
(local-set-key "[" 'vip-scroll-back)
(local-set-key "z" 'vip-toggle-case)
(defun break-long-line ()
   (interactive)
   (save-restriction
      (widen)
      (beginning-of-line)
      (forward-char 79)
      (search-backward " ")
      (delete-char 1)
      (newline)
      (previous-line 1)
))
(local-set-key "t" 'break-long-line)
```

B Summary of the Major Vi Commands

`h`	`move cursor one character to left`
`j`	`move cursor one line down`
`k`	`move cursor one line up`
`l`	`move cursor one character to right`
`w`	`move cursor one word to right`
`b`	`move cursor one word to left`
`0`	`move cursor to beginning of line`
`$`	`move cursor to end of line`
`nG`	`move cursor to line n`
`RET`	`move cursor to next line`
`C-f`	`scroll forward one screen`
`C-b`	`scroll backward one screen`

C-d	scroll down half screen
C-u	scroll up half screen
:.=	determine current line number
i	insert at current cursor position (end with ESC)
a	insert after current cursor position (end with ESC)
dw	delete current word (end with ESC)
cw	change current word (end with ESC)
r	change current character
˜	change case (upper- vs. lower-) of current character
dd	delete current line
x	delete current character
D	delete this character and everything to its right
ma	mark currrent character
d‘a	delete everything from the marked character to here
p	dump out at current place your last deletion
u	undo the last command
.	repeat the last command
J	combine next line with this one
%	find the matching parenthesis/brace/bracket
:w	write file to disk, stay in vi
:q!	quit VI, do not write file to disk,
ZZ	write file to disk, quit vi
/string	search forward for string
?string	search backward for string
n	repeat the last search
:s/s1/s2	replace (the first) s1 in this line by s2
:lr/s/s1/s2/g	replace all instances of s1 in the line range lr by s2 (lr is of form ‘a,b’, where a and b are either explicit line numbers, or . (current line) or $ (last line)
:set ai	set autoindent mode
:set noai	unset autoindent mode

Many of the commands can be prefixed by a number. For example, 3dd means to delete (consecutive) three lines, starting with the current one. As an another example, 4cw will delete the next four words.

The p command can be used for “cut-and-paste” and copy operations. For example, to move three lines from place A to place B:

1. Move the cursor to A.
2. Type ‘3dd’.
3. Move the cursor to B.
4. Type ‘p’.

The same steps can be used to copy text, except that p must be used twice, the first time being immediately after Step 2 (to put back the text just deleted).