sp00ky's Help document

sp00kyb00g13

August 17, 2017

Contents

1 GNU/Linux

1.1 Documentation

Function Name	Description
man hier	Information on filesystem

1.2 Command Line

Function Name	Description	Example
detex	strips tex commands from file	detex <filename>
diction	Prints misused words	detex <filename> | diction -bs
type	prints info about a command	type cd

2 Emacs

General tips on emacs, probably will contain a lot of forum posts.

2.1 Commands or How to use them

2.1.1 Align-regexp

This post gives a good explanation of using `align-regexp`

2.2 AUCTeX

`C-c RET` - TeX-insert-macro to insert macros (/href etc) `C-c C-e` - LaTeX-environment insert stuff

2.3 Emacs Startup Time

Read the comments in this file to see how to evaluate startup time.

2.3.1 Daemon

Emacs can be started as a daemon to boost its startup time. Lots of info can be found here.

```
1 emacs --daemon
2 emacsclient -nw
```

To kill the daemon simply use `M-x kill-emacs`.
To run multiple emacs client observe the following.

```
1 emacs --daemon=<my-server-name>
2 emacsclient -nw -s <my-server-name>
```

2.4 Color-Themes

Function Name	Description	Example
what-cursor-position	With a prefix arg (`C-u`) it will describe the face at the cursor	`C-u M-x what-cursor-position`

3 ELisp Help

Most of the information here is taken from The Emacs manual

3.1 Useful Functions

3.1.1 Misc

Functions that are usefull but don't deserve their own subsubsection.

Function Name	Description	Example
let*	like `let` but emacs set each variable in sequence, variables at the end can make use of variables at the start	n/a

3.1.2 Minibuffer

Function Name	Description	Example
read-from-minibuffer	Prompts for info in the minibuffer	`read-from-minibuffer 'Enter Your Name'`

3.1.3 Buffers

Function Name	Description	Example
buffer-name	Gets current buffer name	n/a
other-buffer	Gets previous buffer name	n/a
switch-to-buffer	Changes emacs focused buffer (the display buffer)	`(switch-to-buffer (other-buffer))`
set-buffer	Changes the attention of the program to a different buffer (doesnt change the displayed buffer)	`(set-buffer (other-buffer))`
beggining-of-buffer	goto beginning of buffer and leave mark on previous position	n/a

3.1.4 Lists

Function Name	Description	Example
cdr	Manipulate list	n/a
mapcar	Apply function to sequence, i.e apply function to every element in sequence	`(mapcar '1+ [1 2 3])`

3.1.5 Markers

Mark is a position in the buffer.

Function Name	Description	Example
mark-whole-buffer	n/a	n/a
save-excursion	performs body then returns to point	`isave-excursion (let *...)`

3.1.6 Points

Point is where the cursor is. Part of the buffer between point and a mark is called `region`

Function Name	Description	Example
save-excursion	Saves the location of point, executes the body of the function and restores point (also restores the buffer).	`(save-excursion body...)` or `(let varlist (save-excursion body...))`

3.2 Writing Elisp

3.2.1 Functions

Here are 5 important points when writing a function:

1. The name of the symbol to which the function definition should be attached.
2. A list of the arguments that will be passed to the function. If no arguments will be passed to the function, this is an empty list, ().
3. Documentation describing the function. (Technically optional, but strongly recommended.)
4. Optionally, an expression to make the function interactive so you can use it by typing M-x and then the name of the function; or by typing an appropriate key or keychord.
5. The code that instructs the computer what to do: the body of the function definition.

Listing 1: Function skeleton

```
1   function-name (arguments...)
2   ``optional-documentation...''
3   (interactive argument-passing-info)     ; optional
4   body...)
```

3.2.2 Example

The following is an example of a function and its usage, to use it evaluate the function using `C-x C-e` then evaluate the following expression:

Listing 2: Multiply by seven example

```
1   (defun multiply-by-seven (num) ; interactive
2   "Multiply numbers  by 7"
3   (interactive "p")
4   (message "The result is \%d" (* 7 num)))
5
6   (multiply-by-seven 3)
```

The following is an example of an interactive function, to use it do `C-u <number> M-x multiply-by-seven` after you have evaluated the function using `C-x C-e`

Listing 3: Interactive multiply by seven example

```
1   (defun multiply-by-seven (number)       ; Interactive
       version.
2   ``Multiply NUMBER by seven.''
3   (interactive ``p'')
4           (message ``The result is %d'' (* 7 number)))
```

3.2.3 Scope of Variables

Using `let` let sets the scope of the variable to that let scope (inside `let`'s brackets). Here is an example of how to use let:

Listing 4: Let general usage

```
1   (let ((variable value)
2   (variable value)
3   ...)
4           body...)))
```

3.2.4 If

Everything except nil is true. If statement works as expected, heres an example:

Listing 5: If statement example with int

```
1   (if (> 5 4)                                        ; if-part
2             (message ``5 is greater than 4!'')) ; then-part
```

Here is an example using a string:

Listing 6: If statement example with string

```
1   (defun type-of-animal (characteristic)
2   ``Print message in echo area depending on CHARACTERISTIC.
3   If the CHARACTERISTIC is the string \"fierce\",
4   then warn of a tiger.''
5   (if (equal characteristic ``fierce'')
6               (message ``It is a tiger!'')))
```

Here is an example of an if-else:

Listing 7: If-else statement with int

```
1   (if (> 4 5)                                          ; if-part
2   (message ``4 falsely greater than 5!'')          ; then-part
3          (message ``4 is not greater than 5!'')) ; else-part
```

3.2.5 Nil

Nil has two meanings, false and empty list, referred to as `nil` or `()`. These mean the same thing.