LEARNING emacs

#emacs

Table of Contents

About

Chapter 1: Getting started with emacs

Remarks

Emacs is a text editor whose most prominent feature is the ability of users to programmatically customize nearly all aspects of it. This is facilitated though a special dialect of the Lisp programming language, called Emacs Lisp, created specifically for use in the Emacs editor.

There are a multitude of extensions written in Emacs Lisp that add to Emacs functionality. These extensions include editing facilities for specific programming languages (similar to what an IDE might provide), e-mail and IRC clients, Git frontends, games such as Tetris and 2048, and much more.

Many aspects of the Emacs editor can be used with no programming knowledge. Users looking to programmatically customize Emacs, however, will find certain features of the Emacs Lisp language such as the (self-)documentation system incredibly helpful and accommodating.

External references:

1. Sacha chua's site is a very good place to find more learning resources on Emacs.

   a. For those who need a more visual appeal on the Emacs learning path

   b. For those who would like to get the key bindings easily

2. Wikemacs is based on mediawiki, and thus has structured content, browsable categories and such. Start exploring !

Versions

Version	Release date
25.1	2016-09-17
24.5	2015-04-10
24.4	2014-10-20
24.3	2013-03-11
24.2	2012-08-27
24.1	2012-06-10
23.4	2012-01-29
23.3	2011-03-10

Version	Release date
23.2	2010-05-08
23.1	2009-07-29
22.3	2008-09-05
22.2	2008-03-26
22.1	2007-06-02
21.4	2005-02-06
21.3	2003-03-24
21.2	2002-03-18
21.1	2001-10-28

Examples

Installation or Setup

Detailed instructions on getting emacs set up or installed.

Official instructions are available on the GNU Emacs website.

Debian systems

On systems with the Debian package manager (such as Debian, Ubuntu, and Mint) Emacs can be installed via the simple command:

```
sudo apt-get install emacs
```

For a bleeding-edge release one can use the following ppa:

```
sudo apt-add-repository ppa:ubuntu-elisp/ppa
sudo apt-get install emacs-snapshot
```

Build for source

If your debian based distro does not have the version of emacs you want you can build it from scratch.

```
sudo apt-get build-dep emacs24 -y
```

```
cd /tmp/

wget http://alpha.gnu.org/gnu/emacs/pretest/emacs-25.0.93.tar.xz
tar -xvf emacs-25.0.93.tar.xz

cd emacs-25.0.93
./configure
make
sudo make install

rm -rf /tmp/emacs-25.0.93*
```

Redhat systems

On systems with the Redhat package manager (such as RHEL, CentOS, and Fedora Core) Emacs can be installed via the simple command:

```
sudo yum install emacs
```

Arch Linux

Emacs can be installed via the simple command:

```
sudo pacman -Syu emacs
```

Gentoo and Funtoo

On systems running Portage, Emacs can be installed via the simple command:

```
sudo emerge emacs
```

GSRC (GNU Source Release Collection)

Works on any GNU/Linux system for getting the latest version of emacs without using the system package manager (which may be out of date) or downloading the archive or binary. To install `gsrc` see it's documentation. Then:

```
cd gsrc
make -C gnu/emacs install
# add the binaries to your PATH
source ./setup.sh
```

Darwin systems

Homebrew

```
brew install emacs --with-cocoa # basic install
# additional flags of interest can be viewed by calling `brew info emacs`
brew linkapps emacs # to put a symlink in your Applications directory
```

MacPorts

```
sudo port install emacs
```

pkgsrc

```
sudo pkgin -y install emacs-24.5
```

App Bundle

Precompiled app bundles for the latest stable and development versions can be downloaded at https://emacsformacosx.com.

Windows

Chocolatey package manager

Emacs can be installed with

```
choco install emacs
```

Scoop package manager

Can be installed from extras bucket

```
scoop bucket add extras
scoop install emacs
```

Official Binary Installers

- stable
- latest

(Note that official binaries do not come with some libraries - e.g., libraries for image formats)

Other Binary Installers

- Emacs with pre-compilled AUCTeX and ESS
- 64-Bit GNU Emacs for MS Windows with optimization provides native and optimized 64-bit binary installer with unmodified source code from git master and release version, with JPEG, GIF, PNG, TIFF, SVG, XML2, and GnuTLS support out-of-box.

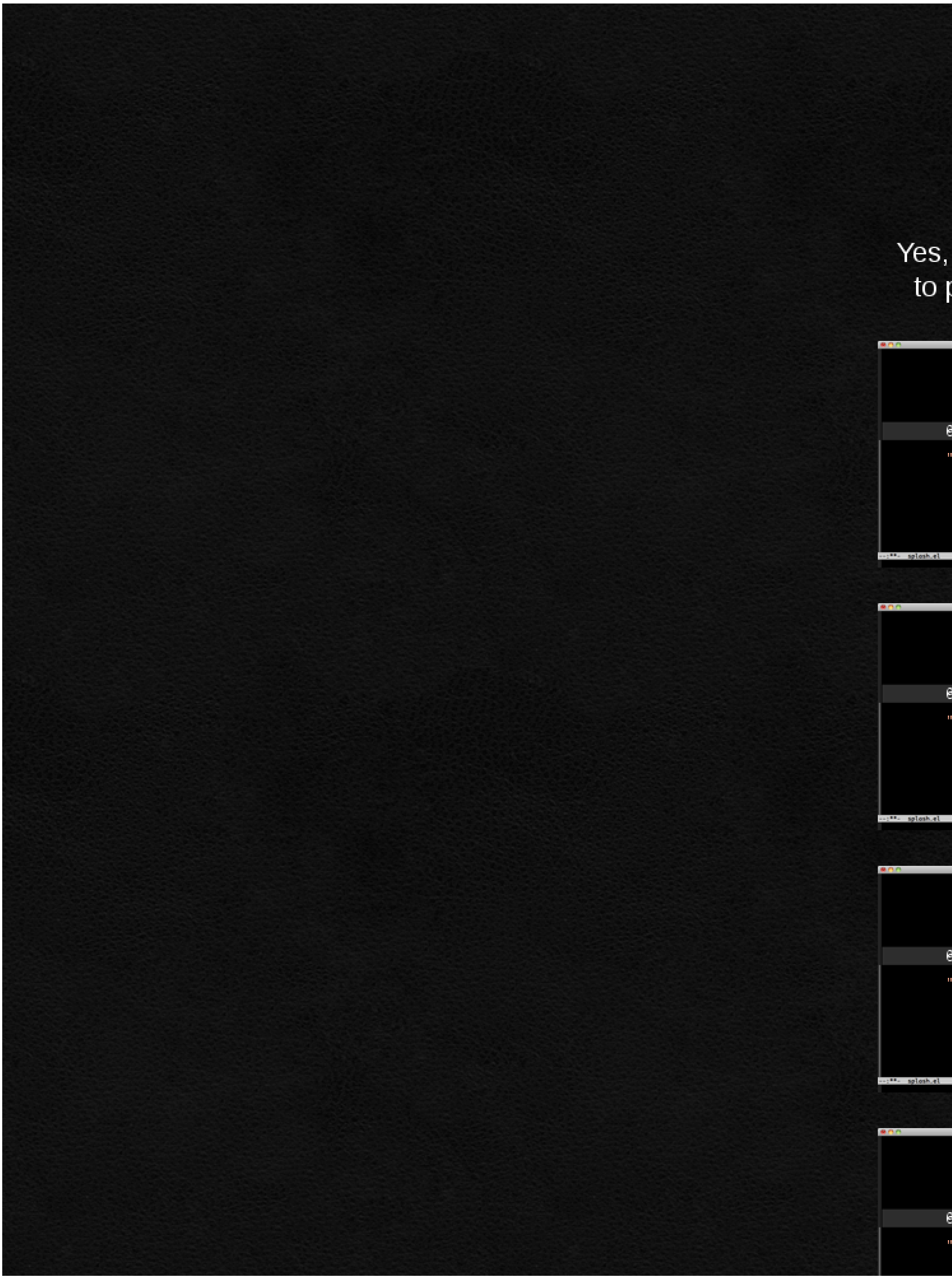
Interactive Emacs Tutorial

From within Emacs, type `C-h t` (Control-h, t) to get an excellent interactive tutorial within Emacs. The user learns basic navigation and editing by operating on the TUTORIAL text itself, as they read the tutorial. (Modifications to the tutorial are discarded when the tutorial is closed, so each time a user requests the tutorial, it's a clean default version of the tutorial.

Helpfully, the first thing in the tutorial is how to understand `C-<chr>` and `M-<chr>` references in the text. The second thing is how to page forward and backwards in the text.

Emacs Rocks Video Tutorials

Good video tutorials about Emacs can be found at emacsrocks.com.



Read Getting started with emacs online:

https://riptutorial.com/emacs/topic/986/getting-started-with-emacs

Chapter 2: Basic Keybindings

Examples

Quit Emacs

You can quit Emacs with the following keybinding:

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

Where `C` is the `control` key.

Suspend Emacs

You can suspend Emacs using the following keybinding :

```
C-z
```

It gets you back to your shell. If you want to resume your emacs session, enter `fg` in your terminal.

File handling

- Re-Save open file under the same filename (Save):

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

- Write as `filename` (Save As):

  ```
  C-x C-w filename
  ```

  The new file name will be prompted in the minibuffer.

- Create new file **or** load existing file (New / Load):

  ```
  C-x C-f filename
  ```

  With the mnemonic here for f meaning file. You will be prompted for a file path in the minibuffer.

- Visit alternate file

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

  If the file does not exist yet, you will be prompted the path of the file to create in the minibuffer.

Abort current command

Often you will get into a state where you have a partially typed command sequence in progress, but you want to abort it. You can abort it with either of the following keybindings:

`C-g`

`EscEscEsc`

Multiples windows or frames

"Window" in Emacs refers to what might otherwise be called a "pane" or "screen division". Some window manipulation commands include:

- Split current window horizontally: `C-x 2`
- Split current window vertically: `C-x 3`
- Select next window: `C-x o`
- Close current window: `C-x 0`
- Close all other windows, except the current one: `C-x 1`

A "frame" in Emacs is what might otherwise be called a "window". Frames are manipulated using these commands:

- Create new frame: `C-x 5 2`
- Delete current frame: `C-x 5 0`
- Delete other frames: `C-x 5 1`

Switching windows can be acheived using

- `S-left`, `S-right`, `S-up`, `S-down` (that is, `Shift` in conjunction with an arrow key) to switch to the neighboring window in a direction, or
- `C-x o` to switch to the next window.

Buffers

- Example of a buffer list

  ```
  CRM Buffer                Size  Mode              Filename[/Process]
  . * .emacs                3294  Emacs-Lisp        ~/.emacs
   %  *Help*                 101  Help
      search.c             86055  C                 ~/cvs/emacs/src/search.c
   %  src                  20959  Dired by name     ~/cvs/emacs/src/
    * *mail*                  42  Mail
   %  HELLO                 1607  Fundamental       ~/cvs/emacs/etc/HELLO
   %  NEWS                481184  Outline           ~/cvs/emacs/etc/NEWS
      *scratch*              191  Lisp Interaction
    * *
  Messages*            1554  Messages
  ```

  The first field of a line indicates:

  - ‘.’ the buffer is current.
  - ‘%’ a read-only buffer.
  - ‘*’ the buffer is modified.

- Select buffer. You can select out of any open buffer with the following keybinding:

```
C-x b
```

You will be prompted for the buffer name you wish to switch to.

- List buffers:

  ```
  C-x C-b
  ```

- Save-some-buffer, giving the choice which buffer to save or not:

  ```
  C-x s
  ```

- Kill one buffer:

  ```
  C-x k
  ```

- Operations on marked buffers:

  `S` Save the marked buffers

  `A` View the marked buffers in this frame.

  `H` View the marked buffers in another frame.

  `V` Revert the marked buffers.

  `T` Toggle read-only state of marked buffers.

  `D` Kill the marked buffers.

  `M-s a C-s` Do incremental search in the marked buffers.

  `M-s a C-M-s` Isearch for regexp in the marked buffers.

  `U` Replace by regexp in each of the marked buffers.

  `Q` Query replace in each of the marked buffers.

  `I` As above, with a regular expression.

  `P` Print the marked buffers.

  `O` List lines in all marked buffers which match a given regexp (like the function `occur').

  `X` Pipe the contents of the marked buffers to a shell command.

  `N` Replace the contents of the marked buffers with the output of a shell command.

  `!` Run a shell command with the buffer's file as an argument.

  `E` Evaluate a form in each of the marked buffers. This is a very flexible command. For example, if you want to make all of the marked buffers read only, try using (read-only-mode 1) as the input form.

`W` - As above, but view each buffer while the form is evaluated.

`k` - Remove the marked lines from the *Ibuffer* buffer, but don't kill the associated buffer.

`x` - Kill all buffers marked for deletion.

- Save-some-buffer, giving the choice which buffer to save or not:

  `C-x s`

- Switch to the next buffer:

  `C-x RIGHT`

- Switch to previous buffer:

  C-x LEFT

Search and Replace

In Emacs, basic search tool (`I-Search`) allows you to search after or before the location of your cursor.

- To search for *sometext* after the location of your cursor (`search-forward`) hit `C-s` *`sometext`*. If you want to go to the next occurence of *`sometext`*, just press `C-s` again (and so on for the next occurences). When cursor lands in the right location, press `Enter` to exit the search prompt.
- To search before the location of your cursor (`search-backward`), use `C-r` the same way you used before.
- To switch from `search-backward` to `search-forward`, press 2 times `C-s`. And press 2 times `C-r` to `search backward` when you're in `search-forward` prompt.
- Search and replace:

  `M-%` (or `Esc-%`) `oldtext Enter newtext Enter`

  - Confirm: `y`
  - Skip: `n`
  - Quit: `q`
  - Replace all: `!`

Region - Cut, Copy, Paste

- Set mark in cursor location:

  `C-space` or `C-@`

- Kill region (Cut):

  `C-w`

- Copy region to kill ring:

  `M-w` or `Esc-w`

- Yank (Paste) most recently killed:

  `C-y`

- Yank (Paste) next last killed:

  `M-y` or `Esc-y`

Kill

`kill`is the command used by Emacs for the deletion of text. The `kill` command is analogous to the `cut` command in Windows. Various commands exist that 'kills' one word (`M-d`), the rest of the line (`C-k`), or larger text blocks. The deleted text is added to the `kill-ring`, from which it can later be `yanked`.

Select and cut (kill)

Killing and yanking Similar to the `select-and-cut` feature in Windows, here we have `C-spc`. The key binding `C-spc` will start the selection, the user can move the mark with the help of arrow keys or other command to make a selection. Once selection is complete - push the `C-w` to kill the selected text

Some basic commands which can be used as quick reference for `kill` command (taken from Emacs tutorial)

```
    M-DEL       Kill the word immediately before the cursor
    M-d         Kill the next word after the cursor

    C-k         Kill from the cursor position to end of line
    M-k         Kill to the end of the current sentence
```

Yank

`yank`describes the insertion of previously deleted text, *e.g.* using `C-y` which yanks the most recently killed text. `Yank` command is analogous to the `paste` command in Windows.

Yank text killed previously

We know that the `kill` command adds the text killed to a `kill-ring`. To retrieve the deleted text from the `kill-ring` use `M-y` command repeatedly until the desired text is yanked.

(Note: For the `M-y` key to work the previous command should be a `YANK` otherwise it wouldn't work)

Cursor (point) movement

In addition to cursor movements using the arrow keys, Home, End, Page up, and Page down, emacs defines a number of keystrokes that can move the cursor over smaller or larger pieces of text:

By character:

- Backward character: `C-b`
- Forward character: `C-f`

By word

- Backward word: `M-b` (*i.e.* `Alt b`, or `Meta b`)
- Forward word: `M-f`

By line:

- Beginning of current line: `C-a`
- Beginning of current line first(non-space)character:`M-m`
- End of current line: `C-e`
- Previous line: `C-p`
- Next line: `C-n`

Entire buffer:

- Beginning of buffer: `M-<`
- End of buffer: `M->`

By 'block', depending on context (mode):

Typical key bindings:

- Backward sentence/statement: `M-a`
- Forward sentence/statement: `M-e`
- Beginning of function: `M-C-a`
- End of function: `M-C-e`

Prefix arguments

In order to move several 'steps' at once, the movement commands may be given a prefix argument by pressing `ESC` or `C-u` and a number before the listed keystrokes. For `C-u`, the number is optional and defaults to 4.
E.g. `ESC 3 C-n` moves 3 lines down, while `C-u M-f` moves 4 words forward.

Undo

To undo something you just did:

`C-_` or `C-x u` or `C-/`

Case

- Capitalize word: `M-c`
- Convert word to upper case: `M-u`
- Convert word to lower case: `M-l`

Key bindings notation

Emacs' documentation uses a consistent notation for all key bindings, which is explained here:

Key chords

A "key chord" is obtained by pressing two or more keys simultaneously. Key chords are denoted by separating all keys by dashes (`-`). They usually involve modifier keys, which are put up front:

- `C-`: control;
- `S-`: shift;
- `M-`: alt (the "M" stands for "Meta" for historical reasons).

Other keys are simply denoted by their name, like:

- `a`: the `a` key;
- `left`: the left arrow key;
- `SPC`: the space key;
- `RET`: the return key.

Examples of key chords thus include:

- `C-a`: pressing `control` and `a` simultaneously;
- `S-right`: pressing `shift` and `right` simultaneously;
- `C-M-a`: pressing `control`, `alt` and `a` simultaneously.

Key sequences

"Key sequences" are sequences of keys (or key chords), which must be typed one after the other. They are denoted by separating all key (or chord) notations by a space.

Examples include:

- `C-x b`: pressing `control` and `x` simultaneously, then releasing them and pressing `b`;
- `C-x C-f`: pressing `control` and `x` simultaneously, then releasing `x` and pressing `f` (since both chords involve the `control` modifier, it is not necessary to release it).

Using ESC instead of Alt

Key chords using the Alt modifier can also be entered as a key sequence starting with `ESC`. This can be useful when using Emacs over a remote connection that does not transmit Alt key chords, or when these key combinations are captured *e.g* by a window manager.

Example:

`M-x` can be entered as `ESC x`.

Describing key bindings in Emacs lisp files

The same notation that is described here can be used when defining key bindings in Emacs lisp files.

Example:

(global-set-key (kbd "C-x C-b") 'buffer-menu)
binds the key sequence `C-x C-b` to the command `buffer-menu`

Read Basic Keybindings online: https://riptutorial.com/emacs/topic/3436/basic-keybindings

Chapter 3: emacs has already very high quality, well organized documentation. why duplicate it?

Introduction

https://www.gnu.org/software/emacs/manual/html_node/emacs/index.html#Top

Examples

Keys

https://www.gnu.org/software/emacs/manual/html_node/emacs/Keys.html#Keys

3 Keys

Some Emacs commands are invoked by just one input event; for example, C-f moves forward one character in the buffer. Other commands take two or more input events to invoke, such as C-x C-f and C-x 4 C-f.

A key sequence, or key for short, is a sequence of one or more input events that is meaningful as a unit. If a key sequence invokes a command, we call it a complete key; for example, C-f, C-x C-f and C-x 4 C-f are all complete keys. If a key sequence isn't long enough to invoke a command, we call it a prefix key; from the preceding example, we see that C-x and C-x 4 are prefix keys. Every key sequence is either a complete key or a prefix key.

Read emacs has already very high quality, well organized documentation. why duplicate it? online: https://riptutorial.com/emacs/topic/9077/emacs-has-already-very-high-quality--well-organized-documentation--why-duplicate-it-

Chapter 4: Emacs nomenclature

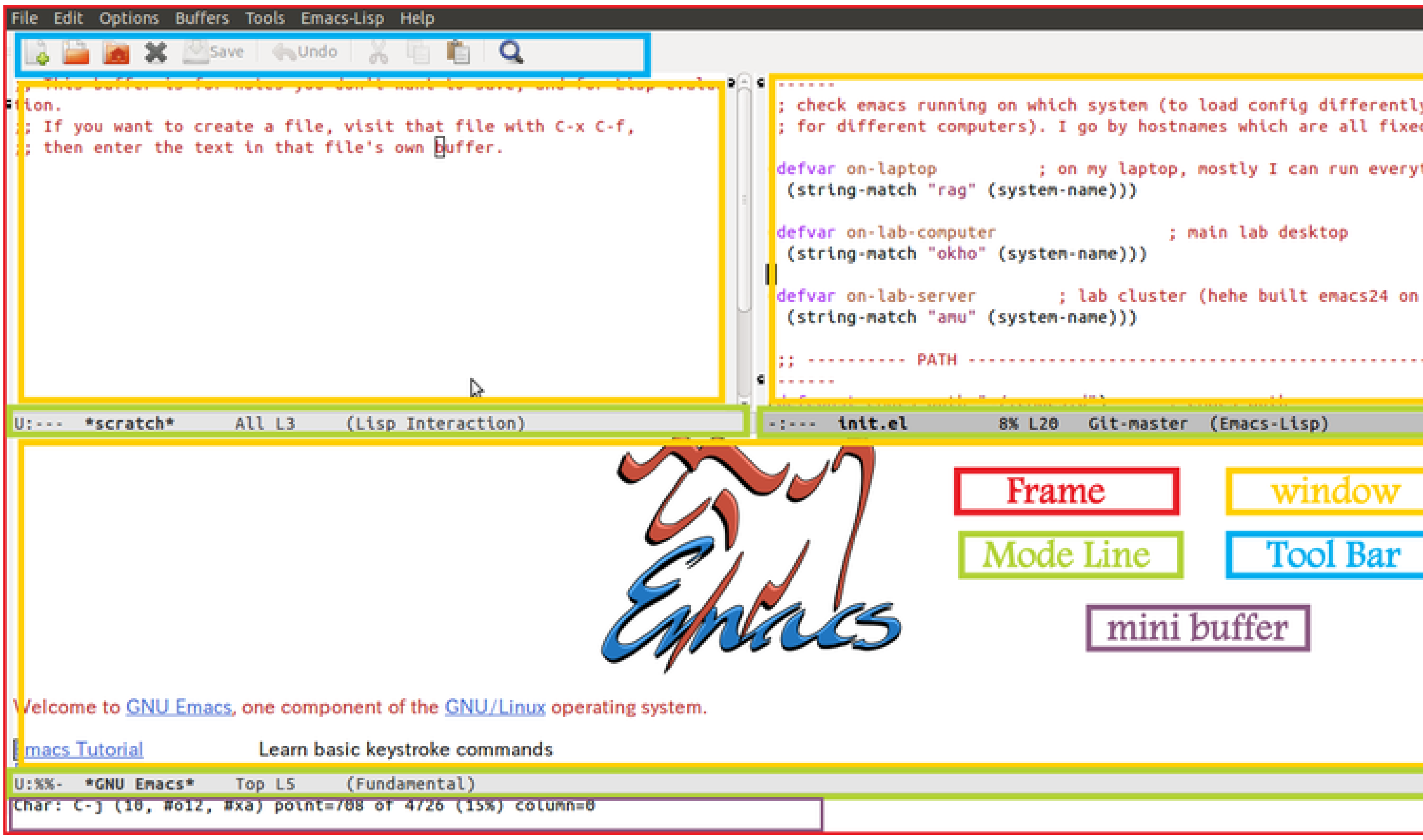
Examples

Files and buffers

In Emacs, *file* has the same meaning as in the operating system, and is used for permanent storage of data. A *buffer* is the internal representation of a file being edited. Files can be read into buffers using `C-x C-f`, and buffers can be written to files using `C-x C-s` (save file at its current location) or `C-x C-w` (write file to a different location, prompting for it - the equivalent of `Save as`).

Elements of the User Interface

Emacs's user interface uses terms that were coined early and can be unsettling to users used to a more modern terminology.



```
ion.
;; If you want to create a file, visit that file with C-x C-f,
;; then enter the text in that file's own buffer.
```

```
U:---  *scratch*      All L3     (Lisp Interaction)
```

```
; check emacs running on which system (to load config differently
; for different computers). I go by hostnames which are all fixed

defvar on-laptop          ; on my laptop, mostly I can run every
 (string-match "rag" (system-name)))

defvar on-lab-computer                   ; main lab desktop
 (string-match "okho" (system-name)))

defvar on-lab-server         ; lab cluster (hehe built emacs24 on
 (string-match "amu" (system-name)))

;; ---------- PATH ------------------------------------------
```

```
-:---  init.el        8% L20    Git-master   (Emacs-Lisp)
```

Welcome to GNU Emacs, one component of the GNU/Linux operating system.

Emacs Tutorial   Learn basic keystroke commands

```
U:%%-  *GNU Emacs*    Top L5     (Fundamental)
Char: C-j (10, #o12, #xa) point=708 of 4726 (15%) column=0
```

Frame

In Emacs, what is otherwise called a window (the area of the display used by a program) is called a *frame*. Emacs starts using one *frame*, though additional frames may be created using `C-x 5`.

Window

A *frame* contains one or more *windows* (otherwise usually called panes), each showing the content of one *buffer*. Each *frame* usually starts with only one *window*, but additional windows can be created by splitting existing ones ; either horizontally using `C-x 2` or vertically with `C-x 3`. See also Multiples windows or frames.

Buffer

The term *buffer* refers to the content displayed in a *window*. Such content may reflect the content of a file in the file system (or maybe an updated version that has not been saved to the disk yet), but more generally it can be any kind of text.

Mode line

At the bottom of each *window* is a *mode line*, which synthetically describes the *buffer* displayed in the window.

Tool Bar

In a similar way to many other softwares, a *tool bar* can be displayed at the top of each *frame*. Its contents may vary depending on the type of *buffer* being currently edited.

Minibuffer

A *minibuffer*, usually displayed at the bottom of each *frame*, allows interacting with Emacs. Each time a command asks for user input, it is prompted in the *minibuffer*. Conversely, messages displayed for the user to see are printed there.

Point, mark and region

Emacs uses the terms **point**, **mark**, and **region** to provide more precision about the selected text and position of the cursor. By understanding these terms, it'll help you understand and use other operations and functions.

The **point** is the place in a buffer where editing (*i.e.* insertion) is currently taking place, and is usually indicated by a cursor.
The **mark** is a marker placed anywhere in the buffer using commands like `set-mark-command` (`C-SPC`) or `exchange-point-and-mark` (`C-x C-x`).
The **region** is the area between point and mark, and many commands operate on the region to *e.g.* delete, spell check, indent, or compile it.

When you click your mouse on a location in a buffer, you're seeing the point. When you select text, you're setting the region (the selected text) and the mark (at the beginning of your selection).

Killing and yanking

Killing and *yanking* more or less correspond to what is usually called "cutting" and "pasting".

Killing

killing means deleting text, and copying it to the *kill-ring* (which could be seen as a sort of "clipboard" in the "cut & paste" terminology). The *kill ring* is so named because it stores several pieces of *killed* text, which can later be accessed in cyclic order.

Various commands exist that *kill* one word (`M-d`), the rest of the line (`C-k`), or larger text blocks (such as the currently selected region: `C-w`).

Other commands exist, that save text to the *kill ring*, without actually *killing* it (in a similar way to "copying" in modern terminologies). For example, `M-w`, which acts on the currently selected region.

Yanking

Entries in the *kill ring* can later be *yanked* back into a buffer. One can typically *yank* the most recently *killed* text using *e.g* `C-y` (which is similar to the "paste" operation in a more modern terminology). But other commands can access and *yank* older entries from the *kill ring*.

Modes

Major mode

Emacs can adapt its behaviour to the specific type of text edited in a *buffer*. The set of specific Emacs customizations for a particular type of text is called a "major mode". Each buffer has exactly one *major mode* depending on its content type.

Major modes can change the meaning of some keys, define syntax highlighing or indentation rules, and install new key bindings (usually beginning with `C-c`) for mode-specific commands. Emacs ships with a wide range of major modes, falling into three main categories:

- support for text (e.g. markup languages),
- support for programming languages,
- applications within emacs (e.g. dired, gnus, ...). Buffers using this last group of major modes are usually not associated to files, but rather serve as a user interface.

Minor mode

Minor modes are optional features that can be turned on and off. *Minor modes* can be enabled for specific buffers (buffer-local modes) or all buffers (global modes). In contrast to *major modes* any number of *minor mode* can be activated for a given buffer.

Emacs provides lots of minor modes. A few examples include:

- *Auto-fill mode* to automatically wrap text lines as you type.
- *Flyspell mode* to highlight spelling errors as you type.
- *Visual Line mode* to wrap long lines to fit the screen.
- *Transient Mark mode* to highlight the current region.

Read Emacs nomenclature online: https://riptutorial.com/emacs/topic/3683/emacs-nomenclature

Chapter 5: Helm

Examples

Installing helm via MELPA

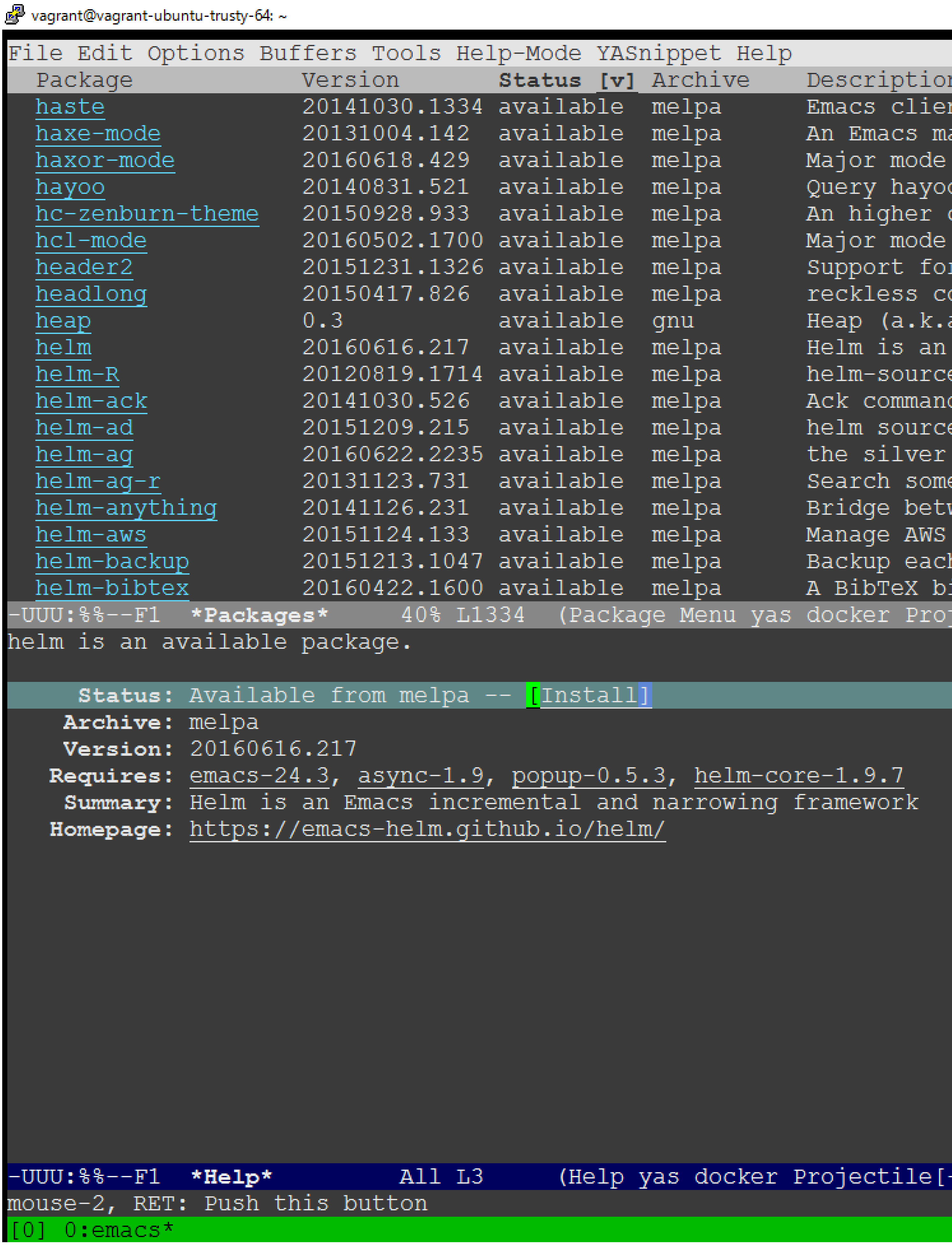
From emacs 24.4 package.el is avalable, and one way to install helm is to do it via MELPA. First, add the MELPA repository as package archive by putting following code somewhere in your `~/.emacs` (or, `~/.emacs.d/init.el`).

```
(require 'package)

;; add the repository before the package-initialize.
(add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/"))

(package-initialize)
```

Next, enter `M-x list-packages` to see the avaiable package list. Search for `helm` entry, put your cursor on the `helm` entry, press `RET`. You'll see the package information buffer. Put your cursor on `[Install]`, and press `RET`. Helm will be installed. The package list window and the package information window is shown in following image.



```
vagrant@vagrant-ubuntu-trusty-64: ~
File Edit Options Buffers Tools Help-Mode YASnippet Help
  Package                  Version       Status  [v] Archive     Descriptior
  haste                    20141030.1334 available   melpa       Emacs clier
  haxe-mode                20131004.142  available   melpa       An Emacs ma
  haxor-mode               20160618.429  available   melpa       Major mode
  hayoo                    20140831.521  available   melpa       Query hayoo
  hc-zenburn-theme         20150928.933  available   melpa       An higher c
  hcl-mode                 20160502.1700 available   melpa       Major mode
  header2                  20151231.1326 available   melpa       Support for
  headlong                 20150417.826  available   melpa       reckless co
  heap                     0.3           available   gnu         Heap (a.k.a
  helm                     20160616.217  available   melpa       Helm is an
  helm-R                   20120819.1714 available   melpa       helm-source
  helm-ack                 20141030.526  available   melpa       Ack command
  helm-ad                  20151209.215  available   melpa       helm source
  helm-ag                  20160622.2235 available   melpa       the silver
  helm-ag-r                20131123.731  available   melpa       Search some
  helm-anything            20141126.231  available   melpa       Bridge betw
  helm-aws                 20151124.133  available   melpa       Manage AWS
  helm-backup              20151213.1047 available   melpa       Backup each
  helm-bibtex              20160422.1600 available   melpa       A BibTeX bi
-UUU:%%--F1  *Packages*      40% L1334  (Package Menu yas docker Proj
helm is an available package.

     Status: Available from melpa -- [Install]
    Archive: melpa
    Version: 20160616.217
   Requires: emacs-24.3, async-1.9, popup-0.5.3, helm-core-1.9.7
    Summary: Helm is an Emacs incremental and narrowing framework
   Homepage: https://emacs-helm.github.io/helm/











-UUU:%%--F1  *Help*          All L3     (Help yas docker Projectile[-
mouse-2, RET: Push this button
[0] 0:emacs*
```

Read Helm online: https://riptutorial.com/emacs/topic/5341/helm

Chapter 6: Help Within Emacs

Remarks

Emacs is described as a self-documenting editor, and provides lots of information on how to use it within the editor itself. Amongst the entry points to this documentation is a tutorial, information about what functions is available related to a given topic,a information about the bindings between keystrokes and functions.The documentation is accessed using the prefix `C-h`, *i.e.* `Ctrl h`, or `F1`, with a list of further choices available by pressing `?`

Examples

Emacs Tutorial

`C-h t` runs the function `help-with-tutorial`, which opens a buffer containing a tutorial on the basic editing functionality of emacs, including moving around in text, and working with files, buffers, and windows.

Available Functions and Key Bindings

Pressing `C-h a` will run the emacs function `apropos-command` which makes emacs prompt for words (or a regexp) to search for. It will then show a buffer containing a list of names and descriptions related to that topic, including key bindings for each of the functions available via keystrokes.

Pressing `C-h m` (`describe-mode`) gives a buffer describing the major and minor modes in effect, including listings of available functions and their key bindings.

Pressing `C-h b` (`describe-bindings`) gives a buffer listing all current key bindings. The listing includes global bindings as well as bindings for the active major and minor modes in the current buffer.

Key Binding Documentation

`C-h k` runs the function `describe-key`, which looks up the function mapped to the key strokes provided, and presents a description of the function which will be run when these keys are pressed.

`C-h c` runs the function `describe-key-briefly`, which only displays the function name mapped to given key sequence.

Function Documentation

`C-h f` runs the function `describe-function`, which displays information on the usage and purpose of a given function. This is especially useful for functions that do not have a mapped key binding that can be used for documentation lookup via `C-h k`.

Read Help Within Emacs online: https://riptutorial.com/emacs/topic/4736/help-within-emacs

Chapter 7: Magit

Introduction

Magit is an interface to the version control system Git, implemented as an Emacs package. It allows you to interact with git in Emacs.

Remarks

Magit is an interface to the version control system Git, implemented as an Emacs package. Magit aspires to be a complete Git porcelain. While we cannot (yet) claim, that Magit wraps and improves upon each and every Git command, it is complete enough to allow even experienced Git users to perform almost all of their daily version control tasks directly from within Emacs. While many fine Git clients exist, only Magit and Git itself deserve to be called porcelains.

Note that Magit can interface itself to Github (with Magithub, see also Github integration in Emacs) and that Emacs also has packages to work with Gitlab, Bitbucket and others.

Examples

Installation

You can install Magit from MELPA with:

```
M-x package-install RET magit RET
```

Basic usage: commit unstaged edits within an existing repo

```
M-x magit-status
s RET <file-to-stage> RET
c c <commit message>
C-c C-c
q
```

Read Magit online: https://riptutorial.com/emacs/topic/3909/magit

Chapter 8: Manage bookmarks within Emacs

Examples

How to bookmark frequently used files

Use the following commands to create bookmarks and access bookmarks from within Emacs.

Let us say that you are editing a file called `foobar.org` and suppose that you visit this file frequently to edit / view contents.

It would be convenient to access this file with couple of key strokes rather than navigate through the file structure (Dired) and visit the file.

Steps:

1. Open `foobar.org` for once by navigating to the file (*visit the file* in Emacs lingo)
2. While the file is open type `C-xrm` this will prompt you to provide the bookmark name for the file. Let us say `foobar` in this case.
3. Close the file (`C-xk` - kill buffer) - save if required
4. Now to visit the file, just type `C-xrl` - this will populate a list which will contain `foobar`.
5. Select `foobar` and hit `Enter` key
6. Use `M-xbookmark-delete` to delete any unnecessary bookmark of a file.

Note: Deleting a bookmark is analogous to deleting a shortcut in your Windows desktop.
The main file will be safe in its location and only the listing of the file from the bookmark menu will be removed.

Read Manage bookmarks within Emacs online: https://riptutorial.com/emacs/topic/5837/manage-bookmarks-within-emacs

Chapter 9: Org-mode

Remarks

Org is a mode for keeping notes, maintaining TODO lists, and project planning with a fast and effective plain-text system. It also is an authoring system with unique support for literate programming and reproducible research.

org Mode official site

Examples

Markup syntax

Org provides a full markup language which helps structuring the document, and is reflected as accurately as possible when exporting to other formats (like HTML or LaTeX).

Structure

Document title

```
#+TITLE: This is the title of the document
```

Sectioning

```
* First level
** Second level
```

Lists

```
Ordered list (items can also be numbered like '1)', with a perenthesis):
1. foo
2. bar
3. baz
```

```
Unordered list (items can also start with '+' or '*'):
- foo
- bar
- baz
```

```
Description
- lorem ipsum :: this is example text
- foo bar :: these are placeholder words
```

Checkboxes

Every item in a plain list can be made into a checkbox by starting it with the string '[ ]'.

```
* TODO [2/4] (or [50%])
  - [-] call people [1/3]
    - [ ] Peter
      - [X] Sarah
      - [ ] Sam
  - [X] order food
  - [ ] think about what music to play
  - [X] talk to the neighbors
```

- `C-c C-c` org-toggle-checkbox
- `C-c C-x C-b` org-toggle-checkbox
- `M-S-` org-insert-todo-heading
- `C-c C-x o` org-toggle-ordered-property
- `C-c #` org-update-statistics-cookies

Emphasis and monospace

```
You can make words *bold*, /italic/, _underlined_, =verbatim=
and ~code~, and, if you must, '+strike-through+'.
```

Text in the code and verbatim string is not processed for Org mode specific syntax, it is exported verbatim.

Links and references

Links

Org-mode will recognize URL formats and activate them as clickable links. However, links can be explicitly declared like this:

```
You will find more information in the [[http://orgmode.org/org.html][Org Manual]].
```

or alternatively :

```
The org manual is located here: [[http://orgmode.org/org.html]]
```

Footnotes

Footnotes can either be named:

```
See the org manual[fn:manual] to get more details.
...
[fn:manual] You will find it here: http://orgmode.org/org.html
```

or anonymous and inline:

```
See the org manual[fn:: You will find it here: http://orgmode.org/org.html]
to get more details.
```

Basic Key Bindings

To cycle the level of outline shown:

- `Tab` Cycle outline level for one heading
- `Shift-Tab` Cycle outline level for the whole document

To cycle through `TODO` states:

- `Shift-Right Arrow`
- `Shift-Left Arrow`

To increase or decrease hierarchical level for a heading

- `Meta-Right Arrow` Make lower level ("increase indent")
- `Meta-Left Arrow` Make higher level ("decrease indent")

To cycle the priority for a given heading:

- `Shift-Up Arrow`
- `Shift-Down Arrow`

To move a heading up or down:

- `Meta-Up Arrow`
- `Meta-Down Arrow`

(`Meta` refers to different keys on different keyboards. Most often it is either `Alt` or `▯`).

Code blocks

To add a code block, surround it with `#+BEGIN_SRC language` and `#+END_SRC`. *language* should correspond to the major mode for the language in question, e.g. the major mode for Emacs Lisp is `emacs-lisp-mode`, so write `#+BEGIN_SRC emacs-lisp`.

```
#+BEGIN_SRC emacs-lisp
(defun hello-world ()
  (interactive)
  (message "hello world"))
#+END_SRC

#+BEGIN_SRC python
print "hello world"
```

```
#+END_SRC
```

You can open the code block in a separate buffer by typing `C-c '` (for `org-edit-special`). If you don't have the major mode for the specified language, that will give an error message such as `No such language mode: foo-mode`.

If the content you want to put in the block is not in any programming language, you can use `#+BEGIN_EXAMPLE` and `#+END_EXAMPLE` instead.

```
#+BEGIN_EXAMPLE
output from a command I just ran
#+END_EXAMPLE
```

There are easy templates for both of these. At the beginning of the line, type either `<s` or `<e`, and then hit `TAB`. It will expand into a block with begin and end markers for `SRC` or `EXAMPLE`, respectively.

These markers are all case insensitive, so you can write `#+begin_src` etc instead if you prefer.

Tables

```
| Name  | Phone | Age |
|-------+-------+-----|
| Peter | 1234  | 17  |
| Anna  | 4321  | 25  |
```

To add a Table in org-mode, simply surround your columns with a bar (`|`)

```
| column1 | column2 | this column is wider |
```

When you press `Return` from inside a column, org-mode will automatically create a new row with the bars.

- `Tab` and `Return` will respectively move to the next cell or row (or create a new one if there isn't any)
- You can swap the rows and columns around with `M-ArrowKey`
- `M-S-Down` and `M-S-Right` will respectively create a row (above the current) and a column (on the left of the current)
- `M-S-Up` and `M-S-Left` will respectively remove the current row and the current column
- `C-c i` create a separator

Read Org-mode online: https://riptutorial.com/emacs/topic/6259/org-mode

Chapter 10: Package Management

Examples

Automatic package installation on emacs start-up

```
;; package.el is available since emacs 24
(require 'package)

;; Add melpa package source when using package list
(add-to-list 'package-archives '("melpa" . "http://melpa.org/packages/") t)

;; Load emacs packages and activate them
;; This must come before configurations of installed packages.
;; Don't delete this line.
(package-initialize)
;; `package-initialize' call is required before any of the below
;; can happen

;; If you do not put the "(package-initialize)" in your ~/.emacs.d/init.el (or
;; ~/.emacs), package.el will do it for you starting emacs 25.1.

;; Below manual maintenance of packages should not be required starting emacs
;; 25.1 with the introduction of `package-selected-packages' variable. This
;; variable is automatically updated by emacs each time you install or delete a
;; package. After this variable is synced across multiple machines, you can
;; install the missing packages using the new
;; `package-install-selected-packages' command in emacs 25.1.

;; To clarify, below technique is useful on emacs 24.5 and older versions.
;; Request some packages:
(defconst my-package-list '()
  "List of my favorite packages")

(defvar my-missing-packages '()
  "List populated at each startup that contains the list of packages that need
to be installed.")

(dolist (p my-package-list)
  (when (not (package-installed-p p))
    (add-to-list 'my-missing-packages p)))

(when my-missing-packages
  (message "Emacs is now refreshing its package database...")
  (package-refresh-contents)
  ;; Install the missing packages
  (dolist (p my-missing-packages)
    (message "Installing `%s' .." p)
    (package-install p))
  (setq my-missing-packages '()))
```

References

- Comparison of package repos
- Blog post on user selected packages feature in emacs 25.1

Automatic Package Installation with use-package

```
;; disable automatic loading of packages after the init file
(setq package-enable-at-startup nil)
;; instead load them explicitly
(package-initialize)
;; refresh package descriptions
(unless package-archive-contents
   (package-refresh-contents))

;;; use-package initialization
;;; install use-package if not already done
(if (not (package-installed-p 'use-package))
    (progn
      (package-refresh-contents)
      (package-install 'use-package)))
;;; use-package for all others
(require 'use-package)

;; install your packages
(use-package helm
  :ensure t)
(use-package magit
  :ensure t)
```

Automatic package management using Cask

Cask is a project management tool which can be also used to easily manage your local emacs configuration.

Installing cask is easy. You can either run the following command on the command-line:

```
 curl -fsSL https://raw.githubusercontent.com/cask/cask/master/go | python
```

Or if you are on a mac, you can install it using `homebrew`:

```
brew install cask
```

Once installed, you create a `Cask` file. Cask files list all package dependencies which should be included in your configuration. You can create a new Cask file at the root of your `~/.emacs` directory.

You will also need to initialize Cask in your `~/.emacs.d/init.el`. If you installed using homebrew, add these lines:

```
(require 'cask "/usr/local/share/emacs/site-lisp/cask/cask.el")
(cask-initialize)
```

Or you can supply the path to cask, if you used the install script:

```
(require 'cask "~/.cask/cask.el")
(cask-initialize)
```

A simple Cask file looks like this:

```
(source gnu)
(source melpa)

(depends-on "projectile")
(depends-on "flx")
(depends-on "flx-ido")
```

Here we are specifying source repositories to look for packages in. Then we are specifying that we want the `projectile`, `flx`, and `flx-ido` packages installed.

Once you have a Cask file, you can install all the dependencies with the follwoing command on the command-line:

```
cask install
```

Automatic Package Management with el-get

el-get is an open source package management system for GNU Emacs. el-get works with `melpa`, as well as with many common version control systms. Its documentation includes a simple self-installer for your `.emacs`:

```
(unless (require 'el-get nil t)
  (url-retrieve
    "https://raw.github.com/dimitri/el-get/master/el-get-install.el"
    (lambda (s)
      (let (el-get-master-branch)
        (goto-char (point-max))
        (eval-print-last-sexp)))))

(el-get 'sync)
```

el-get maintains package installations in a directory structure at `~/.emacs.d/el-get`. It loads definitions from `~/.emacs.d/el-get/.loaddefs.el` and tracks package status with `~/.emacs.d/el-get/.status.el`. `(el-get 'sync)` installs or removes packages to bring the actual machine state in sync with the package `.status.el`.

el-get is self-hosted - here is its own status from `.status.el`:

```
(el-get status "installed" recipe
  (:name el-get :website "https://github.com/dimitri/el-get#readme" :description "Manage the
external elisp bits and pieces you depend upon." :type github :branch "master" :pkgname
"dimitri/el-get" :info "." :compile
         ("el-get.*\\.el$" "methods/")
         :features el-get :post-init
         (when
             (memq 'el-get
                   (bound-and-true-p package-activated-list))
```

```
            (message "Deleting melpa bootstrap el-get")
            (unless package--initialized
              (package-initialize t))
            (when
                (package-installed-p 'el-get)
              (let
                  ((feats
                    (delete-dups
                     (el-get-package-features
                      (el-get-elpa-package-directory 'el-get)))))
                (el-get-elpa-delete-package 'el-get)
                (dolist
                    (feat feats)
                  (unload-feature feat t))))
            (require 'el-get))))
```

Read Package Management online: https://riptutorial.com/emacs/topic/2414/package-management

Chapter 11: Starter Kits

Remarks

Starter kits enable *new users* to start using Emacs quickly and avoid some of the setup hurdles that come from a mature system like Emacs -- one that has grown through decades of evolution and naturally has some historical quirks. *Experienced users* also benefit from having a kit configuration of extensions that are curated by others.

It requires considerable effort to maintain a set of packages and settings that will continue to work well together as packages improve (or bit-rot) over time. Many Emacs users don't desire to do this maintenance, so they turn to starter kits. Assembly and maintenance of a kit bears a small-scale resemblance to management of a Linux distribution.

Themes and Customization

Some starter kits are themed; e.g., for specific programming language environments, or music creation, or emulation of another editor. Others aim to provide a kitchen sink of bundling comfortable/productive modules for as many situations or languages as possible.

Most starter kits have provisions for extension and customization. A user will override particular key bindings and settings, and be able to add packages that are not yet provided.

Popular Kits

There are many starter kits available. In theory, anyone who publishes their `~/.emacs.d` has created one. But a handful have become popular and well maintained by one or more individuals. Some examples (in order of subjective popularity based on Github stars) include Spacemacs, Prelude, Purcell, Emacs Starter Kit, Magnars, and Emacs Live. More details are listed in the **Examples** section above and more starter kits are listed on this wiki.

A notable "micro-kit" is Sane Defaults, providing a handful of settings to remove some of Emacs' default surprise-to-newcomers behaviors.

Is a starter kit needed?

Although there is some controversy around using starter kits, for many the benefits can far outweigh the cost figuring out how to harmonize a dynamic Emacs setup. Arguments against starter kits usually pertain to: users being unaware of some of the nuances and native behavior of Emacs, being difficult to debug, and even making Emacs look more like a foreign editor (Spacemacs).

Examples

Spacemacs

Spacemacs is a popular starter kit for emacs. It features a robust package management solution and centers around emacs's popular evil mode, which provides many of the keybindings from vim.

It is called Spacemacs because it uses the `Space` key as the leader key (the idea is similar to Vim's leader key).

Installation is pretty easy. Just download and install the standard emacs distribution and then clone the git repo:

```
git clone https://github.com/syl20bnr/spacemacs ~/.emacs.d
```

Or, you can download a zip locally from the website and just copy it to `~/.emacs.d`

Once you have it downloaded (cloned), launch it, then press the space bar to explore the interactive list of carefully-chosen key bindings. You can also press the home buffer's [?] button for some first key bindings to try.

Prelude

Prelude is another popular starter kit. It features good support for various programming languages out-of-the-box including, notably - clojure. On *nix systems it can be installed with the following command:

```
curl -L https://git.io/epre | sh
```

emacs-live

emacs-live is another popular emacs starter kit, with an additional focus on live music coding using overtone.

You can install it in 2 ways:

1. On *nix (e.g. linux, OSX, etc.) systems, run the following command on the command-line:

   bash <(curl -fksSL https://raw.github.com/overtone/emacs-live/master/installer/install-emacs-live.sh)

2. - Download the zip from the github page.
   - Backup your current `~/.emacs.d` in your home directory
   - extract the zip you downloaded and move it to `~/.emacs.d`:

Scimax

Scimax is an Emacs starter kit focused on reproducible research, targeted mainly at scientists and engineers. Scimax customizes Org-Mode with features that make cross-referencing, exporting, and coding (in particular Python), simpler.

Installation instructions can be found on the landing page of the project.

Read Starter Kits online: https://riptutorial.com/emacs/topic/1960/starter-kits

Chapter 12: The Many Variants Of Emacs

Introduction

Most of this documentation implicitly or explicitly applies to GNU Emacs. This may be the most well known variant of Emacs, as well as the source of several forks, and the target of some merges.

This topic discusses some of the variants of Emacs one may encounter, and their primary differences from GNU Emacs.

Examples

Spacemacs

Spacemacs (http://spacemacs.org/) is a variant of Emacs that attempts to end the long-term conflict between Emacs and vim users, by making an Emacs that behaves like vim.

Read The Many Variants Of Emacs online: https://riptutorial.com/emacs/topic/9456/the-many-variants-of-emacs

Credits

S. No	Chapters	Contributors
1	Getting started with emacs	Adobe, Community, ebpa, Ehvince, eyqs, IntFloat, Jack Henahan, joon, legoscia, mellowmaroon, Michel de Ruiter, Nemanja Trifunovic, omul, pcurry, Prasanna, salotz, squiter
2	Basic Keybindings	Adeel Ansari, Arjun J Rao, boehm_s, Francesco, Idan, Jeff Bencteux, julienc, leeor, Meaningful Username, mellowmaroon, Nikana Reklawyks, Prasanna, SuperBear, Tej Chajed, Terje D.
3	emacs has already very high quality, well organized documentation. why duplicate it?	erjoalgo
4	Emacs nomenclature	Doug Harris, Francesco, Nikana Reklawyks, Stephen Leppik, Terje D.
5	Helm	Yuki Inoue
6	Help Within Emacs	mellowmaroon, Terje D., ygram
7	Magit	boehm_s, Ehvince, glallen, mellowmaroon, squiter
8	Manage bookmarks within Emacs	Francesco, Prasanna
9	Org-mode	Christopher Bottoms, Francesco, legoscia, SuperBear, Wazam
10	Package Management	Adobe, Kaushal Modi, leeor, pcurry, salotz, squiter
11	Starter Kits	dangom, Ehvince, Kaushal Modi, leeor, Micah Elliott, Xinyang Li
12	The Many Variants Of Emacs	pcurry