Fe User Guide

Michael Haardt <michael@moria.de>

ABSTRACT

Yet another editor? Yes, and like all other editors, its special design goals may or may not make it your personal favourite. Fe is a folding editor, often also called outline editor. I believe that folding is as important as the progress from line editors to screen editors. Fe is a small editor and needs little resources, but still offers many advanced commands. It does not contain an extension language, but instead is supposed to work out of the box. The user interface is similar to Emacs, in particular to the Emacs-like interface of Origami.

1. Copyright and usage conditions

The source is available from *http://www.moria.de/~michael/fe/*.

2. First things first

The first thing you will notice in fe is the status line, which tells you a lot: If it starts with a star, the buffer contains unsaved changes. Then comes the version number of the editor. After, enclosed in parentheses, you see the current fold language and any modes you should be aware of. Next is the file name associated with the buffer you are editing. If switched on, it is followed by the cursor position. As the last, you see the current local time, which can also be switched off.

At the bottom of the screen is the prompt line. If you type a command that consists of multiple keys, you will see any typed but not yet processed keys. The notation for keys is:

C-*chr*
: Hold the **CONTROL** key while pressing the character *chr*. Thus, **C-f** would be: hold the **CONTROL** key and press **f**. Some people may be used to see **^F** or **CTRL F**, here it is **C-f**.

M-*chr*
: If your keyboard has a META key, that prefixes the character with an **ESC**, then hold it down while typing *chr*. Alternatively, press the **ESCAPE** key and release it, then press the character *chr*.

M-C-*chr*
: Hold down **META** and **CONTROL** while typing *chr*. If you have no **META** key, then press the **ESCAPE** key and release it, then hold the **CONTROL** key while pressing the character *chr*.

Several commands print a prompt and put you in a line editor in the prompt line. This command line editor

has a history function, used by the cursor **up** and **down** keys. There are history lists for the following things:

- File names
- Line numbers
- Strings to search for or replace

The command line editor further does file name completion with the **tab** key (**C-i**).

Although it is commonly said that you edit a file, what you really doing is loading the file into a buffer in memory, edit this buffer and later write it back into a file. As such, if you do not save the buffer, all your work, no matter how much, is lost. It is advised to save regularly, because the system might crash because of a software or hardware failure.

If you are a new user, then read below for the most important commands:

M-x This brings you into the main menu. Using the menu is slower than directly typing the commands you want to use, but if you are a new user, it may help you until you know all commands. You can leave any menu with **C-g**.

M-z Save the buffer and exit the editor.

C-x C-c
Exit the editor. If there are unsaved changes, you will be asked if you want to exit anyway, which means to lose the changes.

M-C-l
Redraw the screen. This is useful after you received output from *write*(1) or *talk*(1), or if you got line noise.

3. Files and Buffers

C-x C-b
Name buffer.

C-x C-i
Insert file. The mark will be set at the beginning of the inserted file and the cursor will be at its end.

C-x C-r
Load buffer from file. The mark will be set to the end of the file and the cursor to its beginning.

C-x C-s
Save buffer to file.

C-x C-w
Save buffer to a different file. Fe will ask for a file name and if you want to omit all fold marks. Sometimes people insist on getting files that do not contain all those confusing braces. ;-)

4. Arguments

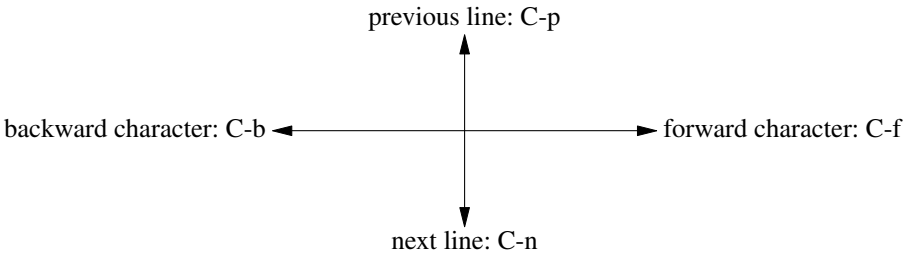
Many commands take an numerical argument. In general, a positive argument repeats the command as often as the argument says, whereas a negative argument repeats the opposite commands as often. Nevertheless, read the description of each command to get to know what an argument does for it, if anything at all.

M-- This starts setting a negative argument.

M-*digit*
If used as the first command to set an argument, this sets a positive argument. Once you start setting an argument, you can type regular digits or **M-***digit* to continue typing the argument.

5. Moving the cursor

Moving the cursor is an important thing in an editor. The most basic commands are:



If you use an argument, any of these will be repeated *argument* times. A negative argument will repeat them just as much, but in the opposite direction. If your terminal supports cursor keys, those will work as well.

Knowing these commands gets the job done, but it will not be efficient on bigger files. For that reason, you should also look to use the more complicated cursor movement commands, they make editing more pleasant and save much time:

C-a Go to the beginning of the line. If the cursor is already at the first non-white space character of the line, it will go to the very first character of the line, else to the first non-white space character. Press **C-a** a few times to see how it works.

C-e Go to the end of the line. If the cursor is already at the last non-white space character of the line, it will go to the very last character of the line, else to the last non-white space character.

C-v, **next page**
Go to the next page or until the end of the current fold. If an argument is given, this causes repetition of this command. If the argument is negative, the cursor will move to the previous page or the beginning of the current fold.

M-v, **previous page**
Go to the previous page or until the beginning of the current fold. If an argument is given, this causes repetition of this command. If the argument is negative, the cursor will move to the next page or until the end of the current fold.

M-< Go to the beginning of the buffer.

M-> Go to the end of the buffer.

M-C-n
Go to the last line inside the fold.

M-C-p
Go to the first line inside the fold.

M-f Move one word forward.

M-b Move one word backward.

M-g Go to line. The number of the line if given by the argument, if you specify one, else you will be prompted for it.

C-l Center display (put the current line in the middle of the display). This is not a cursor motion, but it saves moving the cursor to get just the same effect.

M-# Go to matching fence, that is a matching paren, brace or bracket.

6. Editing

C-d Delete the character under the cursor. You can not delete fold marks or remove a newline which could cause two fold marks to be in one line. An argument causes deletion of *argument* characters under the cursor, a negative argument causes deletion of *−argument* characters left to the cursor.

C-m, **Return**
Insert a newline character and indent the new line as much as the previous line. If auto-indent mode is switched off, **C-m** and **C-j** have reversed meanings.

C-j, **Linefeed**
Insert a newline character. If auto-indent mode is switched off, **C-m** and **C-j** have reversed meanings.

M-C-d
Delete the word right under the cursor. You can not delete fold marks or remove a newline, that could cause two fold marks to be in one line. An argument causes deletion of *argument* words under the cursor, a negative argument causes deletion of *−argument* words left to the cursor.

M-C-h
Delete the word left to the cursor. You can not delete fold marks or remove a newline, that could cause two fold marks to be in one line. An argument causes deletion of *argument* words left to the cursor, a negative argument causes deletion of *−argument* words right to the cursor.

C-h, **backspace**
Delete the character left to the cursor. You can not delete fold marks or remove a newline, that could cause two fold marks to be in one line. An argument causes deletion of *argument* characters left to the cursor, a negative argument causes deletion of *−argument* characters under the cursor.

C-q Quote character. Sometimes you may want to insert a character, that is also part of an editor command. The solution is to quote the character using this command. Example: **C-q C-@** inserts a null character. Further, this function composes ISO-8859-1 characters from ASCII characters using the same pair used by many terminals and X11 servers, e.g. **ss** gives a sharp s and **a"** gives a small a with diaeresis. A positive argument inserts as many characters as specified.

C-t Transpose characters.

C-x =
Describe line.

M-c Capitalise word.

M-l Lower case word.

M-u Upper case word.

C-x #
Insert the value of the counter and increment it.

7. Regions and the kill ring

A region is the amount of text between an invisible mark you can set and the cursor. For many operations, like removing regions, they must respect the fold structure, i.e. you can not remove a region, that only contains the begin of a fold. Similar to the cursor, typing at the mark position pushes the mark forward, so when edit a previously empty buffer, the mark will always be at its end until you set it elsewhere.

Besides the buffer, that holds your file contents, there is a ring of buffers for text you have *killed* (which means removed, erased, wiped out). You can yank any element of this ring back to your text. This is universal, as it allows you to duplicate killed text, move it, exchange regions of text and much more. Kill commands that directly follow each other will append the killed text to the current ring element, but if you do anything else in between, a new element will be created.

The following commands manipulate regions and the kill ring:

C-@, Set the invisible mark, which marks one end of a region at the current cursor position. On some terminals, you have to type **C-@**, on others **CONTROL-space** is fine. Some even accept both. Some ancient ones accept neither, which is why there is a second, less handy, keyboard command.

C-x C-x
Swap cursor and mark.

C-k This command kills all text from the cursor position to the end of the line. If the concerned part contains the beginning of a fold, the whole fold will be killed. If the line is empty, the terminating new line character will be killed. This is a short-cut for setting a mark, moving the cursor appropiately and killing the region.

C-w Kill region.

M-w Copy region.

C-y Yank kill ring element back.

M-y Replace the just yanked back text with the previous element of the kill ring. That way you can browse through the kill ring until you found what you are looking for. A negative argument moves forward.

C-x C-f
Filter region. The marked region will be read as standard input by the command which name you will be prompted for. The standard output of this command replaces the marked region, if the command terminated successfully and if it could be read correctly. Further, the old and the new contents are put in the kill ring, so if filtering did not generate the intended result, you can walk back in the kill ring with **M-y** to get the old region back. A common application of this function is to reformat a paragraph by filtering a region through **fmt**.

C-x | Pipe region. The marked region will be read as standard input by the command which name you will be prompted for. The standard output of this command will be placed in a new buffer. This command is useful to run **spell** or **diction**, or just any command that analyses text.

8. Folding

M-C-@
Fold region.

C-u Unfold.

C-o Open fold.

C-c Close fold. A positive argument closes the given number of folds, a negative argument opens them.

9. Search and replace

C-s Incremental search. This differs from the usual searching by searching as you type what to search for, so you can stop typing the rest of the search string if you already found what you were looking for by typing its beginning. Incremental search has the following commands:

C-h, **backspace**
Delete the last character from the search string.

C-s Move to the next occurence of the search string in the buffer and switch search direction to forward.

C-r Move to the previous occurence of the search string in the buffer and switch search direction to backwards.

C-p, **cursor up**
Switch to the previous search string in the history.

C-n, **cursor down**
Switch to the next search string in the history.

C-q Quote/compose character.

C-g, **cancel**
Abort searching, move the cursor back to where it was before starting incremental search. The search string will not be stored in the history.

Enter
Quit searching and leave the cursor where it is. The search string will be stored in the history.

any other character Append the character to the search string.

C-r Incremental search, but starting with search direction backwards.

M-C-r
Search backwards, go to the last line, that contains the string you will be prompted for.

M-C-s
Search forward, go to the next line, that contains the string you will be prompted for.

M-R Search and replace. You will be promted for what to search and what to replace it by. If no argument is given, all occurences of the search string from the cursor position until the end of the text will be replaced. An argument limits the number of substitutions and a negative argument causes searching backwards.

M-r Query search and replace. You will be promted for what to search and what to replace it by. If no argument is given, you will be asked for each occurences of the search string from the cursor position until the end of the text, if you want to replace it. An argument limits the number of substitutions and a negative argument causes searching backwards. The replace menu offers the following choices:

Yes Replace the string and proceed.

No Do not replace the string and proceed.

Rest, !
Replace the rest without asking any more.

Abort
Do not replace the string and abort.

Replace and abort, .
Replace the string and abort.

10. Modes and variables

Modes are switches, that change the general behaviour of the editor for a certain buffer:

C-x m *mode*
Adds (enables) the specified mode.

C-x C-m *mode*
Deletes (disables) the specified mode.

The following modes exist:

b Show begin and end marks of a file in a fashion similar to Origami.

i Auto-indent mode. While being in this mode, **return** causes the new new line to be indented as much as the previous line, while **C-j** just creates a new line. If auto-indent mode is switched off, the two keys have the reversed function.

l Display the line number of each line.

m Magic mode. Searching (regular and incremental) will search for regular expressions instead of strings.

o Overwrite characters instead of inserting them. You can not overwrite newline characters or fold marks and no auto-indenting will be done in overwrite mode.

p Show the cursor position in the status line.

r Read-only. This mode is automatically set, if you load a file on which you have no write permission.

R Always redraw the screen immediatly instead of only redrawing it when there are no characters typed ahead.

s Scroll the whole screen sideways instead of only scrolling the current line.

t Show the current local time in the status line.

B Display fold marks in a bold font.

Additionally, the **insert** key toggles insert/overwrite mode.

In contrary to modes, which can only be switched on or off, variables can be set to one of many values.

C-x : *variable*
Set the specified *variable*.

The following variables exist:

d Display/window size. The size of a window can be changed by setting this variable.

l Set the folding language, which determines which comments will be used to encapsulate fold marks.

t Tab width. Setting the tab width to 0 causes tabs to be displayed as ^I.

Counter/increment. The counter is global for all displays, whereas the other variables are per-display. It is handy when enumerating something and usually used in recorded macros. Its default value is 0 and the default increment is 1.

11. Macros

Sometimes during editing, you will find that you press the same key sequences over and over. In that case you should record the sequence of keys once and then replay it when you need it. Such a recorded sequence of keys is called a macro. The following commands deal with macros:

C-x (Start recording a macro.

C-x) Stop recording a macro.

C-x e Execute the previously recorded macro. A positive argument repeats the number of executions.

12. Windows

Windows make editing considerably easier. You can split your screen and either edit the same file in two windows (which is why windows are called displays at times) or edit different files. The following commands split and merge windows:

C-x 2
Split the screen in two windows.

C-x 0
Delete the current window.

C-x o
Switch to the next window.

13. Shell commands

The following commands are the interface to the shell:

C-x c Start an interactive sub-shell.

C-x ! Start a non interactive shell command.

C-z Suspend fe.

14. A few words on the design of fe

After working with Origami for many years, both as user and developer, I have decided to implement a new folding editor: fe. There are a few basic design decisions which make it different from Origami:

Origami is partially written in its extension language. While being more of a hack in the beginning, it has changed to a full programming language. As such, it takes time to be learnt. Code written in it is not well usable by most people who don't know it. The conclusion for fe is not to support any extension language, neither a homebrew one or a standardised language, like scheme. Instead, fe is implemented as a library of basic editor primitives. Its is easy to write your own editor by using that library. fe can be modified easy without having to learn a new programming language. The editor also stays small and elegant this way, while Origami has to offer zillions hooks for extension functions.

Origami implements folds as double-linked n-trees, with nodes being single lines or folds. This allows quick manipulation of folds or lines. But region oriented functions become hard to implement and are mostly not too quick any more. fe uses a gap buffer to store text, folds are represented as meta-characters in the flat buffer. This means basic fold primitives are slower than in Origami, but more complex and region oriented functions are faster. During development of the first prototype of fe, I found many functions in Origami not to be as canonical as they should be after I implemented them in fe. From my past experience, the performance of typical personal computers to have increased by a factor of 10 every 5 years in the last decade, so the circumstances for editor design have clearly changed over time.

Folding has its merits, because it adds a structure to flat files. But, it also means that by bad structure design and badly commented folds the file will get harder to understand, not easier. This can happen up to the point where you want all folds to be gone to see what the file contains. If you need to keep many folds open during development to see their contents, you are probably prepraring that situation at least for others. Although in general I don't like programs forcing me to do something, fe makes an exception here for two reasons: If the structure is obvious, you *want* the editor to close folds for you automatically. If it is not, you *want* to recognise that before it is too late. For that reason, fe closes all folds when you leave them. If you want to transport code from one fold to another, just split the current display and edit the file at two places. If both places are sufficient far away from each other, that's what you even had to do if you could leave folds open.