Level up your notes with Org

Nick Anderson

nick@cmdln.org

Table of Contents

Introduction

Org mode is for keeping notes, maintaining TODO lists, planning projects and authoring documents with a fast and effective plain text system. – http://orgmode.org/

Getting started with Spacemacs

Spacemacs is a "community-driven Emacs distribution".

*"The best editor is neither Emacs nor Vim, it's Emacs **and** Vim!".*

– spacemacs.org

Why Spacemacs?

- As a **vim** user I can easily use Emacs.
- Managing configs is much easier.
- **You** don't have to use vim key bindings.

Backup

First backup your existing Emacs config if you have one.

```
mkdir ~/emacs_backup
mv ~/init.el ~/emacs_backup/
mv .emacs.d ~/emacs_backup/
```

Install

Then install spacemacs (I assume you already have Emacs installed).

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

Configuring Spacemacs for Org

~/.spacemacs

- Generate one with defaults

```
<SPC> : dotspacemacs/install RET
```

- To open the installed dotfile:

```
<SPC> f e d
```

Note: The `.spacemacs` used for this presentation is documented at the end of the presentation. You may want to refer to the tangled `dot-spacemacs`.

Layers

- Self contained units of configuration

Enable org layer

Enable the `org` layer in `dotspacemacs/layers`.

```
(defun dotspacemacs/layers ()
  "Configuration Layers declaration.
You should not put any user code in this function besides modifying the variable
values."
  (setq-default
   dotspacemacs-configuration-layers '( org )))
```

Agenda files

TODO items, **time-stamped** items, and **tagged headlines** can be scattered throughout a file or even a number of files. Agenda views allow Org to select entries based on various criteria for display in a separate buffer.

Tell org where you keep your notes. Make sure this presentation is included!

```
;; Set the org-agenda files
;; Subdirectories must be explicitly specified.
(setq org-agenda-files
  '("~/org" "~/src/presentations/Level-up-your-notes-with-Org"))
```

Using Org mode in Spacemacs

Make a new entry

In Org each entry is simply a line starting with one or more *. To insert a new headline after the current one type `<comma> h i`.

Capture a new TODO (or other standard type of entry)

Capture templates make capturing new entries quick and easy.

Note: The configuration used for this presentation allows you to capture a new TODO by typing `<comma> c t`.

Tag entries for search findme

You can tag headlines so that they are easy to find later. With the insertion point on a header type `<comma>` `:` to tag the entry.

- Separate tags with `:`
- Spaces aren't allowed

Agenda views

From an org file:

- `<comma> a`

From anywhere in Spacemacs:

- `SPC a o o`

Searching for entries

CLI Tools

Since this is all just plain text, you can use your favorite command line utilities like ag, ack or grep.

Agenda Searches

- Search for entries tagged with `findme`

  From within an org file type `<comma> a m` and type findme.

Stay focused

This is a good way to keep from being distracted.

- org-narrow-to-subtree <comma> n
- widen <comma> N

Clocking Time

- org-clock-in <comma> I
- org-clock-out <comma> O

Note: Try typing : org-clock-report

Org Markup

Org has its own markup syntax which is **really** beneficial for exporting.

- Markup your text, you will thank me later
- This is **key to getting compliments**

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.
```

You can make words **bold**, *italic*, <u>underlined</u>, `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.

Note: If you can't remember the markup you can access these formatting options by typing `<comma>` x. Try doing this after highlighting a word.

Block quotes

```
#+BEGIN_QUOTE
Clever remark -- me
#+END_QUOTE
```

> *Clever remark – me*

Note: Try typing <q followed by the <TAB> key.

Example blocks

```
#+BEGIN_EXAMPLE
Are useful for rendering mutlilne blocks verbatim.
#+END_EXAMPLE
```

```
Are useful for rendering mutlilne blocks verbatim.
```

Note: Try typing <e followed by the <TAB> key.

Code blocks

Source code can be included in Org mode documents using a 'src' block.

```
#+BEGIN_SRC cfengine3
bundle agent main {
  reports: "Hello World!";
}
#+END_SRC
```

They render nicely when exported

```cfengine3
bundle agent main {
  reports: "Hello World!";
}
```

Note: Try typing <s followed by the <TAB> key. Many languages are supported.

Links

You can link to other nodes in a document or to external places.

```
You can link to [[Emphasis and monospace][other nodes in a document]] or to [[http://lmgtfy.com][external places]].
```

[[TARGET] [TEXT]]

Pictures

Worth a thousand words

```
[[./media/best-nyan-cat-gif-963.gif]]
```

Tables

```
| Tables | Are     |
|--------+---------|
| easy   | to make |
```

Tables	Are
easy	to make

Different kinds of notes

- Meeting minutes
- TODOs
- Random things to remember
- **Technical notes**

Literate Devops (for technical notes)

- Literate Devops
- Literate Devops Demo

Executable code blocks

Some languages support executing code blocks. Try placing the insertion point within the code block and typing <comma> <comma>.

```
#+BEGIN_SRC sh :exports both :wrap EXAMPLE
  /bin/echo "Hello $(date)"
#+END_SRC
```

Execution results can be exported.

```
Hello Tue May 17 16:48:07 CDT 2016
```

Execute code blocks on remote hosts with Tramp

```
#+BEGIN_SRC sh :exports both :wrap EXAMPLE :dir /ssh:user@example.host.tld:
  hostname
#+END_SRC

example.host.tld
```

Getting Compliments

Comments like these are a regular occurrence.

> *"I really love the reporting you use. What is that?" – Todd Rix*

> *"Dammit, Nick!" – Jimis*

> *"Impressive notes!" – Thomas Ryd*

Exporting Notes

Exporting your notes makes **all** the difference.

Exporting to HTML

The default html export is not bad, but to really wow people with html exported versions of your notes check out org-html-themes.

I like readtheorg.

```
#+SETUPFILE: ~/src/org-html-themes/setup/theme-readtheorg.setup
```

Exporting for Presentations

I made this whole presentation in org-mode and I used the `ox-reveal` package to export it. This allows me to version my presentations like any other plain text source.

Narrow before exporting

Narrowing before exporting can be useful so you share only a subset of notes.

Clocktables for the PHB

If you are good about tracking your time and you have a PHB, checkout `: org-clock-report`.

Table 1: Clock summary at [2016-05-18 Wed 15:47]

Headline	Time	
Total time	**0:02**	
Clocktables for the PHB		0:02

Summary Tips

- Use org markup!
- Start with one file
- Keep it simple
- Let it grow organically
- Start recording the compliments you get right away!

Thanks!

Happy Orging

TODO Deliver presentation at KLF2016

DONE Document `.spacemacs`

dotspacemacs/layers

Layers

These are the layers that are needed for this presentation. I have a cfengine snippet in the presentation, so I include that layer. But note it's in the `develop` branch of spacemacs still.

```
;; -*- mode: emacs-lisp -*-
;; This file is loaded by Spacemacs at startup.
;; It must be stored in your home directory.

(defun dotspacemacs/layers ()
  "Configuration Layers declaration.
You should not put any user code in this function besides modifying the variable
values."
  (setq-default
   dotspacemacs-configuration-layers
   '(
     ;; ----------------------------------------------------------------
     ;; Example of useful layers you may want to use right away.
     ;; Uncomment some layer names and press <SPC f e R> (Vim style) or
     ;; <M-m f e R> (Emacs style) to install them.
     ;; ----------------------------------------------------------------
     cfengine
     emacs-lisp
     git
     github
     markdown
     org
     html
     spell-checking
     syntax-checking)
```

Additional Packages

The `ox-jira` package makes it easy to export into text marked up for Jira.

```
;; List of additional packages that will be installed without being
;; wrapped in a layer. If you need some configuration for these
;; packages, then consider creating a layer. You can also put the
;; configuration in `dotspacemacs/user-config'.
dotspacemacs-additional-packages '(ox-jira ox-reveal json-mode)
```

And that wraps up my `dotspacemacs/layers`

```
)
;; End dotspacemacs/layers
```

dotspacemacs/init

I don't really mess with much in here.

```
(defun dotspacemacs/init ()
  "Initialization function.
This function is called at the very startup of Spacemacs initialization
before layers configuration.
You should not put any user code in there besides modifying the variable
values."
  ;; This setq-default sexp is an exhaustive list of all the supported
  ;; spacemacs settings.
  (setq-default
```

Unchanged Spacemacs Defaults

These are spacemacs defaults that I left unchanged.

```
;; If non nil ELPA repositories are contacted via HTTPS whenever it's
;; possible. Set it to nil if you have no way to use HTTPS in your
;; environment, otherwise it is strongly recommended to let it set to t.
;; This variable has no effect if Emacs is launched with the parameter
;; `--insecure' which forces the value of this variable to nil.
;; (default t)
dotspacemacs-elpa-https t
;; Maximum allowed time in seconds to contact an ELPA repository.
dotspacemacs-elpa-timeout 5
;; If non nil then spacemacs will check for updates at startup
;; when the current branch is not `develop'. (default t)
dotspacemacs-check-for-update t
;; One of `vim', `emacs' or `hybrid'. Evil is always enabled but if the
;; variable is `emacs' then the `holy-mode' is enabled at startup. `hybrid'
;; uses emacs key bindings for vim's insert mode, but otherwise leaves evil
;; unchanged. (default 'vim)
dotspacemacs-editing-style 'vim
;; If non nil output loading progress in `*Messages*' buffer. (default nil)
dotspacemacs-verbose-loading nil
;; Specify the startup banner. Default value is `official', it displays
;; the official spacemacs logo. An integer value is the index of text
;; banner, `random' chooses a random text banner in `core/banners'
;; directory. A string value must be a path to an image format supported
;; by your Emacs build.
;; If the value is nil then no banner is displayed. (default 'official)
dotspacemacs-startup-banner 'official
;; List of items to show in the startup buffer. If nil it is disabled.
;; Possible values are: `recents' `bookmarks' `projects'.
;; (default '(recents projects))
dotspacemacs-startup-lists '(recents projects)
;; Number of recent files to show in the startup buffer. Ignored if
;; `dotspacemacs-startup-lists' doesn't include `recents'. (default 5)
dotspacemacs-startup-recent-list-size 5
;; Default major mode of the scratch buffer (default `text-mode')
dotspacemacs-scratch-mode 'text-mode
```

Editing style

I didn't change the default here, but **Emacs users** might want to change the editing style.

```
;; One of `vim', `emacs' or `hybrid'. Evil is always enabled but if the
;; variable is `emacs' then the `holy-mode' is enabled at startup. `hybrid'
;; uses emacs key bindings for vim's insert mode, but otherwise leaves evil
;; unchanged. (default 'vim)
dotspacemacs-editing-style 'vim
```

More Spacemacs defaults

These are some more spacemacs default values.

```
;; If non nil output loading progress in `*Messages*' buffer. (default nil)
dotspacemacs-verbose-loading nil
;; Specify the startup banner. Default value is `official', it displays
;; the official spacemacs logo. An integer value is the index of text
;; banner, `random' chooses a random text banner in `core/banners'
;; directory. A string value must be a path to an image format supported
;; by your Emacs build.
;; If the value is nil then no banner is displayed. (default 'official)
dotspacemacs-startup-banner 'official
;; List of items to show in the startup buffer. If nil it is disabled.
;; Possible values are: `recents' `bookmarks' `projects'.
;; (default '(recents projects))
dotspacemacs-startup-lists '(recents projects)
;; Number of recent files to show in the startup buffer. Ignored if
;; `dotspacemacs-startup-lists' doesn't include `recents'. (default 5)
dotspacemacs-startup-recent-list-size 5
;; Default major mode of the scratch buffer (default `text-mode')
dotspacemacs-scratch-mode 'text-mode
```

Theme

I do prefer the dark solarized theme.

```
;; List of themes, the first of the list is loaded when spacemacs starts.
;; Press <SPC> T n to cycle to the next theme in the list (works great
;; with 2 themes variants, one dark and one light)
dotspacemacs-themes '(
                     solarized-dark
                     spacemacs-dark
                     spacemacs-light
                     solarized-light
                     leuven
                     monokai
                     zenburn)
```

Even more default spacemacs values

```
;; If non nil the cursor color matches the state color in GUI Emacs.
dotspacemacs-colorize-cursor-according-to-state t
;; Default font. `powerline-scale' allows to quickly tweak the mode-line
;; size to make separators look not too crappy.
dotspacemacs-default-font '("Source Code Pro"
                            :size 13
                            :weight normal
                            :width normal
                            :powerline-scale 1.1)
;; The leader key
dotspacemacs-leader-key "SPC"
;; The leader key accessible in `emacs state' and `insert state'
;; (default "M-m")
dotspacemacs-emacs-leader-key "M-m"
;; Major mode leader key is a shortcut key which is the equivalent of
;; pressing `<leader> m`. Set it to `nil` to disable it. (default ",")
dotspacemacs-major-mode-leader-key ","
;; Major mode leader key accessible in `emacs state' and `insert state'.
;; (default "C-M-m)
dotspacemacs-major-mode-emacs-leader-key "C-M-m"
;; These variables control whether separate commands are bound in the GUI to
;; the key pairs C-i, TAB and C-m, RET.
;; Setting it to a non-nil value, allows for separate commands under <C-i>
;; and TAB or <C-m> and RET.
;; In the terminal, these pairs are generally indistinguishable, so this only
;; works in the GUI. (default nil)
dotspacemacs-distinguish-gui-tab nil
;; (Not implemented) dotspacemacs-distinguish-gui-ret nil
;; The command key used for Evil commands (ex-commands) and
;; Emacs commands (M-x).
;; By default the command key is `:' so ex-commands are executed like in Vim
;; with `:' and Emacs commands are executed with `<leader> :'.
dotspacemacs-command-key ":"
;; If non nil `Y' is remapped to `y$'. (default t)
dotspacemacs-remap-Y-to-y$ t
;; Name of the default layout (default "Default")
dotspacemacs-default-layout-name "Default"
;; If non nil the default layout name is displayed in the mode-line.
;; (default nil)
dotspacemacs-display-default-layout nil
;; If non nil then the last auto saved layouts are resume automatically upon
```

```
;; start. (default nil)
dotspacemacs-auto-resume-layouts nil
;; Location where to auto-save files. Possible values are `original' to
;; auto-save the file in-place, `cache' to auto-save the file to another
;; file stored in the cache directory and `nil' to disable auto-saving.
;; (default 'cache)
dotspacemacs-auto-save-file-location 'cache
;; Maximum number of rollback slots to keep in the cache. (default 5)
dotspacemacs-max-rollback-slots 5
;; If non nil then `ido' replaces `helm' for some commands. For now only
;; `find-files' (SPC f f), `find-spacemacs-file' (SPC f e s), and
;; `find-contrib-file' (SPC f e c) are replaced. (default nil)
dotspacemacs-use-ido nil
;; If non nil, `helm' will try to minimize the space it uses. (default nil)
dotspacemacs-helm-resize nil
;; if non nil, the helm header is hidden when there is only one source.
;; (default nil)
dotspacemacs-helm-no-header nil
;; define the position to display `helm', options are `bottom', `top',
;; `left', or `right'. (default 'bottom)
dotspacemacs-helm-position 'bottom
;; If non nil the paste micro-state is enabled. When enabled pressing `p`
;; several times cycle between the kill ring content. (default nil)
dotspacemacs-enable-paste-micro-state nil
;; Which-key delay in seconds. The which-key buffer is the popup listing
;; the commands bound to the current keystroke sequence. (default 0.4)
dotspacemacs-which-key-delay 0.4
;; Which-key frame position. Possible values are `right', `bottom' and
;; `right-then-bottom'. right-then-bottom tries to display the frame to the
;; right; if there is insufficient space it displays it at the bottom.
;; (default 'bottom)
dotspacemacs-which-key-position 'bottom
;; If non nil a progress bar is displayed when spacemacs is loading. This
;; may increase the boot time on some systems and emacs builds, set it to
;; nil to boost the loading time. (default t)
dotspacemacs-loading-progress-bar t
;; If non nil the frame is fullscreen when Emacs starts up. (default nil)
;; (Emacs 24.4+ only)
dotspacemacs-fullscreen-at-startup nil
;; If non nil `spacemacs/toggle-fullscreen' will not use native fullscreen.
;; Use to disable fullscreen animations in OSX. (default nil)
dotspacemacs-fullscreen-use-non-native nil
;; If non nil the frame is maximized when Emacs starts up.
;; Takes effect only if `dotspacemacs-fullscreen-at-startup' is nil.
;; (default nil) (Emacs 24.4+ only)
dotspacemacs-maximized-at-startup nil
```

```
;; A value from the range (0..100), in increasing opacity, which describes
;; the transparency level of a frame when it's active or selected.
;; Transparency can be toggled through `toggle-transparency'. (default 90)
dotspacemacs-active-transparency 90
;; A value from the range (0..100), in increasing opacity, which describes
;; the transparency level of a frame when it's inactive or deselected.
;; Transparency can be toggled through `toggle-transparency'. (default 90)
dotspacemacs-inactive-transparency 90
;; If non nil unicode symbols are displayed in the mode line. (default t)
dotspacemacs-mode-line-unicode-symbols t
;; If non nil smooth scrolling (native-scrolling) is enabled. Smooth
;; scrolling overrides the default behavior of Emacs which recenters the
;; point when it reaches the top or bottom of the screen. (default t)
dotspacemacs-smooth-scrolling t
;; If non nil line numbers are turned on in all `prog-mode' and `text-mode'
;; derivatives. If set to `relative', also turns on relative line numbers.
;; (default nil)
dotspacemacs-line-numbers nil
;; If non-nil smartparens-strict-mode will be enabled in programming modes.
;; (default nil)
dotspacemacs-smartparens-strict-mode nil
;; Select a scope to highlight delimiters. Possible values are `any',
;; `current', `all' or `nil'. Default is `all' (highlight any scope and
;; emphasis the current one). (default 'all)
dotspacemacs-highlight-delimiters 'all
;; If non nil advises quit functions to keep server open when quitting.
;; (default nil)
dotspacemacs-persistent-server nil
;; List of search tool executable names. Spacemacs uses the first installed
;; tool of the list. Supported tools are `ag', `pt', `ack' and `grep'.
;; (default '("ag" "pt" "ack" "grep"))
dotspacemacs-search-tools '("ag" "pt" "ack" "grep")
;; The default package repository used if no explicit repository has been
;; specified with an installed package.
;; Not used for now. (default nil)
dotspacemacs-default-package-repository nil
```

Whitespace Cleanup

I work with source code a lot and I like cleaning up the trailing whitespace automatically.

```
;; Delete whitespace while saving buffer. Possible values are `all'
;; to aggressively delete empty line and long sequences of whitespace,
;; `trailing' to delete only the whitespace at end of lines, `changed'to
;; delete only whitespace for changed lines or `nil' to disable cleanup.
;; (default nil)
dotspacemacs-whitespace-cleanup 'trailing
))
```

dotspacemacs/user-init

```
(defun dotspacemacs/user-init ()
  "Initialization function for user code.
It is called immediately after `dotspacemacs/init'.  You are free to put almost
any user code here.  The exception is org related code, which should be placed
in `dotspacemacs/user-config'."
  )
```

Function to add unique ID to each entry

I have this function so that I can add a unique id to each entry in the file. It can be obnoxious but my property drawers are folded so its not that intrusive, and it does allow for linking to a nodes unique id so that links can remain working after re-filing entries into different files. Sometimes I have a save hook configured to run this.

```
(defun my/org-add-ids-to-headlines-in-file ()
  "Add ID properties to all headlines in the current file which
   do not already have one."
  (interactive)
  (org-map-entries 'org-id-get-create))
```

And that wraps up my `dotspacemacs/user-init`.

```
)
;; End dotspacemacs/user-init
```

dotspacemacs/user-config

This is where I put the majority of my custom configuration.

```
(defun dotspacemacs/user-config ()
  "Configuration function for user code.
This function is called at the very end of Spacemacs initialization after
layers configuration. You are free to put any user code."
```

Pretty Bullets in org mode

I like these prettier bullets that I found in the org layer README.

You should be able to set any utf-8 characters. If your looking for some options you might try here.

```
;; Prettier Bullets for org-mode
(setq org-bullets-bullet-list '("■" "◆" "▲" "▶"))
```

Fast TODO selection

This enables fast todo selection to bring up the menu of TODO keywords when pressing `t` on a headline.

```
(setq org-use-fast-todo-selection t)
```

Configure TODO Keywords

This sets up quick access to `TODO` states. I definitely use `TODO` and `DONE` but I haven't really found my flow with any other states so this gets messed with from time to time.

```
;; TODO Keywords
(setq org-todo-keywords
      (quote ((sequence "TODO(t)" "IN_PROGRESS(i)" "|" "DONE(d)")
              (sequence "WAITING(w@/)" "HOLD(h@/)" "|" "CANCELLED(c@/)"))))
```

You might like to have some fancy utf characters in the TODO keywords, but it could complicate manual editing (which probably should be avoided anyway).

```
;; TODO Keywords
(setq org-todo-keywords
      (quote ((sequence "☛ TODO(t)" "IN_PROGRESS(i)" "|" "✔ DONE(d)")
              (sequence "⚑ WAITING(w@/)" "HOLD(h@/)" "|" "✘ CANCELLED(c@/)"))))
```

Enforce TODO dependencies

I have `TODO=s` scattered all over the place. This prevents me from `marking an entry as =DONE` when there are still sub entries that are not `DONE`.

```
;; Avoid setting entries as DONE when there are still sub-entries that are not
;; DONE.
(setq org-enforce-todo-dependencies t)
```

Capture Templates

I use the `To-do` capture all the time. All of the captured TODOs goto the same place and then I re-file them into other files later on. Note that I clock time during the capture and then automatically resume the previously clocked task when I am done capturing.

```
;; Configure custom capture templates
(setq org-capture-templates
      '(
        ("t" "To-do" entry (file+headline "~/org/main.org" "Tasks")
         "** TODO %^{Task Description}\nCreated From: %a\n" :clock-in t :clock-resume t :prepend t)))
```

Don't F-up my headline when pressing Alt-RETURN

You can use alt-return when your on a heading to create a new one below it. If your in the middle of the heading, it chops it off and that is really annoying.

Somewhere I stumbled on this setting in this guys private layer. There is some other good stuff in there too.

```
;; When hitting alt-return on a header, please create a new one without
;; messing up the one I'm standing on.
(setq org-insert-heading-respect-content t)
```

Don't let org clock time run amuck

I try to clock my time. I'm not great about doing it 100% of the time, but tracking my time is really useful. Clocktables can be really useful if your good about managing the clocked in task.

With this setting after 30 minutes of idle time I am prompted to see if I want to keep or throw away the idle time. I think this really helps my clocking accuracy.

```
;; Resolve open-clocks if idle more than 30 minutes
(setq org-clock-idle-time 30)
```

Hyperlink Abbreviations

I link to several issue tracking systems frequently, and link abbreviations make it much nicer.

```
;; Link abbreviations http://orgmode.org/manual/Link-abbreviations.html#Link-abbreviations
;; This makes it easy to create links in org files to common urls
;; Note: The actual link is not stored in the text, only when rendered
;; Usage: [[zendesk:2753]] or [[redmine:7481][My text]]
(setq org-link-abbrev-alist
      '(("zendesk" . "https://cfengine.zendesk.com/agent/tickets/")
        ("redmine" . "https://dev.cfengine.com/issues/")
        ("jira" . "https://tracker.mender.io/browse/")))
```

Rendering in-line images by default

I now do the majority of my writing in org-mode. Sometimes I include images. This makes them render in-line by default. I might switch this back off. It's nice but I have found it makes navigating documents more difficult when images are displayed inline.

```
; Enable automatic inline image rendering
; http://orgmode.org/manual/In_002dbuffer-settings.html
(setq org-startup-with-inline-images t)
```

Agenda Files

I consider myself quite unorganized. At least inside my head it feels that way. Org mode really helps me because it can search across multiple files. So I can have as many different files as I like for different projects or clients and then I can search across all of them for something using the agenda view. This specifies each of your org files, or the directories to be searched for org files.

For this presentation I just point to the directory that contains my git clone.

```
;; Set the org-agenda files
(setq org-agenda-files
  '("~/src/presentations/Level-up-your-notes-with-Org"))
```

Refile Targets

When I refile, I want to be able to go up to 5 levels deep. Probably I could take this down a notch or two. Usually I am only filing 1 or 2 levels deep.

```
;; Set refile locations to whats in org-agenda
(setq org-refile-targets (quote ((org-agenda-files :maxlevel . 5))))
```

Automatic Visual Indention

I picked this up from a co-worker. Before I started using this setting I was managing whitespace indentation inside my org files. Now its all indented automatically but the source itself is not indented.

```
;; Keep the indentation well structured by. OMG this is a must have. Makes
;; it feel less like editing a big text file and more like a purpose built
;; editor for org mode that forces the indentation.
(setq org-startup-indented t)
```

Zooming Text

Add these keys in to support familiar zooming of text with `Ctrl+` and `Ctrl-`.

```
;; Familiar zooming with Ctrl+ and Ctrl-
(define-key global-map (kbd "C-+") 'text-scale-increase)
(define-key global-map (kbd "C--") 'text-scale-decrease)
```

Custom Key bindings

Spacemacs reserves SPC o and SPC m o for user defined key bindings.

```
(spacemacs/set-leader-keys
 "oc" 'org-capture
 "oa" 'org-agenda)
```

Automatically generate unique IDs on save

Internal links can get broken when you refile entries. This hook makes sure that each capture gets a unique id. Using the ID property for linking can help to keep links functional when refiling.

```
  ;; This makes sure that each captured entry gets a unique ID
(add-hook 'org-capture-prepare-finalize-hook 'org-id-get-create)
```

Code block execution support

This is where you add support for specific languages in 'src' code blocks.

```
(org-babel-do-load-languages
'org-babel-load-languages
'(
  (sh . t)
  ))
```

Here I have a hook to automatically add unique IDs to each entry on save.

Auto save hooks

I used to have auto save configured as well for all agenda files, but I ran into some perform acne issues with some of my large (30k line) org files.

```
;; AUTOMATICLALLY CREATE IDS for all nodes in org mode file on save. This
;; helps when you use link to an entry and then it is moved to a different
;; file.
(add-hook 'org-mode-hook
          (lambda ()
            (add-hook 'before-save-hook 'my/org-add-ids-to-headlines-in-file nil 'local)))

;; auto save agenda I ran into some performance issues when this is combined
;; with adding ids to headlines. This seemed to be mostly related to some
;; large (30k line) org files.
;(add-hook 'org-agenda-mode-hook
;          (lambda ()
;            (add-hook 'auto-save-hook 'org-save-all-org-buffers nil t)
;            (auto-save-mode)))
```

Keeping Secrets Encrypted

Some things in my notes need to be kept secret. This allows me to tag entries with `crypt` to have them encrypted with my gpg key.

```
;; http://yenliangl.blogspot.com/2009/12/encrypt-your-important-data-in-emacs.html
;; http://emacs-fu.blogspot.com/2011/02/keeping-your-secrets-secret.html
;; This allows me to encrypt subtrees that are tagged with crypt automatically.
;; by default I want to encrypt it to myself. but with properties entries I cna encrypt to other people. which is useful
(require 'org-crypt)

;; Automatically re-encrypt entries on save to avoid leaking decrypted
;; information.
(org-crypt-use-before-save-magic)
(setq org-crypt-disable-auto-save (quote encrypt))

;; GPG key to use for encryption
;; Either the Key ID or set to nil to use symmetric encryption.
(setq org-crypt-key "5D1CCC11")

;; This prevents the crypt tag from being included in inheritance.
(setq org-tags-exclude-from-inheritance (quote ("crypt")))
```

Initalize other packages

And finally require some of the additional packages that I use.

```
;; Export to jira markup for more beautiful issue comments
(require 'ox-jira)
;; Export to markdown for more beautiful comments in zendesk
(require 'ox-md)
;; So that I can make presentations in org and display them with the beautiful
;; reveal.js
(require 'ox-reveal)
```

And finally, we reach the end.

```
)
;; End dotspacemacs/user-config
```

Created by Nick Anderson.