Editing with Emacs

Written by Dave Pawlowski, September 23, 2012

Introduction

Emacs is a free, portable, and extensible text editor that you will find on pretty much every Linux based system in existence. It works on a variety of architectures and other operating systems, and as it is quite portable, no matter what system you are on, you will find that the behavior of emacs is always the same. Emacs is extremely popular with programmers. If you use a common language, Emacs probably provides a mode that makes it especially easy to edit code in that language, which provides context sensitive indentation and layout, colors, and even the ability to run programming sessions, compile programs, debug your code, and interact directly with the language interpreter inside the Emacs editor itself.

Previously I mentioned that emacs is a WYSIWYG editor. This is actually not the case, in that you can do so much more with it. However, we will be using it mostly in this capacity.

A fantastic description of emacs resides at the GNU Emacs website.

Notation

Emacs was designed to run on systems that did not have access to devices other than the screen and keyboard, i.e. no mice. Therefore, the only way to save, open, close, delete, move the cursor, etc. was by using the keyboard. For this reason, there is an extensive list of keyboard commands that will allow you to do everything that you need to do when editing text using the keyboard.

Every keystroke in Emacs is a command. Typing the letter “A” is a command to insert the letter “A”. There other commands that do not result in a character being printed to the screen. The standard notation that describes these keystrokes follows:

C-x
For any *x*, the character Control-*x*

M-x
For any *x*, the character Meta-*x*. The Meta character is usually the escape key as few keyboards have the Meta character.

C-M-x

For any x, the character Control-Meta-x.

RET
The return key.

SPC
The space bar.

ESC
The escape key.

Many, but not all, commands in emacs can be performed using some combination of the keystrokes above. However, every command has a long name, like *kill-line* or *delete-backward-char*. Most typical commands are bound to keystrokes for convenience. This is called key binding. One of the great things about emacs is that it is possible to create custom key bindings, or change the default ones to suit your preferences.

Starting emacs

To get acquainted with emacs, lets start by opening the editor by typing:

```
% emacs temp.txt &
```

at the Linux command prompt. The file temp.txt doesn't have to exist, it will create one for you. Its fine if you leave off the file name as well. (If emacs doesn't seem to open when you do this, you should try leaving off the &. You may not have X11 forwarding enabled, or you may not be using the correct version of emacs.)

A note on that & symbol. When you enter a command in the prompt, Linux wants to execute the command in the foreground. What this means for you is that you will not be able to interact with the shell why Linux does its thing. That way, if the system is printing information to the screen, you can read it or whatever. For certain commands though, this is not ideal. You know that nothing will be spit out to the screen, and the command may launch a process that might run for a while, so you want to have control of the shell. You can regain control of the shell by putting the process in the background.

This is especially useful if the process that you are running is using one of those graphical X-windows things, such as emacs. If you start emacs without putting it in the background, you wont be able to interact with the shell.

A process can be put in the background in 2 different ways. The first is by adding the & symbol after the command that will launch the process, as above. This will automatically start the process in the background and give back to you control of the shell.

What happens if you start a process, but forgot to include the &? You don't have to kill the process and start over. Instead, simultaneously pressing the Control key and the letter z, C-z, will stop the foreground job and give you control of the shell. If you do just this, you will not be able to interact with the process that you just started. By immediately typing bg, the process will be put in the background and you will be able to interact with the shell and the process.

Try starting another emacs session in your terminal, only this time leave off the &.

```
% emacs
```

Notice that you can only interact with the X-window, and not the terminal itself. Now, click on the shell, and type:

```
% C-z
% bg
```

After the C-z the terminal should say "Suspended", meaning that the emacs process is stopped and you can no longer interact with it. When you type bg, you put the process in the background and are now able to interact with both emacs and the terminal.

Emacs commands you absolutely need to know

So you should have a file open. Type some stuff. Once you do that, you'll actually want to save your work.

```
> C-x C-s
```

will accomplish this. Note that is two separate key strokes. You type control-x then let go and then type control-s in succession.

If you look at the bottom of the emacs window, emacs will tell you that you just saved a file. This area of the window is called the minibuffer. When you open a file in emacs, you open what is referred to as a buffer. You can have many buffers open at once, which is nice as they are easy to switch through.

The minibuffer tells you information about a command that you may have entered, provides a space to enter longer, more complicated commands, and gives you options when working with certain commands.

Now that you saved a file, lets open another one. Type

```
> C-x C-f
```

In the minibuffer, emacs wants you to specify a file. By default it sets you up to create a new file in, or open a new file from, your current working directory.

In the minibuffer, type

```
> temp2.txt
```

Now the active buffer should be a file called temp2.txt. You have opened 2 files in emacs and it is easy to switch between them.

```
> C-b
```

This tells emacs that you want to switch to a different buffer. Note this is not the same as opening a different file. The file already has to be loaded in the emacs' memory. The default is the last buffer that you were working with. Since temp.txt is the only other file that we've been working with, that's the only other option:

```
> Ret
```

Now temp.txt should be the active buffer. Open up a few more temporary files, temp3.txt, and temp4.txt. Now lets switch back to temp2.txt:

```
> C-b
```

Note that in the minibuffer, the default is not temp2.txt, since that was not the last one we worked with. You can just type temp2.txt in the minibuffer to access that one. Even better though, type:

```
> temp2
```

in the minibuffer (leave off the .txt), and hit the tab key. Tab completion works in emacs just like in the shell. If there are multiple options based on the letters that you entered, then you will get a list of the possibilities.

Undo

Obviously, there will be a time when you need to undo a keystroke. This is accomplished using the

```
> C-x u
```

key binding.

Exit the minibuffer

If you do something that gives you access to the minibuffer, like type

```
> C-x C-f
```

to open a file, and then decide that you don't want to actually do that, you can quit the minibuffer using the

```
> C-g
```

key binding.

Exiting Emacs

Finally, actually exiting the emacs process is accomplished by

```
> C-x C-c
```

Emacs is smart, so it will make sure you saved all the buffers that you were working on, and if you haven't it will ask you if you would like to.

Other useful commands

When programming especially, it is useful to be able to delete text quickly without having to hit backspace for every character.

```
> M-d
```

will delete the word directly after the cursor, and

```
> M-delete
```

will delete the word preceding the cursor.

Also,

```
> C-k
```

will delete everything on a single line after the cursor. When doing any of these deletions, emacs automatically copies the word/words that were deleted into memory, and you can then paste them elsewhere using

```
> C-y
```

This is especially useful, especially for programmers when you often have multiple lines that are identical, or very similar. Instead of re-typing the entire line, you simply go to the beginning of the line, type

```
> C-k
> C-y
> C-y
```

and you will now have a copy. See, much quicker.

Lastly, if you just want to move the cursor, but don't want to have to hit the arrow keys over and over again, there are a few commands to do that too:

```
> M-f
```

will move the cursor one word forward.

```
> M-b
```

will move it one word backwards.

```
> C-v
```

will move the cursor down one page

```
> M-v
```

will move it up one page.

Summary

For now, those commands will get you using emacs, and doing so more efficiently than a lot of people. Utilizing the word/line cutting key bindings and pasting along with being able to move the cursor through text quickly can really speed up the text editing process. And to be honest, it is remarkable how much time is saved by not having to rely on the mouse to do a lot of things.

One last note. There are literally thousands of command options in emacs. These are just the basics of the basics. Many of the commands need to be bound to keys manually. This is done using a file that is read when ever emacs is started up, called .emacs. This file must be created by you, and must reside in your home directory (where all startup files typically go). The syntax for adding or changing commands is a bit cryptic, which is why its usually just easier to google whatever it is you want to do and find someone that has done it before. A good place to start for some useful customization is my .emacs file, which you are welcome to copy and use however you want.