# Emacs and UNIX tricks for programmers

Adrian Perrig
aperrig@di.epfl.ch
Student in CS at EPFL

Version 3.0

June 28, 1996

**Abstract**

I introduce basic tools and methods for programming in the UNIX environment. Knowing about the various programs can increase the productivity substantially. We further introduce the idea of amortized productivity. WARNING, this is a beta version of the final document. If you find errors, inconsistencies or ways to improve this report, please write suggestions to aperrig@di.epfl.ch. Thanks. The latest version is available from my homepage [Per96]

## 1 Introduction

Every user has a certain repertoire of commands he uses. Even with just a small set of commands, we can accomplish a wide variety of tasks, but not really in an efficient way. Having used emacs now for several years, I noticed that I could gain orders of magnitude speedup by using the right command at the right place. In the beginning, watching friends working, pointed me to some useful commands I did not know. Later by looking at online documentation I discovered more and more jewels. This observation lead me to form the expression *Amortized Productivity*, with the idea in mind that when you invest time for improving a method, you can gain that time back later during the usage of the method. In some cases, you can even get speedups of entire orders of magnitude.

Having identified some of the most useful non-trivial commands in Emacs and Unix, I wrote this report to give ideas to other people that work in that environment.

While writing this document, I made the following assumptions:

- Keyboard commands are faster than mouse commands. In the beginning, using the mouse during editing under emacs seems natural and easy. There is just one problem with this - it is slower than plain keyboard commands. That's where *Amortized Productivity* comes in - it takes longer to learn the keyboard shortcuts, but later, you are always a little bit faster.
- If a functionality seems to be useful, it is very probable that somebody already implemented it. Unix and Emacs both exist now for several years. Numerous creative programmers worked for years to improve the interface and add new useful capabilities. So when you would like to have a certain function in emacs, look for it and you will almost certainly find it.

- We try to make the common case fast. The proposed methods are not always faster. For instance, keyboard is not always faster than mouse commands. But if a method is faster on average, you will win by using it.

### Conventions

`Mono-spaced font` is used for commands and keys to type. A keystroke such as `C-e` means Control-e, `MC-e` stands for Meta-Control-e. For emacs commands containing the Meta prefix, you can either type `Escape` before the next key, or hold the `Alt` key while typing the rest of the command (On some keyboards, it may be another key for Meta).

# 2 Emacs

Emacs was written initially by Richard Stallman at MIT. One of the objectives was to provide a free, expandable and powerful editor that would actively support the most common editing tasks. The result is overwhelming. Emacs is the most useful tool for any programmer with the premise of doing high quality work productively. In this section, we will only see a glimpse of the myriads of possibilities emacs provides. But it will be enough to get you on the right track to learn more.

In this section, it is impossible to fully describe emacs. I tried to identify the tasks, where a rich repertoire of commands can substantially improve productivity. Further, knowing how to find more information is also important for finding new useful commands. Identifying what would be useful and then looking for the feature are the first steps of *Amortized Productivity*.

## 2.1 Moving around

There are many possibilities to move the cursor position around. During programming, we need to move in a very special fashion and emacs provides support in many different ways. Some of the commands are also available through the cursor keys, such as up, left, page up, etc. Keeping in Mind that each time the hand leaves its keyboard position to search the cursor keys, the hand also needs to come back to its natural position, which is much slower than using the equivalent control sequences. It is also more natural to work in a way that keeps the hands on the same place.

Table 1 summarizes the most basic movement commands. These commands are not only useful in emacs, they are also valid in other programs such as the c-shell or in text entering dialogs in Netscape.

A very quick way to move to a different position provide the search functions, especially the incremental search functions. If you want to be more fancy and you know for example, that the line you would like to jump to contains a `printf` statement and the variable `Data_p`, you can try the incremental regular expression searches, which are also explained in table 2. You can also set markers in your emacs buffers and jump around with the corresponding commands which are described in table 3.

## 2.2 Entering frequent text

During programming, we usually need to enter often similar words. This can take a long time in the case of long identifiers. For example, who likes to type `Ada.Strings.Unbounded.UnboundedStrings` more than once? Check out Table 4 if you like typing jumbo names just once.

The abbrev commands provide further support for frequently typed data. Please refer to the manual for the details, or try `M-x help a abbrev` or `M-x command-apropos abbrev`, which lists all the possible commands of the abbrev mode.

Table 1:

| Basic Movements | |
|---|---|
| Key | Action |
| C-p | Up, previous line |
| C-n | Down, next line |
| C-b | Left, backward char |
| C-f | Right, forward char |
| M-b | Backward word |
| M-f | Forward word |
| C-a | Beginning of the line |
| C-e | End of line |
| C-v | Page down |
| M-v | Page up |

Table 2:

| Searches | | |
|---|---|---|
| Key | Action | Explanation |
| C-s or C-r | Incremental search forward or backward | The direction of the search can be changed continuously during the search by typing C-r or C-s. Further, if we want to skip to the next occurrence of the match, we can type C-s again (or C-r respectively). Typing C-s (or C-r) twice, repeats the last search. |
| MC-s or MC-r | Regular expression incremental search forward or backward | The usage of these commands is similar to the normal search commands. The difference is that the search arguments are specified by regular expressions. |

Table 3:

| **Jumps** | | |
|---|---|---|
| Key | Action | Explanation |
| C-x / (key) | Set mark | (key) designates any key on the keyboard, including control sequences. The mark will be saved in an emacs register identified by the key entered. Hint: The registers saving text or points are the same. Any register can either hold text or a position. |
| C-x j (key) | Jump to mark | This command jumps to the place pointed to by register named (key). If the position is in a different buffer, the buffer is changed automatically. This function is very practical for programming. For instance, the position in the code that I currently work on is the c register, the position of the local variables in the l register, the global variables in the g register, etc. This allows very fast switching between points in large files. |
| M-. | Jump to Tag | See section 2.4 for more details on this and other etags commands. |
| C-x o | Jump to other window | If the screen is split in multiple windows, we can switch between the different windows with C-x o |
| C-x b | Switch to buffer | This exchanges the current buffer with a different loaded buffer |
| C-x C-x | Exchange point with mark | If we frequently move forth and back between two points in the same buffer, we can use this command to jump between two locations. It is very useful if we just briefly visit another portion of the code and we want to return back. This is accomplished by setting the mark at the current position (C-Space), visit the other location and return with C-x C-x. |
| C-u C-Space | Cycle mark ring | All the marks are saved in a mark ring. Repetitively calling this command lets you visit all the last positions you set a mark. |
| MC-n (MC-p) | Jump forward (backward) to the matching parenthesis | This jumps forward or backward to the next or previous matching parenthesis of the same level of indentation. |
| MC-u (MC-d) | Move up (down) in the current list structure | Jumps up or down to the next higher or deeper level of parenthesis. These list commands are extremely useful if you have deep nested block levels in your source code. |

Table 4:

| Frequent text | | |
|---|---|---|
| Key | Action | Explanation |
| M-/ | Find closest word, dynamic abbreviation expand | This function is very useful if we have long variable names in programs. It looks backwards and expands to the closest word that starts with the same letters as we already typed. If no word backward in the text was found, the text forward of the point is searched. |
| C-x x (key) | Save the region in the register (key) | If we need to frequently insert the same pieces of text, we can store the text in a register. |
| C-x g (key) | Insert register into buffer | The text previously saved with the above command, is inserted in the text. |
| M-TAB | Expand tag | Tries to complete the word started to a function name or constant name. Ambiguities are presented in a separate buffer. |

## 2.3 Cutting and Pasting Text

Another task often used is to cut a part of your text and insert it in other places. Emacs has a kill ring, where it inserts all the deleted text. Except the character deletion commands, namely `C-d` and `backspace`, all kill commands insert the deleted text into the kill ring. Consecutive kills append the deleted text at the same location of the kill ring. Pasting text back is made with `C-y`, the so-called yank commands. If you would like to paste older killed text from the kill ring, you can repeatedly type `M-y` (after the initial `C-y`). Table 5 summarizes the different methods for marking, deleting and pasting.

## 2.4 Etags

Etags is a stand-alone Unix program which creates an index database of functions and other identifiers out of the specified C/C++ source files. Emacs reads this file and uses the information for the etags commands. These functions are very useful for programmers. One of the features they allow is to jump to the beginning of the code of a specified function.

To generate the etags database file, you need to be in the directory containing your c-files. For example, typing `etags *.[ch]` in your shell will create the file `TAGS`, containing the references to all C-functions and definitions.

Table 6 describes the basic emacs commands.

## 2.5 Writing Keyboard Macros

Keyboard macros are really easy to write and they can save a great amount of time. Using the registers for file positions or text, the search and jump functions described above, you can write powerful macros that, for example, collect information in different files and write them to another file. Useful macros can be saved in the .emacs file and assigned to a key shortcut[1]. Table 7 summarizes the most useful commands.

[1]The C-c ( + key ) combinations are good candidates for shortcuts, as they are not often used for other commands. If you would like to check if a key combination is already used, type `M-x describe-key-briefly` followed by your key combination

Table 5:

| **Cut Functions** | | |
|---|---|---|
| Key | Action | Explanation |
| C-k | Delete line | Delete everything up to the end of the line. If you pass a numerical argument (C-u (digits)), it deletes that many lines. It deletes backwards by specifying a negative argument. |
| M-z (key) | Zap to char | Delete everything up to and including the character (key). For example, when you type `M-z .` everything up to the end of the phrase is deleted. |
| M-d (M-backspace) | Kill word forward (backward) | Deletes the entire word forward or backward and is inserted in the kill ring. |
| C-w | Kill region (Delete) | Kill the entire region from the mark to the current position. |
| M-w | Insert region into kill ring (Copy) | The same as C-w, but leaves the text in the buffer. |
| M-x mark-c-function | Mark current c function | If we want to kill or insert an entire function into the kill ring, we can mark it first with this function and then delete the entire funciton with C-w or insert it with M-w. |
| MC-Space | mark subexpression | Marks the entire subexpression following the point. (Note: a subexpression is a logical entity in a program. For example an identifier, a block inside parenthesis, for examples are subexpressions.) If you would like to mark a parenthesized expression, move the cursor to the opening parenthesis (using for example `MC-p` described in section 2.1) and call this command. |
| MC-k | kill subexpression | Deletes the subexpression following the cursor. The command behaves similar to the one just described before. |

Table 6:

| **Etags** | | |
|---|---|---|
| Key | Action | Explanation |
| M-. | find tag | Jumps to the location where the function or constant is defined. Emacs opens the file containing the function automatically, if it is not yet open. |
| M-TAB | Expand tag | Any function name that is indexed by a tag file can be expanded to its full name. If the expansion is ambiguous, the variants are displayed in a separate buffer. This feature can also be very useful, if we are not sure about the function name. |

Table 7:

| Macro | |
|---|---|
| Key, Action | Explanation |
| C-x ( | Start recording a keyboard macro |
| C-x ) | End the recording |
| C-x e | Calls the last created macro |
| M-x name-last-kbd-macro | Assigns a name to last created keyboard macro. This is necessary, if you would like to insert the macro into a buffer (or into the .emacs file). |
| M-x insert-kbd-macro | Inserts the keyboard macro into the current buffer. Usually, when a useful macro was created and inserted into the .emacs, a keyboard redefinition such as `(global-set-key "\C-cn" 'prototype_to_function)` is very useful. |

## 2.6 .emacs File

The .emacs file is located in your home directory and it is loaded when emacs is started. It contains initializations and user defaults. It is common to define own key shortcuts, colors and fonts. A sample `.emacs` file can be loaded from [Per96].

At this place, I would like to give an example of how to use macros in conjunction with the `.emacs` file. A common task during programming is to write function headers. It is very practical to write only the function definition in the .h file, copy it to the .c file and apply a pre-written macro with a key shortcut.

We have the following function definition:

```
   void
deb_printf( int level, const char *fmt, ...);
```

The definition for the `prototype_to_function` macro in the .emacs file looks as follows:

```
(fset 'prototype_to_function
   "^S(^B^@^A\367^P^P
/*^U77-^M * ^Y
*
* input:
*
* output: returns SUCCESS/FAILURE
*
* sideeffects:
*
*^U77-
*/^S;^H^N^A{
return SUCCESS;}^H /* ^Y *./^?^?/^M")

(global-set-key "$\backslash$C-cn" 'prototype_to_function)
```

This definition looks very cryptic, but we actually never really need to edit or write the macro, as emacs takes care of this, when you use the commands in section 2.5. Typing C-c n with the cursor on the `deb_printf` name gives us the desired transformation:

```
/*----------------------------------------------------------------------------
 * deb_printf
 *
 * input:
 *
 * output: returns SUCCESS/FAILURE
 *
 * sideeffects:
 *
 *----------------------------------------------------------------------------
 */
    void
deb_printf( int level, const char *fmt, ...)
{
    return SUCCESS;
} /* deb_printf */
```

Such macros are very practical, they can save us a great deal of typing and speed up the work.

Setting colors for a given screen and taste can be difficult. Emacs provides the commands `M-x list-faces-display` to display the current font settings and `M-x list-color-display` to show the available colors. Syntax highlighting can be turned on with the command `M-x font-lock-mode` or automatically with a command such as `(add-hook 'c-mode-hook '(lambda () (font-lock-mode t)))` in the `.emacs` file.

If you use emacs for programming, line numbers are often used. I have the following lines in my .emacs file:

```
(line-number-mode t)
(global-set-key "\C-cg" 'goto-line)
```

The first line displays always the current line number in the mode line. If you need to jump to a specific line, you can just use `C-c g`.

## 2.7 Getting more information

This brief chapter on emacs tricks is far from being complete. Emacs provides many more capabilities. To inform and help the user, emacs provides two different help systems, info and "normal" help. The normal help system is called by typing C-h[2] and the info system with `M-x info` or `C-h i`.

The help system provides powerful functions for quickly finding the desired information. Table 8 explains some of the capabilities.

The info pages provide detailed help on a various UNIX programs or even on programming libraries. Emacs is just one of the chapters of the info system, which is invoked by typing `M-x info`. Table 9 shows some of the valid commands in info mode.

Especially for GNU programs, the provided information is excellent. It contains examples, background information, cross references to follow, etc. Info is also a separate program, you can call it from the shell, for example `info gcc` will open directly the gcc section.

[2]On some keyboards, C-h is identical with the backspace key. Usually it is practical to redefine the backspace (and also C-h) to delete the previous character with the command `(global-set-key "\C-h" 'backward-delete-char)` . To start the help system in this case, we need to type M-x help.

Table 8:

| **Help** | | |
|---|---|---|
| Key | Action | Explanation |
| C-h a | command apropos | Shows all the commands that contain a string entered in form of a regular expression. Typing just Return shows a list of the ~1000 commands. If you would like to know about the different search functions, try `C-h a search` and you will find out about interesting commands, such as M-x tags-search. |
| C-h b | describe bindings | All active key bindings are shown within the corresponding mode. This feature is invaluable for example in Bib TeX mode, with its dozens of special bindings. Further, commands you knew they existed but did not dare to ask for, are listed. |
| C-h c | describe key briefly | Describes the function that is bound to a certain key shortcut. |
| C-h f | describe function | Gives a description of the function. |
| C-h C-f | info about function | Jumps to the info page describing the function specified. This facility also gives you background information about the command. Further, similar functions are also described on the same info page, which is also very useful. |
| C-h m | describe mode | Gives a thorough description about the current mode used. |
| C-h w | Where is the function | This help function shows you how a given function can be called. It will display the shortcut key or where to find the function in the menu. |

Table 9: Info table

| **Info** | | |
|---|---|---|
| Key | Action | Explanation |
| ? | help about info | Displays the valid info commands briefly. |
| d | jump to root node | Takes you back to the first info node. |
| m | visit topic | Pick menu item specified by name or abbreviation. |
| u | up one level | Go back to next higher page (usually to the page where “m” was typed the last time). |
| l | last page | Go back to the last page visited. |
| s | search regexp | Search the regexp on the current page. |
| 1, 2, ... | visit nth entry | Opens the nth menu entry of the current page. |

### 2.8 Code writing support

Emacs provides special modes for many different programming languages, such as `perl-mode, c-mode, c++-mode, ada-mode, makefile-mode, lisp-mode,` etc. Each mode has specific knowledge on how to structure your code, about the keywords for syntax highlighting, about subexpressions and list structures for the `MC-p` family commands, and others. Everybody has his own style for structuring the code, so you can define your preferences in the `.emacs` file. Please check the info system for more information on this topic, or load my `.emacs` file from my homepage. The variables that affect the code indentation, are described in the mode help, try `C-h m` when you are in your favorite programming mode.

See table 10 for special programming commands.

Table 10:

| **Programming** | |
|---|---|
| Key, Action | Explanation |
| MC-\, M-x indent-region | Restructures the marked region. This command only restructures the spacing before the beginning of the line, the rest of the line is unchanged. It is not as powerful as the indent command, which is described in section 4.2. |
| M-x compile | Executes the command specified in the variable `compile-command`, which can be set in .emacs with `(setq compile-command "make -k")`. |
| C-x ‘ | Parses the `*make*` buffer for errors and jumps to the point in the file where the error occurred. To use this feature, it is necessary to compile with `M-x compile`. |

## 3 Finding Information

As usual in UNIX, there are many different ways and methods to find the desired information. Useful commands are `man`, `apropos`, `whatis`, and `info`. Info was treated in the emacs section 3.

A great new source of information is the WWW. Using the search-engines such as AltaVista, HotBot, Lycos or Infoseek, it is surprising how much material can be found on a topic in very little time, provided that you know the syntax of the search-engine and that you enter precise criteria.

### 3.1 man, apropos, whatis

The information provided by man is divided into different sections. Table 11 shows the different sections.

Usually, section 3 is subdivided into different subsections, such as 3c for c libraries, 3x for x libraries, 3tk or 3tcl for tcl/tk, etc.

Sometimes, there is a program and a library function with the same name. To specify, which section to read, you can try one of the following commands (examples for the time c-function which is in section 3):

- `man -S 3 time`
- `man 3 time`

Table 11: Man Sections

| | |
|---|---|
| 1 | Programs |
| 2 | System Calls |
| 3 | Library functions |
| 4 | Special Files |
| 5 | File Formats |
| 6 | Games |
| 7 | Miscellany section |
| 8 | Administration and privileged commands |
| 9 | Kernel reference |

- `man -S 3:2:1 time`

The information about a certain man section, for example the library section, can be obtained with `man -3 intro`.

If you would like to know in which sections a word appears, try either of the two equivalent commands:

- `man -f time`
- `whatis time`

`Apropos` is another command which searches for keywords in a short descriptions of commands. It is equivalent to `man -k` and can be useful in case you do not exactly know what you are looking for.

# 4 Various UNIX tools

In this section, we will look at various tools which may help during program development.

## 4.1 strace

System calls can be traced with strace. If you always wanted to know what your program 'buggy' does "behind your back", try `strace buggy | more` and you will be surprised (if you started it for the first time).

## 4.2 indent

It takes hours to restructure big files of unreadable c-code. Much faster and less of a pain is to read the man pages about the indent program, write a .indent.pro file and run indent to do the task.

## 4.3 limit

Unix can impose restrictions on a process, such as the memory size, the CPU time used, the maximum size of a core dump file, etc. You can view the preset limits with `limit`. If the maximum allowable core dump file size is 0, and you would like to have the core file, try `limit coredumpsize 10000` to set it to 10 MBytes. You can also create core files, which are analyzable by gdb, by typing C-\.

## 5 Acknowledgments

Most of the knowledge was assembled from group projects at Carnegie Mellon University. Mainly John Drum influenced me and showed me most of the tricks and hacks. I would also like to thank Daniel Schneider very much for the numerous discussions and his corrections to the report.

## References

[Per96] Adrian Perrig. WWW Homepage. http://diwww.epfl.ch/~aperrig, 1996.