

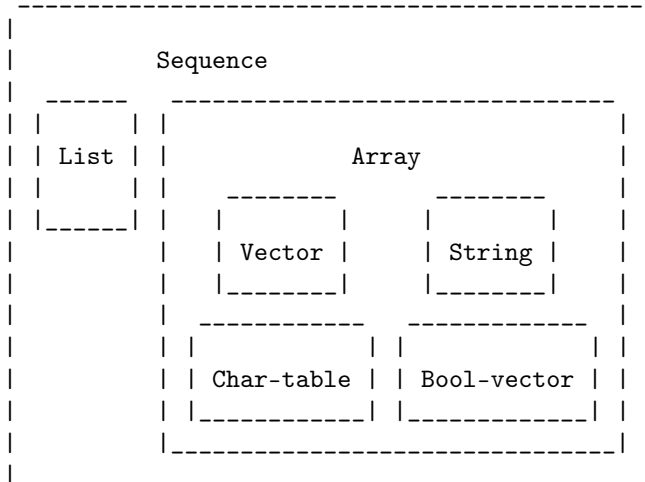
1 Emacs CheatSheet

LANGUAGES

- PDF Link: [cheatsheet-emacs-A4.pdf](#), Category: languages
- Blog URL: <https://cheatsheet.dennyzhang.com/cheatsheet-emacs-A4>
- Related posts: Vim CheatSheet, [#denny-cheatsheets](#)

File me Issues or star this repo.

See more CheatSheets from Denny: [#denny-cheatsheets](#)



1.1 Features

1.1.1 View In Emacs

Name	Comment
Move forward across one balanced expression	<code>forward-sexp</code> C-M-f
Move backward across one balanced expression	<code>backward-sexp</code> C-M-b

1.1.2 Org-mode export Latex

Name	Comment
Make page size bigger	<code>#+LATEX_HEADER: \usepackage[margin=0.5in]{geometry}</code> Link: stackexchange
Change document class	<code>#+LaTeX_CLASS_OPTIONS: [a4paper]</code> Link: stackexchange
Add the date of today	<code>#+LATEX_HEADER: \rhead{Updated: \today}</code> Link: stackexchange
Add page number with total pages	<code>#+LATEX_HEADER: \rfoot{\thepage\ of \pageref{LastPage}}</code>

1.1.3 Buffer Operations

Name	Comment
Move to top	<code>(goto-char (point-min))</code>
Replace string by regexp	<code>buffer-replace.el</code>
Delete region	<code>(delete-region start-pos end-pos)</code>
Buffer string with plain text	<code>(buffer-substring-no-properties start-pos end-pos)</code>
	<code>(get-buffer-create BUFFER-OR-NAME)</code>
	<code>(current-buffer)</code>
	<code>(set-buffer BUFFER-OR-NAME)</code>
	<code>(kill-buffer)</code>
	<code>(set-buffer-modified-p nil)</code>

1.1.4 GNUS - Mail In Emacs

Name	Comment
Create delayed email	<code>gnus-delay-article C-c C-j</code>
Save mail's attachment	<code>gnus-summary-save-parts</code>
Forward mail	<code>gnus-summary-mail-forward</code>
Send gnus drafts	<code>gnus-draft-send-message</code>
Send all the sendable drafts	<code>gnus-draft-send-all-messages</code>
Add attachment	<code>mml-attach-file(C-c C-m f)</code>
Create group	<code>gnus-group-make-group (G m)</code>

1.1.5 Table in Org-mode

```
#+NAME: supplies
| Date          | Category          | Amount |
|-----+-----+-----|
| 2014/01/14 | Supplies          | 43.97 |
| 2014/02/15 | Supplies          | 56.48 |
| 2014/02/11 | Book              | 17.99 |
| 2014/06/10 | Kinesis Keyboard | 289.16 |
| 2014/08/23 | Printer           | 99.96 |
| 2014/08/30 | Supplies          | 58.26 |
| 2014/08/22 | Books             | 18.99 |
| 2014/08/25 | Books             | 7.50 |
| 2014/09/15 | Books             | 21.49 |
| 2014/12/31 | Toner Service     | :=24.95*4 |
|-----+-----+-----|
|              | Total:            |        |
#+TBLFM: @>$3=vsum(@2..@-1);%.2f
```

1.2 Data Structures

1.2.1 Debug

Name	Comment
Debug a function	<code>edebug-defun</code>
Change function via advice	<code>defadvice ;; Super inspiring feature!</code>
Set default value	<code>(setq-default indent-tabs-mode nil)</code>

1.2.2 String

Name	Comment
string1 contains string2	<code>(string-match ".*README.org" buffer-file-truename)</code>
Replace by regexp	<code>(setq ret (replace-regexp-in-string "<hr/>" "" ret))</code>
Format string	<code>(format "%s/%s" mywordpress-server-url blog-uri)</code>
String replace	<code>(replace-string from-string to-string &optional start end)</code>
Replace by regexp	<code>(replace-regexp REGEXP TO-STRING &optional DELIMITED START END)</code>
replace-match	<code>(while (search-forward-regexp "myRegexPattern" nil t) (replace-match " " t))</code>
The second captured string	<code>(match-string 2)</code>
Get the position of the 2nd captured string	<code>(match-beginning 2) (match-end 2)</code>
List matched count	<code>(setq myStr (replace-regexp-in-string "myRegex1" "myRep1" myStr)) (count myStr "myRep1")</code>
Grab the start and end positions of a word	<code>(setq myBoundaries (bounds-of-thing-at-point 'word))</code> <code>(setq myStr (buffer-substring myStartPos myEndPos))</code> <code>(setq myStr (buffer-substring-no-properties myStartPos myEndPos))</code>

1.2.3 Regexp

Name	Comment
Regexp In Emacs	regexp-string-match.el
Change a given string using regex	(replace-regexp-in-string "^ +" "" url)
Search regexp in some string	(string-match myRegex myStr)
Get captured match	(match-string 1 myStr)
Escape special characters	(regexp-quote "^") (regexp-opt '("hello" "world"))=

1.2.4 Integer

Name	Comment
String to int	(string-to-number STRING &optional BASE)
Check whether it's int	(integerp 23)
decimal to hex	(format "%x" 10)
hex to decimal	(format "%d" #xa)

1.2.5 Array & List

- Get items from list

Name	Comment
Get the first element	(car mylist)
Get the nth element	(nth n mylist)
Get the last element	(car (last mylist))
Get the 2nd to the last elements	(cdr mylist)
Get the nth to the last elements	(nthcdr n mylist)
Similar to (car (car value))	(caar value)
Similar to (cdr (car value))	(cdar value)
Return the cdr of the cdr of X.	(cddr X)

- More about list

Name	Comment
Create a list	(defvar my-list (list "item1, item2"))
Add item to list	(add-to-list 'my-list "item3")
Head of a list	(car '(a b c))
Tail of a list	(cdr '(a b c))
Loop a list	(dolist (item my-list) (message item))
Concat two lists	(nconc '("a" "b" "c") '("d" "e" "f")) link
Return a newly created list	(list x)
Append x to the head of a list	(cons x mylist)
Append without duplication	(add-to-list 'auto-mode-alist '("\\.gp\$" . gnuplot-mode))
Add ELEMENT if missing	(add-to-list LIST-VAR ELEMENT &optional APPEND COMPARE-FN)

1.2.6 Position

Name	Comment
Return character at position	(char-after (point))
Return character preceding position	(char-before (point)) (setq myStr (thing-at-point 'word)) (setq myStr (thing-at-point 'symbol)) (setq myStr (thing-at-point 'line))

- Insert text

Name	Comment
Insert string	(insert "hello world") (insert-buffer-substring buffer &optional start end) (insert-buffer-substring-no-properties buffer &optional start end) (insert-file-contents myPath) (insert-file-contents-literally filename &optional visit beg end replace)

- Delete text

Name	Comment
	(delete-char 9)
	(delete-region myStartPos myEndPos)
	(erase-buffer)
	(upcase obj)
	(upcase-word n)
	(upcase-region beg end)
	(upcase-initials obj)
	(upcase-initials-region beg end)
	(capitalize obj)
	(capitalize-word n)
	(capitalize-region beg end)
	(downcase)
	(downcase-word n)
	(downcase-region beg end)

1.2.7 DateTime

Name	Comment
Convert time to string	(format-time-string "<%Y-%m-%d %H:%M UTC +8>" (current-time))
Get current time	(current-time)
Add some offset for a time	(time-add time (seconds-to-time seconds))
Subtract two time values	(time-subtract after-init-time before-init-time)
Get second count	(float-time (time-subtract after-init-time before-init-time))
Return date as a list (mm/dd/yyyy)	calendar-current-date
	(calendar-extract-month date)
m1 will be changed	(calendar-increment-month m1 y1 -1)
	(calendar-date-compare '((12 27 2012)) '((12 26 2012)))
	(calendar-holiday-list)

1.2.8 Hook

Name	Comment
Add hook	(add-hook 'myhook '(lambda () (insert "fun1 was called ")))
Run each hook in myhook.	(run-hooks 'myhook)

1.2.9 Files

Name	Comment
Open file	(find-file html-file)
Save file	(write-file html-file nil)
Get short filename	(file-name-nondirectory somefilename)
Get the directory name from filename	(file-name-directory FILENAME)
Check file/directories existence	(file-exists-p bfilename)
Insert contents of file FILENAME after point	(insert-file-contents somefilename)
Return FILENAME's final "extension"	(file-name-extension "test.erl")
Return FILENAME sans final "extension"	(file-name-sans-extension "test.erl")
Return a list of names of files in DIRECTORY	(directory-files DIRECTORY &optional FULL MATCH NOSORT)
Insert contents of file FILENAME after point	(insert-file-contents FILENAME &optional VISIT BEG END REPLACE)
Confirm directory exists	(file-directory-p FILENAME)
Create directory	(make-directory "~/emacs.d/autosaves/" t)
Find files by name	(find-dired "../" "-name defined.hrl")
read file content into a string	(setq dddstring (with-temp-buffer (insert-file-contents "dd.txt"))=

1.3 Common Scripts

- emacs multiline regexp

```
;; http://stackoverflow.com/questions/1309050/emacs-query-replace-regexp-multiline
(setq content-str "hello
this
----
Denny
Sophia")
```

```
(message (replace-regexp-in-string
  "\n----\\(\\.\\|\\|\\n\\|\\)*" "" content-str))
```

- cond: like switch or case

Link: [WhenToUseIf](#)

```
(let ((x 1))
  (cond ((eq x 0) "It's zero")
        ((eq x 1) "It's one")
        :else "It's something else")
)
```

1.4 More Resources

License: Code is licensed under MIT License.