Spacemacs documentation

Marvin Johanning

October 27, 2017

Contents

1 Core Pillars

Four core pillars: Mnemonic, Discoverable, Consistent and "Crowd-Configured".

If any of these core pillars is violated open an issue and we'll try our best to fix it.

1.1 Mnemonic

Key bindings are organized using mnemonic prefixes like `b` for buffer, `p` for project, `s` for search, `h` for help, etc...

1.2 Discoverable

Innovative real-time display of available key bindings. Simple query system to quickly find available layers, packages, and more.

1.3 Consistent

Similar functionalities have the same key binding everywhere thanks to a clearly defined set of conventions. Documentation is mandatory for any layer that ships with Spacemacs.

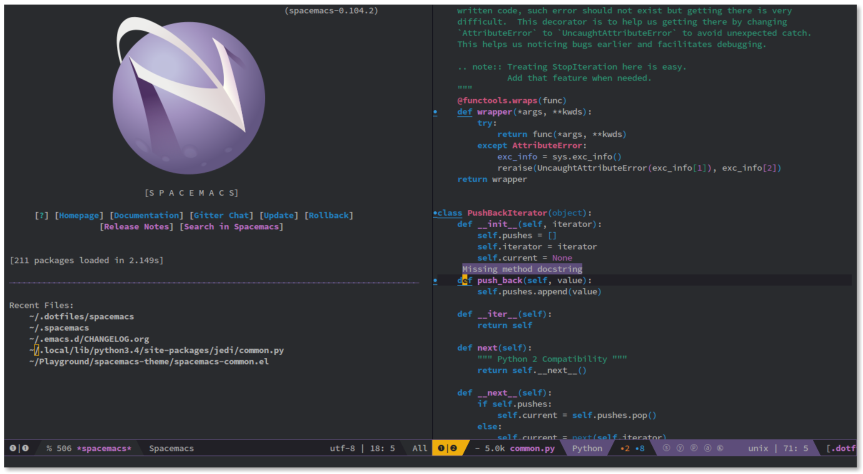
1.4 Crowd-Configured

Community-driven configuration provides curated packages tuned by power users and bugs are fixed quickly.

2 Highlighted feature

- **Bring the efficiency of modal editing** to the powerful Emacs lisp platform. Modal UX is optional and Spacemacs can be used with only Emacs key bindings.
- Integrate nicely with `Evil` states (`Vim` modes).
- **Keep your fingers on the home row** for quicker editing with support for QWERTY and BEPO layouts.
- **Minimalistic and nice graphical UI** keeps your available screen space for what matters: your text files.
- **Fast boot time**: packages and configuration are lazy-loaded as much as possible.
- **Lower the risk of RSI** by heavily using the space bar instead of modifiers. If you have issues with your thumbs you can still use Spacemacs using modifiers.
- Contribute easily your improvements and new configuration layers.
- **Very active and helpful community** on `Gitter` and `IRC` (via Gitter IRC bridge)

3 Screenshots



```
(spacemacs-0.104.2)

[S P A C E M A C S]

[?] [Homepage] [Documentation] [Gitter Chat] [Update] [Rollback]
        [Release Notes] [Search in Spacemacs]

[211 packages loaded in 2.149s]

Recent Files:
    ~/.dotfiles/spacemacs
    ~/.spacemacs
    ~/.emacs.d/CHANGELOG.org
    ~/.local/lib/python3.4/site-packages/jedi/common.py
    ~/Playground/spacemacs-theme/spacemacs-common.el
```

```
    written code, such error should not exist but getting there is very
    difficult.  This decorator is to help us getting there by changing
    `AttributeError` to `UncaughtAttributeError` to avoid unexpected catch.
    This helps us noticing bugs earlier and facilitates debugging.

    .. note:: Treating StopIteration here is easy.
              Add that feature when needed.
    """
    @functools.wraps(func)
    def wrapper(*args, **kwds):
        try:
            return func(*args, **kwds)
        except AttributeError:
            exc_info = sys.exc_info()
            reraise(UncaughtAttributeError(exc_info[1]), exc_info[2])
    return wrapper


class PushBackIterator(object):
    def __init__(self, iterator):
        self.pushes = []
        self.iterator = iterator
        self.current = None
     Missing method docstring
    def push_back(self, value):
        self.pushes.append(value)

    def __iter__(self):
        return self

    def next(self):
        """ Python 2 Compatibility """
        return self.__next__()

    def __next__(self):
        if self.pushes:
            self.current = self.pushes.pop()
        else:
            self.current = next(self.iterator)
```

```
% 506 *spacemacs*   Spacemacs   utf-8 | 18: 5   All   - 5.0k common.py   Python   •2 •8   unix | 71: 5   [.dotf
```

Python

Note: Even though screenshots are updated frequently, Spacemacs is evolving quickly and the screenshots may not reflect exactly the current state of the project.

4 Who can benefit from this?

- Spacemacs was initially intended to be used by **Vim users** who want to go to the next level by using Emacs (see guide for Vimmers). But it is now perfectly **usable by non Vim users** by choosing the `emacs` editing style.
- It is also a good fit for people wanting to **lower the risk of RSI** induced by the default Emacs key bindings. (This is an assumption, there are no official studies to prove this!) If you have issues using your thumbs you can still use the `emacs` editing style which puts the leader key on a modifier combination.
- Emacs users wanting to learn **a different way to edit files** or wanting to learn Vim key bindings or even wanting to mix both editing styles by setting their style to `hybrid`.
- Emacs users wanting a simple but deep configuration system that greatly **lower the risk of .emacs bankruptcy**.

- **Pair-programming** users thanks to out of the box support for dynamic switching of editing style. A Vim user and an Emacs user can use the same Spacemacs comfortably.

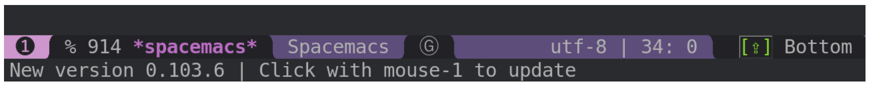
5 Update and Rollback

5.1 Update Spacemacs repository

There are several methods of updating the core files and layer information for Spacemacs. It is recommended to update the packages first; see the next section.

5.1.1 Automatic Updates

Spacemacs will automatically check for a new version every startup. When it detects that a new version is available an arrow will appear in the modeline. Click it to update Spacemacs. You must restart Emacs after updating.



```
① % 914 *spacemacs*  Spacemacs Ⓖ          utf-8 | 34: 0    [⇪] Bottom
New version 0.103.6 | Click with mouse-1 to update
```

Update Button

Note: If you use the `develop` branch of Spacemacs, automatic update is disabled—you have to update manually using git.

5.1.2 Updating from the Spacemacs Buffer

Use the button labeled "Update Spacemacs" in the Spacemacs buffer. You will be prompted for the version you would like to use.

Note: If you use the `develop` branch of Spacemacs, you cannot use this method.

5.1.3 Updating Manually with git

To update manually close Emacs and update the git repository:

```
$ git pull origin master
```

Note: The master branch is considered to be immutable in the sense that you must not modify it by adding your own commit. If you do so you will break the automatic update of Spacemacs on the master branch. To fork Spacemacs code you have to use a custom branch that you manage manually.

5.2 Update packages

To update the Emacs packages used by Spacemacs press RET (enter) or click on the link `[Update Packages]` in the startup page under the banner then restart Emacs. If you prefer, you can use the command `configuration-layer/update-packages` instead of the button.

If anything goes wrong you should be able to rollback the update by pressing `RET` or clicking on the `[Rollback Package Update]` link in the startup page and choosing a rollback slot (sorted by date). This button uses the command `configuration-layer/rollback`.

6 Configuration layers

This section is an overview of layers. A more extensive introduction to writing configuration layers can be found here (recommended reading!).

6.1 Purpose

Layers help collect related packages together to provide features. For example, the `python` layer provides auto-completion, syntax checking, and REPL support for python files. This approach helps keep configuration organized and reduces overhead for the user by keeping them from having to think about what packages to install. To install all the `python` features the user has just to add the `python` layer to their dotfile.

6.2 Structure

Configuration is organized in layers. Each layer has the following structure:

```
[layer_name]
  |__ [local]
  | |__ [package 1]
  | |      ...
  | |__ [package n]
  |-- layers.el
  |__ packages.el
  |__ funcs.el
  |__ config.el
  |__ keybindings.el

[] = directory
```

Where:

File	Usage
layers.el	The place to declare additional layers
packages.el	The list of packages and their configuration functions (init, post-init, etc…)
funcs.el	All functions defined in the layer (used in package configuration for instance)
config.el	Layer configuration (defines the layer variables default values and setup some config
keybindings.el	General key bindings no tied to a specific package configuration

`Packages` can be:

- `ELPA` packages installed from an `ELPA` compliant repository
- local packages in a layer's `local` folder
- installed from an online source using quelpa.

6.3 Configure packages

6.3.1 With a layer

1. Declaration `Packages` are declared in a variable called `<layer>-packages` where `<layer>` is the name of the layer.

   Example:

   ```
   (setq <layer>-packages '(package1 package2 ...)
   ```

   All packages from all layers are processed in alphabetical order so sometimes you'll have to use some `with-eval-after-load` black magic to configure them properly. For instance if package `A` depends on `B` then you can configure `A` with:

   ```
   (with-eval-after-load 'B ...)
   ```

   For details on installing packages using quelpa or local packages see LAYERS.

2. Initialization To initialize a package `xxx`, define a function with this format in `packages.el`:

   ```
   (defun <layer>/init-xxx () ...body )
   ```

It is common to define the body with the use-package macro.

3. Exclusion It is possible to exclude some packages from Spacemacs on a per-layer basis. This is useful when a configuration layer aims to replace a stock package declared in the Spacemacs layer.

   To do so add the package names to exclude to the variable `<layer>-excluded-packages`.

   Example:

   ```
   (setq <layer>-excluded-packages '(package1 package2 ...)
   ```

6.3.2 Without a layer

Sometimes a layer can be an unnecessary overhead, this is the case if you just want to install a package with very few configuration associated to it. A good example is some niche language where you are only interested in syntax highlighting.

You can install such packages by adding them to the variable `dotspacemacs-additional-packages` under the `dotspacemacs/layers` function in your dotfile.

For example, to install `llvm-mode` and `dts-mode`:

```
(defun dotspacemacs/layers ()
  "Configuration Layers declaration..."
  (setq-default
   ;; ...
   dotspacemacs-additional-packages '(llvm-mode dts-mode)
   ;; ...
   ))
```

If you want to add some configuration for them then put the configuration in the `dotspacemacs/user-config` function or consider creating a layer.

6.4 Packages synchronization

Spacemacs will only install the packages that are explicitly used by the user. A package is considered to be used if its layer is used (i.e. listed in `dotspacemacs-configuration-layers`). Any packages that are not used is considered to be orphan and is deleted at the next startup of Emacs.

6.5 Types of configuration layers

There are two types of configuration layers:

- distributed layers (in the `layers` directory, those layers are contributions shared by the community and merged upstream)
- private (in the `private` directory, they are ignored by Git)

6.6 Submitting a configuration layer upstream

If you decide to provide a configuration layer, please check the contribution guidelines first in CONTRIBUTING.

6.7 Example: Themes Megapack example

This is a simple configuration layer listing a bunch of themes which you can find here.

To install it, just add `themes-megapack` to your `~/.spacemacs` like so:

```
(setq-default dotspacemacs-configuration-layers '(themes-megapack))
```

Adding this layer will install around 100 themes; to uninstall them remove the layer from the `dotspacemacs-configuration-layers` and press `SPC f e R`.

6.8 Managing private configuration layers

Spacemacs's configuration system is flexible enough to let you manage your private layers in different ways.

6.8.1 Using the private directory

Everything in the private directory is ignored by Git so it is a good place to store private layers. There is a huge drawback to this approach though: *your layers are not source controlled.*

6.8.2 Using an external Git repository

This is the recommended way to manage your private layers.

The best approach is to store all your private layers into an external Git repository. It is especially a good practice to store them in your `dotfiles` repository if you have one. Put also your `~/.spacemacs` file in it.

Then you are free to symlink your layers into `~/emacs.d/private` *or* let them anywhere you want and reference the parent directory in the variable `dotspacemacs-configuration-layer-path` of your `~/.spacemacs`.

Note that you could also have a dedicated repository for all your private layers and then directly clone this repository in `~/.emacs.d/private`.

6.8.3 Using a personal branch

The final main way to manage your private layers is to push them in a personal branch that you keep up to date with upstream `master` or `develop`.

6.9 Tips for writing layers

Please refer to this introduction for some tips on writing layers, and how to best make them fit with the Spacemacs philosophy and loading strategy.

7 Dotfile Configuration

User configuration can be stored in your `~/.spacemacs` file.

7.1 Dotfile Installation

The very first time Spacemacs starts up, it will ask you several questions and then install the `.spacemacs` in your `HOME` directory.

7.2 Alternative dotdirectory

A dotdirectory `~/.spacemacs.d/` can be used instead of a dotfile. If you want to use this option, move `~/.spacemacs` to `~/.spacemacs.d/init.el`.

It is also possible to override the location of `~/.spacemacs.d/` using the environment variable `SPACEMACSDIR`. Of course you can also use symlinks to change the location of this directory.

Note: `~/.spacemacs` will always take priority over `~/.spacemacs.d/init.el`, so `~/.spacemacs` must not exist for `~/.spacemacs.d/init.el` to be used by Spacemacs.

7.3 Synchronization of dotfile changes

To apply the modifications made in `~/.spacemacs` press `SPC f e R`. It will re-execute the Spacemacs initialization process.

Note: A synchronization re-executes the functions `dotspacemacs/init`, `dotspacemacs/user-init` and `dotspacemacs/user-config`. Depending on the content of this functions you may encounter some unwanted side effects. For instance if you use a toggle in `dotspacemac/user-config` to enable some behavior, this behavior will be turned off whenever the dotfile is re-synchronized. To avoid these side-effects it is recommended to either use `setq` expressions instead of toggle functions, or to use the `on` or `off` versions instead (i.e. instead of `spacemacs/toggle-<thing>`, use `spacemacs/toggle-<thing>-on` or `spacemacs/toggle-<thing>-off`).

It is possible to *skip* the execution of `dotspacemacs/user-config` with the universal argument (`SPC u SPC f e R`).

7.4 Testing the dotfile

You can use the command `SPC SPC dotspacemacs/test-dotfile` to check if your `~/.spacemacs` looks correct. This will check, among other things, whether the declared layers can be found and that the variables have sensible values. These tests are also run automatically when you synchronize with `SPC f e R`.

7.5 Dotfile Contents

7.5.1 Configuration functions

Three special functions in the `~/.spacemacs` file can be used to perform configuration at the beginning and end of Spacemacs loading process:

- `dotspacemacs/layers` is called at the very startup of Spacemacs initilialization, this is where you set the Spacemacs distribution and declare layers to be used in your configuration. You can also add or excluded packages of your choice and tweak some behavior of Spacemacs loading.

- `dotspacemacs/init` is called at the very startup of Spacemacs initialization before layers configuration. **You should not put any user code** in there besides modifying the Spacemacs variable values prefixed with `dotspacemacs-`.

- `dotspacemacs/user-init` is called immediately after `dotspacemacs/init`, before layer configuration. This function is mostly useful for variables that need to be set before packages are loaded.

- `dotspacemacs/user-config` is called at the very end of Spacemacs initialization after layers configuration. This is the place where most of your configurations should be done. Unless it is explicitly specified that a variable should be set before a package is loaded, you should place your code here.

7.5.2 Custom variables

Custom variables configuration from `M-x customize-group` built-in Emacs feature are automatically saved by Emacs at the end of your `~/.spacemacs` file.

7.6 Declaring Configuration layers

To use a configuration layer, declare it in your dotfile by adding it to the `dotspacemacs-configuration-layers` variable of your `~/.spacemacs`.

Note: In this documentation a `used layer` is equivalent to a `declared layer`.

For instance, RMS can add his private configuration layer like this:

```
(setq-default dotspacemacs-configuration-layers
  '(
    ;; other layers
    ;; rms layer added at the end of the list
    rms
  ))
```

Official layers shipped with Spacemacs are stored in `~/.emacs.d/layers`. The directory `~/.emacs.d/private` is a drop-in location for your private layers. It is possible to put layers at the location of your choice provided you tell Spacemacs where to look for them. This is done by setting the list `dotspacemacs-configuration-layer-path` in your `~/.spacemacs`. For instance to add some layers in `~/.myconfig`, set the variable like this:

```
(setq-default dotspacemacs-configuration-layer-path '("~/.myconfig/"))
```

7.6.1 Setting configuration layers variables

Some configuration layers have configuration variables to enable specific feature. For instance the git layer has several configuration variables, they can be set directly in the `dotspacemacs-configuration-layers` like this:

```
(defun dotspacemacs/layers ()
  ;; List of configuration layers to load.
  (setq-default dotspacemacs-configuration-layers
    '(auto-completion
      (git :variables
           git-magit-status-fullscreen t
           git-variable-example nil)
      smex)))
```

The `:variables` keyword is a convenience to keep layer configuration close to their declaration. Setting layer variables in the `dotspacemacs/user-init` function of your dotfile is also a perfectly valid way to configure a layer.

7.6.2 Disabling layer services in other layers

Often layers enable services that other layers can use. For instance if you use the layer `auto-completion` then every other layers supporting `auto-completion` will have this feature enabled.

Sometimes you may want to disable a service added by a layer in some specific layers. Say you want to disable `auto-completion` in `org` and `git` layers, you can do it with the following layer declaration.

```
(defun dotspacemacs/layers ()
  ;; List of configuration layers to load.
  (setq-default dotspacemacs-configuration-layers
    '(org git
      (auto-completion :disabled-for org git))))
```

You can also use the `:enabled-for` construct to disable it for *all* layers *except* those explicitly identified.

```
(defun dotspacemacs/layers ()
  ;; List of configuration layers to load.
  (setq-default dotspacemacs-configuration-layers
    '(java python c-c++
      (auto-completion :enabled-for java python))))
```

Note that `:enabled-for` may be an empty list.

```
(defun dotspacemacs/layers ()
  ;; List of configuration layers to load.
  (setq-default dotspacemacs-configuration-layers
    '(java python c-c++
      (auto-completion :enabled-for))))
```

:enabled-for takes precedence over :disabled-for if both are present.

7.6.3 Selecting/Ignoring packages of a layer

By default a declared layer installs/configures all its associated packages. You may want to select only some of them or ignoring some of them. This is possible with the :packages keyword.

For instance to ignore the neotree and fancy-battery packages from spacemacs-ui-visual layer:

```
(defun dotspacemacs/layers ()
  ;; List of configuration layers to load.
  (setq-default dotspacemacs-configuration-layers
    '(auto-completion
      (spacemacs-ui-visual :packages (not neotree fancy-battery))))
```

The opposite would be to ignore all packages except neotree and fancy-battery:

```
(defun dotspacemacs/layers ()
  ;; List of configuration layers to load.
  (setq-default dotspacemacs-configuration-layers
    '(auto-completion
      (spacemacs-ui-visual :packages neotree fancy-battery)))
```

Note: Ignoring a package from a layer is different than excluding a package. An excluded packages is completely removed from your configuration whereas an ignored package is ignored only for a given layer but it can remain on your system. It happens that if the given layer is the owner of the package then ignoring this package is the same as excluding it (because the package becomes orphan so it is considered unused by Spacemacs).

7.6.4 Excluding packages

You can exclude packages you don't want to install with the variable dotspacemacs-excluded-packages (see Configuration layers for more info on packages).

For instance, to disable the rainbow-delimiters package:

```
(setq-default dotspacemacs-excluded-packages '(rainbow-delimiters))
```

When you exclude a package, Spacemacs will automatically delete it for you the next time you launch Emacs or at the next dotfile synchronization. All the orphan dependencies are also deleted automatically. Excluding a package effectively remove all references to it in Spacemacs without breaking the rest of the configuration, this is a powerful feature which allows you to quickly remove any feature from Spacemacs.

Note: A few packages are essential for Spacemacs to correctly operate, those packages are protected and cannot be excluded or uninstalled even if they become orphans or are excluded. `use-package` is an example of a protected package that cannot be removed from Spacemacs.

8 Concepts

8.1 Editing Styles

Spacemacs comes with several editing styles which can be switched dynamically providing an easier way to do pair programming, for instance between a Vim user and an Emacs user.

Three styles are available:

- Vim,
- Emacs,
- Hybrid (a mix between Vim and Emacs).

8.1.1 Vim

Spacemacs behaves like in Vim using Evil mode package to emulate Vim key bindings. This is the default style of Spacemacs; it can be set explicitly by setting the `dotspacemacs-editing-style` variable to `vim` in the dotfile.

To bind keys in Vim editing style (`insert state`):

```
(define-key evil-insert-state-map (kbd "C-]") 'forward-char)
```

8.1.2 Emacs

Spacemacs behaves like in raw Emacs using the Holy mode which configures Evil to make the `emacs state` the default state everywhere. Set the `dotspacemacs-editing-style` variable to `emacs` in the dotfile.

In Emacs style the leader is available on `M-m`. It is possible to toggle it on and off with `SPC t E e` and `M-m t E e`. When off the `vim` style is enabled.

To bind keys in Emacs editing style (`emacs state`):

```
(define-key evil-emacs-state-map (kbd "C-]") 'forward-char)
```

8.1.3 Hybrid

The hybrid editing style is like the Vim style except that `insert state` is replaced by a new state called `hybrid state`. In `hybrid state` all the Emacs key bindings are available; this is like replacing the `insert state` with the `emacs state` but provides an isolated key map `evil-hybrid-state-map`.

To bind keys in Hybrid editing style (`hybrid state`):

```
(define-key evil-hybrid-state-map (kbd "C-]") 'forward-char)
```

This style can be tweaked to be more like Emacs or more like Vim depending on the user preferences. The following variables are available to change the style configuration:

- `hybrid-mode-default-state` The default state when opening a new buffer, default is `normal`. Set it to `emacs` for a more emacsy style.
- `hybrid-mode-enable-hjkl-bindings` If non nil then packages will configure `h j k l` key bindings for navigation.
- `hybrid-mode-enable-evilified-state` If non nil buffer are `evilified` when supported, if nil then `emacs` state is enabled in those buffers instead.

Default configuration is:

```
(setq-default dotspacemacs-editing-style '(hybrid :variables
                                            hybrid-mode-enable-evilified-state t
                                            hybrid-mode-enable-hjkl-bindings nil
                                            hybrid-mode-default-state 'normal)
```

To toggle the hybrid style on and off use `SPC t E h` and `M-m t E h`. When off the `vim` style is enabled.

8.2 States

Spacemacs has 10 states:

State	Default Color	Description
normal	orange	like the `normal mode of Vim`, used to execute and combine commands
insert	green	like the `insert mode of Vim`, used to actually insert text
visual	gray	like the `visual mode of Vim`, used to make text selection
motion	purple	exclusive to `Evil`, used to navigate read only buffers
emacs	blue	exclusive to `Evil`, using this state is like using a regular Emacs without
replace	chocolate	exclusive to `Evil`, overwrites the character under point instead of insert
hybrid	blue	exclusive to Spacemacs, this is like the insert state except that all the e
evilified	light brown	exclusive to Spacemacs, this is an `emacs state` modified to bring Vim n
lisp	pink	exclusive to Spacemacs, used to navigate Lisp code and modify it (more
iedit	red	exclusive to Spacemacs, used to navigate between multiple regions of te
iedit-insert	red	exclusive to Spacemacs, used to replace multiple regions of text using `i`

Note: Technically speaking there is also the `operator` evil state.

8.3 Evilified modes

Some buffers are not for editing text and provide their own keybindings for certain operations. These often conflict with Vim bindings. To make such buffers behave more like Vim in a consistent manner, they use a special state called *evilified* state. In evilified state, a handful of keys work as in Evil, namely `/`, `:`, `h`, `j`, `k`, `l`, `n`, `N`, `v`, `V`, `gg`, `G`, `C-f`, `C-b`, `C-d`, `C-e`, `C-u`, `C-y` and `C-z`. All other keys work as intended by the underlying mode.

Shadowed keys are moved according to the pattern: `a` → `A` → `C-a` → `C-A`

For example, if the mode binds a function to `n`, that is found under `C-n` in evilified state, since both `n` and `N` are reserved, but `C-n` is not. On the other hand, anything originally bound to `k` will be found on `K`, since `k` is reserved but `K` is not. If there is a binding on `K`, that will be moved to `C-k`.

In addition to this, `C-g`, being an important escape key in Emacs, is skipped. So anything bound to `g` originally will be found on `C-G`, since `g`, `G` and `C-g` are all reserved.

8.4 Evil leader

Spacemacs uses a leader key to bind almost all its key bindings.

This leader key is commonly set to `,` by Vim users, in Spacemacs the leader key is set on `SPC` (the space bar, hence the name `spacemacs`). This key is the most accessible key on a keyboard and it is pressed with the thumb which is a good choice to lower the risk of RSI. It can be customized to any other key using the variable `dotspacemacs-leader-key` and `dotspacemacs-emacs-leader-key`.

With Spacemacs there is no need to remap your keyboard modifiers to attempt to reduce the risk of RSI, every command can be executed very easily while you are in `normal` mode by pressing the `SPC` leader key, here are a few examples:

- Save a buffer: `SPC f s`
- Save all opened buffers: `SPC f S`
- Open (switch) to a buffer with `helm`: `SPC b b`

8.5 Universal argument

The universal argument `C-u` is an important command in Emacs but it is also a very handy Vim key binding to scroll up.

Spacemacs binds `C-u` to `scroll-up` and changes the universal argument binding to `SPC u`.

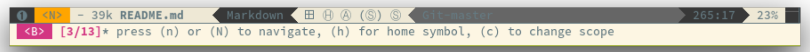
Note: `SPC u` is not working before `helm-M-x` (`SPC SPC`). Instead, call `helm-M-x` first, select the command you want to run, and press `C-u` before pressing `RETURN`. For instance: `SPC SPC org-reload C-u RET`

8.6 Transient-states

Spacemacs defines a wide variety of `transient states` (temporary overlay maps) where it makes sense. This prevents one from doing repetitive and tedious presses on the `SPC` key.

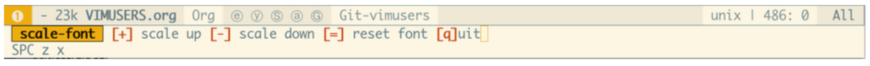
When a `transient state` is active, a documentation is displayed in the minibuffer. Additional information may as well be displayed in the minibuffer.

Auto-highlight-symbol transient state:



```
1 <N> - 39k README.md  Markdown  Git-master  265:17  23%
<B> [3/13]* press (n) or (N) to navigate, (h) for home symbol, (c) to change scope
```

Text scale transient state:



```
1 - 23k VIMUSERS.org  Org  Git-vimusers  unix | 486: 0  All
scale-font [+] scale up [-] scale down [=] reset font [q]uit
SPC z x
```

9 Differences between Vim, Evil and Spacemacs

- The `,` key does "repeat last `f`, `t`, `F`, or `T` command in opposite direction in `Vim`, but in Spacemacs it is the major mode specific leader key by default (which can be set on another key binding in the dotfile).

Send a PR to add the differences you found in this section.

9.1 The vim-surround case

There is one obvious visible difference though. It is not between `Evil` and `Vim` but between Spacemacs and vim-surround: in visual mode the `surround` command is on `S` in `vim-surround` whereas it is on `s` in Spacemacs.

This is something that can surprise some Vim users so here are some motivations behind this change:

- `s` and `c` do the same thing in `visual state`,
- `s` is only useful to delete *one* character and add more than one character which is a *very* narrow use case
- `c` accept motions and can do everything `s` can do in `normal state` (note that this is also true for `r` but `r` is more useful because it stays in `normal state`)
- `surround` command is just a more powerful command than `s`.

If you are not convinced, then here is the snippet to revert back to the default `Vim + vim-surround` setup (add it to your `dotspacemacs/user-config` function or your `~/.spacemacs`):

```
(evil-define-key 'visual evil-surround-mode-map "s" 'evil-substitute)
(evil-define-key 'visual evil-surround-mode-map "S" 'evil-surround-region)
```

10 Evil plugins

Spacemacs ships with the following evil plugins:

Mode	Description
evil-args	motions and text objects for arguments
evil-exchange	port of vim-exchange
evil-indent-textobject	add text object based on indentation level
evil-matchit	port of matchit.vim
evil-nerd-commenter	port of nerdcommenter
evil-numbers	like `C-a` and `C-x` in vim
evil-search-highlight-persist	emulation of hlsearch behavior
evil-surround	port of vim-surround
evil-visualstar	search for current selection with `*`
NeoTree	mimic NERD Tree

11 Binding keys

Key sequences are bound to commands in Emacs in various keymaps. The most basic map is the `global-map`. Setting a key binding in the `global-map` is achieved with the function `global-set-key`. Example to bind a key to the command `forward-char`:

```
(global-set-key (kbd "C-]") 'forward-char)
```

The `kbd` macro accepts a string describing a key sequence. The `global-map` is often shadowed by other maps. For example, `evil-mode` defines keymaps that target states (or modes in vim terminology). Here is an example that creates the same binding as above but only in `insert state` (`define-key` is a built-in function. `Evil-mode` has its own functions for defining keys).

```
(define-key evil-insert-state-map (kbd "C-]") 'forward-char)
```

Perhaps most importantly for Spacemacs is the use of the bind-map package to bind keys behind a leader key. This is where most of the Spacemacs bindings live. Binding keys behind the leader key is achieved with the functions `spacemacs/set-leader-keys` and `spacemacs/set-leader-keys-for-major-mode`, example:

```
(spacemacs/set-leader-keys "C-]" 'forward-char)
(spacemacs/set-leader-keys-for-major-mode 'emacs-lisp-mode "C-]" 'forward-char)
```

These functions use a macro like `kbd` to translate the key sequences for you. The second function, `spacemacs/set-leader-keys-for-major-mode`, binds the key only in the specified mode. The second key binding is active only when the major mode is `emacs-lisp`.

Finally, one should be aware of prefix keys. Essentially, all keymaps can be nested. Nested keymaps are used extensively in spacemacs, and in vanilla Emacs for that matter. For example, `SPC a` points to key bindings for "applications", like `SPC a c` for `calc-dispatch`. Nesting bindings is easy.

```
(spacemacs/declare-prefix "]" "bracket-prefix")
(spacemacs/set-leader-keys "]]" 'double-bracket-command)
```

The first line declares `SPC ]` to be a prefix and the second binds the key sequence `SPC ]]` to the corresponding command. The first line is actually unnecessary to create the prefix, but it will give your new prefix a name that key-discovery tools can use (e.g., which-key).

There is much more to say about bindings keys, but these are the basics. Keys can be bound in your `~/.spacemacs` file or in individual layers.

12 GUI Elements

Spacemacs has a minimalistic and distraction free graphical UI:

- custom powerline mode-line with color feedback according to current Flycheck status
- Unicode symbols for minor mode lighters which appear in the mode-line
- custom fringe bitmaps and error feedbacks for Flycheck

12.1 Color themes

The official Spacemacs theme is spacemacs-dark and it is the default theme installed when you first started Spacemacs. There are two variants of the theme, a dark one and a light one. Some aspects of these themes can be customized in the function `dotspacemacs/user-init` of your `~/.spacemacs`:

- the comment background with the boolean `spacemacs-theme-comment-bg`
- the height of org section titles with `spacemacs-theme-org-height`

It is possible to define your default themes in your `~/.spacemacs` with the variable `dotspacemacs-themes`. For instance, to specify `spacemacs-light`, `leuven` and `zenburn`:

```
(setq-default dotspacemacs-themes '(spacemacs-light leuven zenburn))
```

Key Binding	Description
`SPC T n`	switch to next theme listed in `dotspacemacs-themes`.
`SPC T s`	select a theme using a `helm` buffer.

You can see samples of all included themes in this theme gallery from Rob Merrell.

Note:

- You don't need to explicitly list in a layer the theme packages you are defining in `dotspacemacs-themes`, Spacemacs is smart enough to remove those packages from the list of orphans.
- Due to the inner working of themes in Emacs, switching theme during the same session may have some weird side effects. Although these side effects should be pretty rare.
- In the terminal version of Emacs, color themes will not render correctly as colors are rendered by the terminal and not by emacs. You will probably have to change your terminal color palette. More explanations can be found on emacs-color-theme-solarized webpage.

Hint: If you are an `Org` user, leuven-theme is amazing ;-)

12.2 Font

The default font used by Spacemacs is Source Code Pro by Adobe. It is recommended to install it on your system if you wish to use it.

To change the default font set the variable `dotspacemacs-default-font` in your `.spacemacs` file. By default its value is:

```
(setq-default dotspacemacs-default-font '("Source Code Pro"
                                           :size 13
                                           :weight normal
                                           :width normal
                                           :powerline-scale 1.1))
```

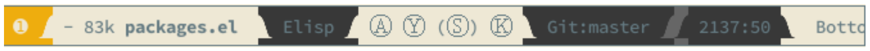
If the specified font is not found, the fallback one will be used (depends on your system). Also note that changing this value has no effect if you are running Emacs in terminal.

The properties should be pretty straightforward, it is possible to set any valid property of a font-spec:

- `:family` Font family or fontset (a string).
- `:width` Relative character width. This should be one of the symbols:
  - ultra-condensed
  - extra-condensed
  - condensed
  - semi-condensed
  - normal
  - semi-expanded
  - expanded
  - extra-expanded
  - ultra-expanded
- `:height` The height of the font. In the simplest case, this is an integer in units of 1/10 point.
- `:weight` Font weight- one of the symbols (from densest to faintest):
  - ultra-bold
  - extra-bold
  - bold
  - semi-bold
  - normal
  - semi-light
  - light
  - extra-light
  - ultra-light
- `:slant` Font slant- one of the symbols:
  - italic
  - oblique
  - normal
  - reverse-italic
  - reverse-oblique

- `:size` The font size- either a non-negative integer that specifies the pixel size, or a floating-point number that specifies the point size.
- `:adstyle` Additional typographic style information for the font, such as 'sans'. The value should be a string or a symbol.
- `:registry` The charset registry and encoding of the font, such as 'iso8859-1'. The value should be a string or a symbol.
- `:script` The script that the font must support (a symbol).

The special property `:powerline-scale` is Spacemacs specific and it is for quick tweaking of the mode-line height in order to avoid crappy rendering of the separators like on the following screenshot (default value is 1.1).



Ugly separators

12.3 GUI Toggles

Some graphical UI indicators can be toggled on and off (toggles start with `t` and `T`):

Key Binding	Description
`SPC t 8`	highlight any character past the 80th column
`SPC t f`	display the fill column (by default the fill column is set to 80)
`SPC t h h`	toggle highlight of the current line
`SPC t h i`	toggle highlight indentation levels
`SPC t h c`	toggle highlight indentation current column
`SPC t h s`	toggle syntax highlighting
`SPC t i`	toggle indentation guide at point
`SPC t l`	toggle truncate lines
`SPC t L`	toggle visual lines
`SPC t n`	toggle line numbers
`SPC t v`	toggle smooth scrolling

Key Binding	Description
`SPC T ~`	display ~ in the fringe on empty lines
`SPC T F`	toggle frame fullscreen
`SPC T f`	toggle display of the fringe
`SPC T m`	toggle menu bar
`SPC T M`	toggle frame maximize
`SPC T t`	toggle tool bar
`SPC T T`	toggle frame transparency and enter transparency transient state

Note: These toggles are all available via the `helm-spacemacs-help` interface (press `SPC h SPC` to display the `helm-spacemacs-help` buffer).

1. Global line numbers Line numbers can be toggled on in all `prog-mode` and `text-mode` buffers by setting the `dotspacemacs-line-numbers` variable in your `~/.spacemacs` to `t`.

   ```
   (setq-default dotspacemacs-line-numbers t)
   ```

   If it is set to `relative`, line numbers are show in a relative way:

   ```
   (setq-default dotspacemacs-line-numbers 'relative)
   ```

   `dotspacemacs-line-numbers` can also be set to a property list for finer control over line numbers activation.

   Available properties:

Property	Description
`:disabled-for-modes`	list of major modes where line numbering is inhibited
`:enabled-for-modes`	disable for all major modes except those listed. Takes precedence over
`:relative`	if non-nil, line numbers are relative to the position of the cursor
`:size-limit-kb`	size limit in kilobytes after which line numbers are not activated

   Examples:

   Disable line numbers in dired-mode, doc-view-mode, markdown-mode, org-mode, pdf-view-mode, text-mode as well as buffers over 1Mb:

```
(setq-default dotspacemacs-lines-numbers '(:relative nil
                                           :disabled-for-modes dired-mode
                                                               doc-view-mode
                                                               markdown-mode
                                                               org-mode
                                                               pdf-view-mode
                                                               text-mode
                                           :size-limit-kb 1000))
```

Relative line numbers only in c-mode and c++ mode with a size limit of `dotspacemacs-large-file-size`:

```
(setq-default dotspacemacs-lines-numbers '(:relative t
                                           :enabled-for-modes c-mode
                                                              c++-mode
                                           :size-limit-kb (* dotspacemacs-large-fi
```

Enable line numbers everywhere, except for buffers over 1Mb:

```
(setq-default dotspacemacs-lines-numbers '(:relative nil
                                           :size-limit-kb 1000))
```

Enable line numbers only in programming modes, except for c-mode and c++ mode:

```
(setq-default dotspacemacs-lines-numbers '(:relative nil
                                           :enabled-for-modes prog-mode
                                           :disabled-for-modes c-mode c++-mode
                                           :size-limit-kb (* dotspacemacs-large-fi
```

12.4 Mode-line

The mode line is a heavily customized powerline with the following capabilities:

- show the window number
- color code for current state
- show the number of search occurrences via anzu

- toggle flycheck info
- toggle battery info
- toggle minor mode lighters

Reminder of the color codes for the states:

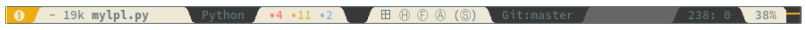
Evil State	Color
Normal	Orange
Insert	Green
Visual	Grey
Emacs	Blue
Motion	Purple
Replace	Chocolate
Lisp	Pink
Iedit/Iedit-Insert	Red

Some elements can be dynamically toggled:

Key Binding	Description
`SPC t m b`	toggle the battery status
`SPC t m c`	toggle the `org` task clock (available in `org` layer)
`SPC t m m`	toggle the minor mode lighters
`SPC t m M`	toggle the major mode
`SPC t m n`	toggle the cat! (if `colors` layer is declared in your dotfile)
`SPC t m p`	toggle the point character position
`SPC t m t`	toggle the time
`SPC t m T`	toggle the mode line itself
`SPC t m v`	toggle the version control info
`SPC t m V`	toggle the new version lighter

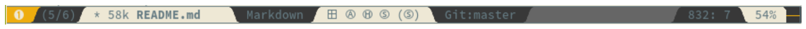
1. Powerline font installation for terminal-mode users Users who run Emacs in terminal mode may need to install the Powerline patched fonts and configure their terminal clients to use them to make the Powerline separators render correctly.

2. Flycheck integration When Flycheck minor mode is enabled, a new element appears showing the number of errors, warnings and info.



Flycheck integration in mode-line

3. Anzu integration Anzu shows the number of occurrence when performing a search. Spacemacs integrates nicely the Anzu status by displaying it temporarily when `n` or `N` are being pressed. See the `5/6` segment on the screenshot below.



```
(5/6)  * 58k README.md  Markdown  ⊞ Ⓐ Ⓗ Ⓢ (Ⓢ)  Git:master  832: 7  54%
```

Anzu integration in mode-line

4. Battery status integration fancy-battery displays the percentage of total charge of the battery as well as the time remaining to charge or discharge completely the battery.

   A color code is used for the battery status:

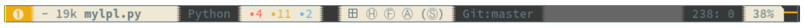
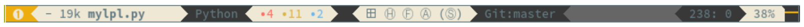
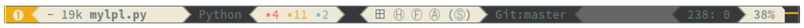
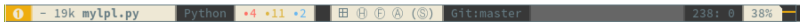
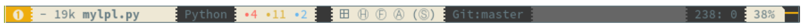
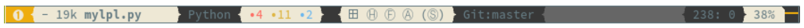
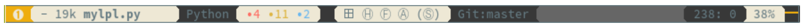
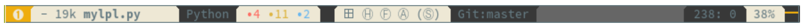
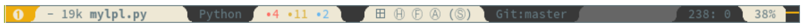
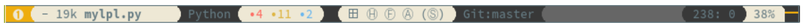
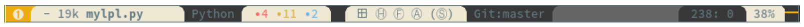
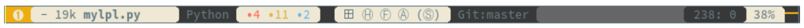
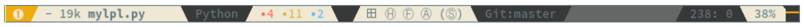
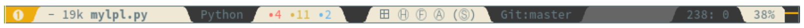
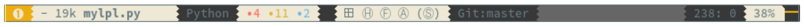
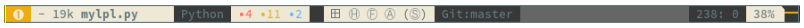
Battery State	Color
Charging	Green
Discharging	Orange
Critical	Red

   Note the these colors may vary depending on your theme.

5. Powerline separators It is possible to easily customize the `powerline separator` by setting the `powerline-default-separator` variable in your `~./spacemacs` and then recompiling the modeline. For instance if you want to set back the separator to the well-known `arrow` separator add the following snippet to your configuration file:

```
(defun dotspacemacs/user-config ()
  "This is were you can ultimately override default Spacemacs configuration.
This function is called at the very end of Spacemacs initialization."
  (setq powerline-default-separator 'arrow))
```

   To save you the time to try all the possible separators provided by the powerline, here is an exhaustive set of screenshots:

Separator	Screenshot
`alternate`	
`arrow`	
`arrow-fade`	
`bar`	
`box`	
`brace`	
`butt`	
`chamfer`	
`contour`	
`curve`	
`rounded`	
`roundstub`	
`slant`	
`wave`	
`zigzag`	
`nil`	

6. Minor Modes Spacemacs uses diminish mode to reduce the size of minor mode indicators:

   The minor mode area can be toggled on and off with `SPC t m m`

   Unicode symbols are displayed by default. Setting the variable `dotspacemacs-mode-line-unicode` to `nil` in your `~/.spacemacs` will display ASCII characters instead (may be useful in terminal if you cannot set an appropriate font).

   The letters displayed in the mode-line correspond to the key bindings used to toggle them.

   Some toggle have two flavors: local and global. The global version of the toggle can be reached using the `control` key.

Key Binding	Unicode	ASCII	Mode
`SPC t -`		-	centered-cursor mode
`SPC t 8`		8	toggle highlight of characters for long lines
`SPC t C-8`		8	global toggle highlight of characters for long lines
`SPC t C--`		-	global centered cursor
`SPC t a`		a	auto-completion
`SPC t c`		c	camel case motion with subword mode
`none`		e	evil-org mode
`SPC t E e`	`e`	Ee	emacs editing style (holy mode)
`SPC t E h`	`h`	Eh	hybrid editing style (hybrid mode)
`SPC t f`		f	fill-column-indicator mode
`SPC t F`		F	auto-fill mode
`SPC t g`		g	golden-ratio mode
`SPC t h i`	`i`	hi	toggle highlight indentation levels
`SPC t h c`	`c`	hc	toggle highlight indentation current column
`SPC t i`		i	indentation guide
`SPC t C-i`		i	global indentation guide
`SPC t I`		I	aggressive indent mode
`SPC t K`		K	which-key mode
`SPC t p`		p	smartparens mode
`SPC t C-p`		p	global smartparens
`SPC t s`		s	syntax checking (flycheck)
`SPC t S`		S	enabled in spell checking layer (flyspell)
`SPC t w`		w	whitespace mode
`SPC t C-w`		w	global whitespace
`SPC t W`		W	automatic whitespace cleanup (see `dotspacemacs-whitespac`
`SPC t C-W`		W	automatic whitespace cleanup globally
`SPC t y`		y	yasnippet mode

7. Customizing the mode-line Spacemacs uses Spaceline to provide its mode-line. It consists of a number of *segments* arranged on the left and right sides. These are defined in the variables `spaceline-left` and `spaceline-right`. Segments can be defined using `spaceline-define-segment`, and added to the appropriate location in the left or right hand side variables.

   Please see the Spaceline documentation for more information.

13 Layouts and workspaces

Layouts are window configurations with buffer isolation, each layout can define several workspaces (think of them as sub-layouts) sharing the same list of buffers as their parent layout.

13.1 Layouts

A layout is a window configuration associated with a list of buffers. The list of buffers can be an arbitrarily chosen set of buffers. Spacemacs provides some facilities to create meaningful sets of buffers, for instance the buffers related to a projectile project.

The name of the current layout appears in the mode-line at the far left (first element of the mode-line).

To create a new layout type a layout number that does not exist yet. For instance if you have two layouts currently then type `SPC l 3` to create a third layout.

13.1.1 The default layout

The `default` layout (the layout created at the startup of Emacs) is not displayed in the mode-line but it is possible to display it by setting the variable `dotspacemacs-display-default-layout` to `t`.

Its name is "default" by default but it can be changed by setting the variable `dotspacemacs-default-layout-name`.

The `default` layout is special because it has a global scope which means that all the opened buffers belong to it. So using only the `default` layout feels like not using layouts at all.

13.1.2 Project layouts

A project layout is bound to a projectile project. To create a project layout use `SPC p l`.

The name of the layout is the name of the project root directory.

13.1.3 Custom Layouts

Custom layouts can be defined using the macro `spacemacs|define-custom-layout`, they are accessible via `SPC l o`.

By convention the name of a custom layout should start with `@`.

Example of custom layout definition for `ERC` buffers:

```
(spacemacs|define-custom-layout "@ERC"
  :binding "E"
  :body
  (progn
    ;; hook to add all ERC buffers to the layout
    (defun spacemacs-layouts/add-erc-buffer-to-persp ()
      (persp-add-buffer (current-buffer)
                        (persp-get-by-name
                         erc-spacemacs-layout-name)))
    (add-hook 'erc-mode-hook #'spacemacs-layouts/add-erc-buffer-to-persp)
    ;; Start ERC
    (call-interactively 'erc)))
```

Then use `SPC l o E` to start ERC inside its own layout. Any new ERC buffer will be part of the custom layout.

Some custom layouts that ship with Spacemacs:

Name	Key Binding	Description
@Spacemacs	`e`	Custom perspective containing all buffers of `~/.emacs.d`
@ERC	`E`	Custom perspective containing all ERC buffers (needs the erc layer ena
@RCIRC	`i`	Custom perspective containing all RCIRC buffers (needs the rcirc layer
@Org	`o`	Custom perspective containing all the `org-agenda` buffers

13.1.4 Save/Load layouts into a file

With `SPC l s` and `SPC l L` you can save and load layouts to/from a file.

Note: By default, Spacemacs will automatically save the layouts under the name `persp-auto-save`.

Setting the variable `dotspacemacs-auto-resume-layouts` to `t` will automatically resume the last saved layouts.

13.1.5 Layout key bindings

The key bindings are registered in a transient state. The docstring of the transient state displays the existing layouts and the currently active layout has square brackets. Pressing a layout number will activate it (or create a new one) and exit the transient state. It is possible to just preview a layout with `Ctrl-<number>`. Pressing `TAB` will activate the previously selected layout.

Press `?` to toggle the full help.

Key Binding	Description
`SPC l`	activate the transient- state
`?`	toggle the documentation
`[0..9]`	switch to nth layout
`[C-0..C-9]`	switch to nth layout and keep the transient state active
`<tab>`	switch to the latest layout
`a`	add a buffer to the current layout
`A`	add all the buffers from another layout in the current one
`b`	select a buffer in the current layout
`d`	delete the current layout and keep its buffers
`D`	delete the other layouts and keep their buffers
`h`	go to default layout
`C-h`	previous layout in list
`l`	select/create a layout with helm
`L`	load layouts from file
`C-l`	next layout in list
`n`	next layout in list
`N`	previous layout in list
`o`	open a custom layout
`p`	previous layout in list
`r`	remove current buffer from layout
`R`	rename current layout
`s`	save layouts
`t`	display a buffer without adding it to the current layout
`w`	workspaces transient state (needs eyebrowse layer enabled)
`x`	kill current layout with its buffers
`X`	kill other layouts with their buffers

13.2 Workspaces

Workspaces are sub-layouts, they allow to define multiple layouts into a given layout, those layouts share the same buffer as the parent layout.

The currently active workspace number is displayed before the window number, for instance "|" or "1|4" means the fourth window of the first workspace.

Any new layout comes with a default workspace which is the workspace 1.

Switching to a workspace that does not exist in the current layout will create a new one. For instance at startup you can press `SPC l w 2` to create the workspace 2 in the `default` layout.

When created a workspace is anonymous, you can give them a name with `SPC l w R`.

13.2.1 Workspace key bindings

The key bindings are registered in a transient state. The docstring of the transient state displays the existing workspaces and the currently active workspace has square brackets. Pressing a workspace number will activate it (or create a new one) and exit the transient state. It is possible to just preview a workspace with `Ctrl-<number>`. Pressing `TAB` will activate the previously selected workspace.

Press `?` to toggle the full help.

Key Binding	Description
`SPC l w`	activate the transient state
`?`	toggle the documentation
`[0..9]`	switch to nth workspace
`[C-0..C-9]`	switch to nth workspace and keep the transient state active
`TAB`	switch to last active workspace
`d`	close current workspace
`n` or `l`	switch to next workspace
`N` or `p` or `h`	switch to previous workspace
`R`	set a tag to the current workspace
`w`	switched to tagged workspace

There are also some handy globally available key bindings related to workspaces:

Key Binding	Description
`gt`	go to next workspace
`gT`	go to previous workspace
`SPC b W`	go to workspace and window by buffer

14 Commands

14.1 Vim key bindings

Spacemacs is based on `Vim` modal user interface to navigate and edit text. If you are not familiar with the `Vim` way of editing text you can try the evil-tutor lessons by pressing `SPC h T` at any time.

14.1.1 Escaping

Spacemacs uses evil-escape to easily switch between `insert state` and `normal state` by quickly pressing the `fd` keys.

The choice of `fd` was made to be able to use the same sequence to escape from "everything" in Emacs:

- escape from all stock evil states to normal state
- escape from evil-lisp-state to normal state
- escape from evil-iedit-state to normal state
- abort evil ex command
- quit minibuffer
- abort isearch
- quit magit buffers
- quit help buffers
- quit apropos buffers
- quit ert buffers
- quit undo-tree buffer
- quit paradox
- quit gist-list menu
- quit helm-ag-edit
- hide neotree buffer

If you find yourself in a buffer where the Spacemacs (`SPC`) or Vim keybindings don't work you can use this to get back to `normal state` (for example in `SPC SPC customize` press `fd` to make `SPC b b` work again).

This sequence can be customized in your `~/.spacemacs`. Example to set it to `jj`:

```
(defun dotspacemacs/user-config ()
  (setq-default evil-escape-key-sequence "jj"))
```

Note: Although `jj` or `jk` are popular choices of vim users, these key sequences are not optimal for Spacemacs. Indeed it is very easy in `visual state` to press quickly `jj` and inadvertently escape to `normal state`.

14.1.2 Executing Vim and Emacs ex/M-x commands

Command	Key Binding
Vim (ex-command)	`:`
Emacs (M-x)	`SPC SPC`

The emacs command key `SPC` (executed after the leader key) can be changed with the variable `dotspacemacs-emacs-command-key` of your `~/.spacemacs`.

14.1.3 Leader key

On top of `Vim` modes (modes are called states in Spacemacs) there is a special key called the leader key which once pressed gives a whole new keyboard layer. The leader key is by default `SPC` (space). It is possible to change this key with the variable `dotspacemacs-leader-key`.

14.1.4 Additional text objects

Additional text objects are defined in Spacemacs:

Object	Description
`a`	an argument
`g`	the entire buffer
`$`	text between `$`
`*`	text between `*`
`8`	text between `/*` and `*/`
`%`	text between `%`
`\vert`	text between `\vert`

14.2 Reserved prefix command for user

`SPC o` and `SPC m o` are reserved for the user. Setting key bindings behind these is **guaranteed** to never conflict with Spacemacs default key bindings.

Example: Put `(spacemacs/set-leader-keys "oc" 'org-capture)` inside `dotspacemacs/user-config` in your `~/.spacemacs` file, to be able to use `SPC o c` to run org mode capture.

14.3 Completion

Spacemacs is powered by one of two incremental completion and selection narrowing frameworks: Helm (default) or Ivy. To use Ivy, add the `ivy` layer to your list of enabled layers. If the `ivy` layer is not enabled, Helm will be enabled automatically. (Please note that, as Helm is the more mature of the two, some functions may be unavailable if you choose Ivy.)

These completion systems are the central control towers of Spacemacs, they are used to manage buffers, projects, search results, configuration layers, toggles and more…

Mastering your choice of completion system will make you a Spacemacs power user.

14.3.1 Helm

Do not hesitate to read the Helm documentation wiki.

1. C-z and Tab switch The command bound to `C-z` is much more useful than the one bound to Tab, so it makes sense to swap them. It's also recommended here.

2. Helm focus If you find yourself unable to return focus to Helm (after a careless mouse-click for example), use `SPC w b` to return focus to the minibuffer.

3. Helm transient state Spacemacs defines a transient state for `Helm` to make it work like Vim's Unite plugin.

   Initiate the transient state with `M-SPC` or `s-M-SPC` while in a `Helm` buffer.

Key Binding	Description
`M-SPC` or `s-M-SPC`	initiate the transient state
`q`	quit transient state
`TAB`	switch to actions page and leave the transient state
`1`	execute action 0
`2`	execute action 1
`3`	execute action 2
`4`	execute action 3
`5`	execute action 4
`6`	execute action 5
`7`	execute action 6
`8`	execute action 7
`9`	execute action 8
`0`	execute action 9
`a`	switch to actions page
`g`	go to first candidate
`G`	go to last candidate
`h`	go to previous source
`j`	select next candidate
`k`	select previous candidate
`l`	go to next source
`t`	mark current candidate
`T`	mark all candidates
`v`	execute persistent action

14.3.2 Ivy

If you choose `ivy` as completion system, make sure to read official manual. In case you don't want to read everything, at least familiarise with minibuffer key bindings. Some useful key bindings are presented in following table.

Key Binding	Description
`C-m` or `RET`	call default action on current candidate
`M-o`	show the list of valid actions on current candidate (then press any of described keys to
`C-M-m`	the same as `RET` but doesn't close completion minibuffer
`C-M-o`	the same as `M-o` but doesn't close completion minibuffer
`C-'`	use avy to quickly select completion on current page (sometimes faster than using arr

14.4 Discovering

14.4.1 Key bindings

1. Which-key A help buffer is displayed each time the `SPC` key is pressed in normal mode. It lists the available key bindings and their associated commands.

   By default the which-key buffer will be displayed quickly after the key has been pressed. You can change the delay by setting the variable `dotspacemacs-which-key-delay` to your liking (the value is in second).

2. Helm describe key bindings It is possible to search for specific key bindings by pressing `SPC ?`.

   To narrow the list to some key bindings using the leader key type a pattern like this regular expression: `SPC\ b` which would list all `buffer` related bindings.

14.4.2 Getting help

`Describe functions` are powerful Emacs introspection commands to get information about functions, variables, modes etc. These commands are bound thusly:

Key Binding	Description
`SPC h d b`	describe bindings in a `helm` buffer
`SPC h d c`	describe current character under point
`SPC h d d`	describe current expression under point
`SPC h d f`	describe a function
`SPC h d F`	describe a face
`SPC h d k`	describe a key
`SPC h d K`	describe a keymap
`SPC h d l`	copy last pressed keys that you can paste in gitter chat
`SPC h d m`	describe current modes
`SPC h d p`	describe a package (Emacs built-in function)
`SPC h d P`	describe a package (Spacemacs layer information)
`SPC h d s`	copy system information that you can paste in gitter chat
`SPC h d t`	describe a theme
`SPC h d v`	describe a variable

Other help key bindings:

Key Binding	Description
`SPC h SPC`	discover Spacemacs documentation, layers and packages using `helm`
`SPC h i`	search in info pages with the symbol at point
`SPC h k`	show top-level bindings with `which-key`
`SPC h m`	search available man pages
`SPC h n`	browse emacs news

Navigation key bindings in `help-mode`:

Key Binding	Description
`g b` or `[`	go back (same as clicking on `[back]` button)
`g f` or `]`	go forward (same as clicking on `[forward]` button)
`g h`	go to help for symbol under point

Reporting an issue:

Key Binding	Description
`SPC h I`	Open Spacemacs GitHub issue page with pre-filled information
`SPC u SPC h I`	Open Spacemacs GitHub issue page with pre-filled information - include last presse

Note: If these two bindings are used with the `*Backtrace*` buffer open, the backtrace is automatically included

14.4.3 Available layers

All layers can be easily discovered via `helm-spacemacs-help` accessible with `SPC h SPC`.

The following helm actions are available:

- default: open the layer `README.org`
- 2nd: open the layer `packages.el`

1. Available packages in Spacemacs `helm-spacemacs-help` also lists all the packages available in Spacemacs. The entry format is `(layer) packages`. If you type `flycheck` you'll be able to see all the layers where `flycheck` is used.

   The following helm actions are available on packages:

   - default: go the package init function

2. New packages from ELPA repositories `package-list-packages` is where you can browse for all available packages in the different Elpa repositories. It is possible to upgrade packages from there but it is not recommended, use the `[Update Packages]` link on the Spacemacs startup page instead.

   Spacemacs uses Paradox instead of `package-list-packages` to list available ELPA packages. Paradox enhances the package list buffer with better feedbacks, new filters and Github information like the number of stars. Optionally you can also star packages directly in the buffer.

   Important Note 1: Installing a new package from `Paradox` won't make it persistent. To install a package persistently you have to add it explicitly to a configuration layer.

   Important Note 2: Don't *update* your packages from `Paradox` or `package-list-packages` because they don't support the rollback feature of Spacemacs.

Key Binding	Description
`SPC a k`	launch `paradox`
`/`	evil-search
`f k`	filter by keywords
`f r`	filter by regexp
`f u`	display only installed package with updates available
`h`	go left
`H`	show help (not accurate)
`j`	go down
`k`	go up
`l`	go right
`L`	show last commits
`n`	next search occurrence
`N`	previous search occurrence
`o`	open package homepage
`r`	refresh
`S P`	sort by package name
`S S`	sort by status (installed, available, etc...)
`S *`	sort by Github stars
`v`	`visual state`
`V`	`visual-line state`
`x`	execute (action flags)

14.4.4 Toggles

`helm-spacemacs-help` is also a central place to discover the available toggles. To display only the toggles source press `C-l` (or in Helm transient state you can press just `l`).

The following helm actions are available on packages:

- default: toggle on/off

Tips Use `SPC h l` to resume the last helm session. It is handy to quickly toggle on and off a toggle.

14.5 Navigating

14.5.1 Point/Cursor

Navigation is performed using the Vi key bindings `hjkl`.

Key Binding	Description
`h`	move cursor left
`j`	move cursor down
`k`	move cursor up
`l`	move cursor right
`H`	move cursor to the top of the screen
`L`	move cursor to the bottom of the screen
`SPC j 0`	go to the beginning of line (and set a mark at the previous location in the line)
`SPC j $`	go to the end of line (and set a mark at the previous location in the line)
`SPC t -`	lock the cursor at the center of the screen

1. Smooth scrolling smooth-scrolling prevent the point to jump when it reaches the top or bottom of the screen. It is enabled by default.

   On Windows, you may want to disable it. To disable the smooth scrolling set the `dotspacemacs-smooth-scrolling` variable in your `~/.spacemacs` to `nil`:

   ```
   (setq-default dotspacemacs-smooth-scrolling nil)
   ```

   You can also toggle smooth scrolling with `SPC t v`.

14.5.2 Vim motions with avy

Spacemacs uses the `evil` integration of avy which enables the invocation of `avy` during motions.

For instance, it is useful for deleting a set of visual lines from the current line. Try the following sequence in a buffer containing some text: `d SPC j l`, followed by selecting an avy candidate.

Key Binding	Description
`SPC j b`	go back to the previous location (before the jump)
`SPC j j`	initiate avy jump char
`SPC j w`	initiate avy jump word
`SPC j l`	initiate avy jump line

1. ace-link mode Similar to `avy`, ace-link allows one to jump to any link in `help-mode` and `info-mode` with two key strokes.

Key Binding	Description
`o`	initiate ace link mode in `help-mode` and `info-mode`

14.5.3 Unimpaired bindings

Spacemacs comes with a built-in port of tpope's vim-unimpaired.

This plugin provides several pairs of bracket maps using `[` to denote previous, and `]` as next.

KeyBindings	Description
`[ SPC`	Insert space above
`] SPC`	Insert space below
`[ b`	Go to previous buffer
`] b`	Go to next buffer
`[ f`	Go to previous file in directory
`] f`	Go to next file in directory
`[ l`	Go to the previous error
`] l`	Go to the next error
`[ h`	Go to the previous vcs hunk
`] h`	Go to the next vcs hunk
`[ q`	Go to the previous error
`] q`	Go to the next error
`[ t`	Go to the previous frame
`] t`	Go to the next frame
`[ w`	Go to the previous window
`] w`	Go to the next window
`[ e`	Move line up
`] e`	Move line down
`[ p`	Paste above current line
`] p`	Paste below current line
`g p`	Select pasted text

14.5.4 Jumping, Joining and Splitting

The `SPC j` prefix is for jumping, joining and splitting.

1. Jumping

Key Binding	Description
`SPC j 0`	go to the beginning of line (and set a mark at the previous location in the line)
`SPC j $`	go to the end of line (and set a mark at the previous location in the line)
`SPC j b`	undo a jump (go back to previous location)
`SPC j d`	jump to a listing of the current directory
`SPC j D`	jump to a listing of the current directory (other window)
`SPC j f`	jump to the definition of an Emacs Lisp function
`SPC j i`	jump to a definition in buffer (imenu)
`SPC j I`	jump to a definition in any buffer (imenu)
`SPC j j`	jump to a character in the buffer (works as an evil motion)
`SPC j J`	jump to a suite of two characters in the buffer (works as an evil motion)
`SPC j k`	jump to next line and indent it using auto-indent rules
`SPC j l`	jump to a line with avy (works as an evil motion)
`SPC j q`	show the dumb-jump quick look tooltip
`SPC j u`	jump to a URL in the current buffer
`SPC j v`	jump to the definition/declaration of an Emacs Lisp variable
`SPC j w`	jump to a word in the current buffer (works as an evil motion)

2. Joining and splitting

Key Binding	Description
`J`	join the current line with the next line
`SPC j k`	go to next line and indent it using auto-indent rules
`SPC j n`	split the current line at point, insert a new line and auto-indent
`SPC j s`	split a quoted string or s-expression in place
`SPC j S`	split a quoted string or s-expression, insert a new line and auto-indent

14.5.5 Window manipulation

1. Window manipulation key bindings Every window has a number displayed at the start of the mode-line and can be quickly accessed using `SPC number`.

Key Binding	Description
`SPC 1`	go to window number 1
`SPC 2`	go to window number 2
`SPC 3`	go to window number 3
`SPC 4`	go to window number 4
`SPC 5`	go to window number 5
`SPC 6`	go to window number 6
`SPC 7`	go to window number 7
`SPC 8`	go to window number 8
`SPC 9`	go to window number 9
`SPC 0`	go to window number 0

Windows manipulation commands (start with `w`):

Key Binding	Description
`SPC w TAB`	switch to alternate window in the current frame (switch back and forth)
`SPC w =`	balance split windows
`SPC w b`	force the focus back to the minibuffer (useful with **helm** popups)
`SPC w c`	maximize/minimize a window and center it
`SPC w C`	maximize/minimize a window and center it using ace-window
`SPC w d`	delete a window
`SPC u SPC w d`	delete a window and its current buffer (does not delete the file)
`SPC w D`	delete another window using ace-window
`SPC u SPC w D`	delete another window and its current buffer using ace-window
`SPC w t`	toggle window dedication (dedicated window cannot be reused by a mod
`SPC w f`	toggle follow mode
`SPC w F`	create new frame
`SPC w h`	move to window on the left
`SPC w H`	move window to the left
`SPC w j`	move to window below
`SPC w J`	move window to the bottom
`SPC w k`	move to window above
`SPC w K`	move window to the top
`SPC w l`	move to window on the right
`SPC w L`	move window to the right
`SPC w m`	maximize/minimize a window (maximize is equivalent to delete other wi
`SPC w M`	swap windows using ace-window
`SPC w o`	cycle and focus between frames
`SPC w p m`	open messages buffer in a popup window
`SPC w p p`	close the current sticky popup window
`SPC w r`	rotate windows forward
`SPC w R`	rotate windows backward
`SPC w s` or `SPC w -`	horizontal split
`SPC w S`	horizontal split and focus new window
`SPC w u`	undo window layout (used to effectively undo a closed window)
`SPC w U`	redo window layout
`SPC w v` or `SPC w /`	vertical split
`SPC w V`	vertical split and focus new window
`SPC w w`	cycle and focus between windows
`SPC w W`	select window using ace-window

2. Window manipulation transient state A convenient window manipulation transient state allows performing most of the actions listed above. The transient state allows additional actions as well like window resiz-

ing.

Key Binding	Description
`SPC w .`	initiate transient state
`?`	display the full documentation in minibuffer
`0`	go to window number 0
`1`	go to window number 1
`2`	go to window number 2
`3`	go to window number 3
`4`	go to window number 4
`5`	go to window number 5
`6`	go to window number 6
`7`	go to window number 7
`8`	go to window number 8
`9`	go to window number 9
`/`	vertical split
`-`	horizontal split
`[`	shrink window horizontally
`]`	enlarge window horizontally
`{`	shrink window vertically
`}`	enlarge window vertically
`d`	delete window
`D`	delete other windows
`g`	toggle `golden-ratio` on and off
`h`	go to window on the left
`j`	go to window below
`k`	go to window above
`l`	go to window on the right
`H`	move window to the left
`J`	move window to the bottom
`K`	move bottom to the top
`L`	move window to the right
`o`	focus other frame
`r`	rotate windows forward
`R`	rotate windows backward
`s`	horizontal split
`S`	horizontal split and focus new window
`u`	undo window layout (used to effectively undo a closed window)
`U`	redo window layout
`v`	vertical split
`V`	horizontal split and focus new window
`w`	focus other window
Any other key	leave the transient state

3. Golden ratio If you resize windows like crazy you may want to give a try to golden-ratio.

   `golden-ratio` resizes windows dynamically depending on whether they are selected or not. By default `golden-ratio` is off.

   The mode can be toggled on and off with `SPC t g`.

14.5.6 Buffers and Files

By default Spacemacs uses `helm` to open files.

1. Buffers manipulation key bindings Buffer manipulation commands (start with `b`):

Key Binding	Description
`SPC TAB`	switch to alternate buffer in the current window (switch back and forth)
`SPC b b`	switch to a buffer using helm
`SPC b d`	kill the current buffer (does not delete the visited file)
`SPC u SPC b d`	kill the current buffer and window (does not delete the visited file)
`SPC b D`	kill a visible buffer using ace-window
`SPC u SPC b D`	kill a visible buffer and its window using ace-window
`SPC b C-d`	kill buffers using a regular expression
`SPC b e`	erase the content of the buffer (ask for confirmation)
`SPC b h`	open `*spacemacs*` home buffer
`SPC b n`	switch to next buffer avoiding special buffers
`SPC b m`	kill all buffers except the current one
`SPC u SPC b m`	kill all buffers and windows except the current one
`SPC b M`	kill all buffers matching the regexp
`SPC b p`	switch to previous buffer avoiding special buffers
`SPC b P`	copy clipboard and replace buffer (useful when pasting from a browser)
`SPC b R`	revert the current buffer (reload from disk)
`SPC b s`	switch to the `*scratch*` buffer (create it if needed)
`SPC b w`	toggle read-only (writable state)
`SPC b Y`	copy whole buffer to clipboard (useful when copying to a browser)
`z f`	Make current function or comments visible in buffer as much as possible

2. Buffers manipulation transient state A convenient buffer manipulation transient state allows to quickly cycles through the opened buffer and kill them.

Key Binding	Description
`SPC b .`	initiate transient state
`K`	kill current buffer
`n`	go to next buffer (avoid special buffers)
`N`	go to previous buffer (avoid special buffers)
Any other key	leave the transient state

3. Special Buffers Unlike vim, emacs creates many buffers that most people do not need to see. Some examples are `*Messages*` and `*Compile-Log*`. Spacemacs tries to automatically ignore buffers that are not useful. However, you may want to change the way Spacemacs marks buffers as useful. For instructions, see the special buffer howto.

4. Files manipulations key bindings Files manipulation commands (start with `f`):

Key Binding	Description
`SPC f b`	go to file bookmarks
`SPC f c`	copy current file to a different location
`SPC f C d`	convert file from unix to dos encoding
`SPC f C u`	convert file from dos to unix encoding
`SPC f D`	delete a file and the associated buffer (ask for confirmation)
`SPC f E`	open a file with elevated privileges (sudo edit)
`SPC f f`	open file with `helm`
`SPC f F`	try to open the file under point `helm`
`SPC f h`	open binary file with `hexl` (a hex editor)
`SPC f j`	jump to the current buffer file in dired
`SPC f J`	open a junk file, in mode determined by the file extension provided (defaulting t
`SPC f l`	open file literally in `fundamental mode`
`SPC f L`	Locate a file (using `locate`)
`SPC f o`	open a file using the default external program
`SPC f R`	rename the current file
`SPC f s`	save a file
`SPC f S`	save all files
`SPC f r`	open a recent file with `helm`
`SPC f t`	toggle file tree side bar using NeoTree
`SPC f v d`	add a directory variable
`SPC f v f`	add a local variable to the current file
`SPC f v p`	add a local variable to the first line of the current file
`SPC f y`	show and copy current file absolute path in the minibuffer

5. Emacs and Spacemacs files Convenient key bindings are located under the prefix `SPC f e` to quickly navigate between `Emacs` and Spacemacs specific files.

Key Binding	Description
`SPC f e d`	open the spacemacs dotfile (`~/.spacemacs`)
`SPC f e D`	open `ediff` buffer of `~/.spacemacs` and `.spacemacs.template`
`SPC f e f`	discover the `FAQ` using `helm`
`SPC f e i`	open the all mighty `init.el`
`SPC f e l`	locate an Emacs library
`SPC f e R`	resync the dotfile with spacemacs
`SPC f e v`	display and copy the spacemacs version

6. Browsing files with Helm In `vim` and `hybrid` styles, Spacemacs remap the navigation in Helm find-files to keep finger on the home row.

Key Binding	Description
`C-h`	go up one level (parent directory
`C-H`	describe key (replace `C-h`)
`C-j`	go to previous candidate
`C-k`	go to next candidate
`C-l`	enter current directory

14.5.7 Ido

Spacemacs displays the `ido` minibuffer vertically thanks to the ido-vertical-mode.

Basic `ido` operations can be done with `Ctrl` key:

Key Binding	Description
`C-<return>`	open a `dired buffer`
`M-<return>`	open a `dired buffer` in terminal
`C-d`	delete selected file (ask for confirmation)
`C-h`	go to parent directory
`C-j`	select next file or directory
`C-k`	select previous file or directory
`C-l`	open the selected file
`C-n`	select next file or directory
`C-o`	open selected file in other window
`C-p`	select previous file or directory
`C-s`	open selected file in a vertically split window
`C-t`	open selected file in a new frame
`C-v`	open selected file in a horizontally split window
`C-S-h`	go to previous directory
`C-S-j` or `C-S-n`	next history element
`C-S-k` or `C-S-p`	previous history element
`C-S-l`	go to next directory

14.5.8 Ido transient state

Spacemacs defines a transient state for `ido`.

Initiate the transient state with `M-SPC` or `s-M-SPC` while in an `ido` buffer.

Key Binding	Description
`M-SPC` or `s-M-SPC`	initiate or leave the transient state
`?`	display help
`e`	open dired
`h`	delete backward or parent directory
`j`	next match
`J`	sub directory
`k`	previous match
`K`	parent directory
`l`	select match
`n`	next directory in history
`o`	open in other window
`p`	previous directory in history
`q`	quit transient state
`s`	open in a new horizontal split
`t`	open in other frame
`v`	open in a new vertical split

14.5.9 NeoTree file tree

Spacemacs provides a quick and simple way to navigate in an unknown project file tree with NeoTree.

To toggle the `NeoTree` buffer press `SPC f t` or `SPC p t` (the latter open NeoTree with the root set to the projectile project root).

The NeoTree window always has the number `0` so it does not shift the current number of the other windows. To select the NeoTree window you then use `SPC 0`.

VCS integration is supported, the file color will change depending on its current state. With default `spacemacs-dark` theme:

- green: new file
- purple: modified file

1. NeoTree navigation Navigation is centered on the `hjkl` keys with the hope of providing a fast navigation experience like in ranger:

Key Binding	Description
`h`	collapse expanded directory or go to parent node
`H`	select previous sibling
`j`	select next file or directory
`J`	select next expanded directory on level down
`k`	select previous file or directory
`K`	select parent directory, when reaching the root change it to parent directory
`l` or `RET`	expand directory
`L`	select next sibling
`R`	make a directory the root directory

   Note: Point is automatically set to the first letter of a node for a smoother experience.

2. Opening files with NeoTree By default a file is opened in the last active window. It is possible to choose window number where to open a file by using a numeric argument, for instance `2 l` or `2 RET` will open the current file in window 2. It is also possible to open the file in a split window with `|` and `-`:

Key Binding	Description
`l` or `RET`	open file in last active window
`# l` or `# RET`	open file in window number `#`
`¦`	open file in an vertically split window
`-`	open file in an horizontally split window

3. Other NeoTree key bindings

Key Binding	Description
`TAB`	toggle stretching of the buffer
`c`	create a node
`C`	copy a node
`d`	delete a node
`gr`	refresh
`s`	toggle showing of hidden files
`q` or `fd`	hide `NeoTree` buffer
`r`	rename a node
`?`	show help

4. NeoTree mode-line The mode-line has the following format `[x/y] d (D:a, F:b)` where:

   - `x` is the index of the current selected file or directory
   - `y` the total number of items (file and directory) in the current directory
   - `d` the name of the current directory
   - `a` the number of directories in the current directory
   - `b` the number of files in the current directory

5. NeoTree Source Control Integration If you would like NeoTree to show source control information, you can use the setting `neo-vc-integration`. It is a list containing the possible values:

Setting	Description
`face`	Show information by changing the color of the file/directory name.
`char`	Show information with a character to the left of the file/directory name.

The default is `nil` (do not show source control information), which is recommended.

For example,

```
(setq neo-vc-integration 'face)
```

Note: At this time, it is not recommended to set this to anything other than `nil`. Otherwise, it will become very slow with larger source trees. See `https://github.com/jaypei/emacs-neotree/issues/126` for more information.

6. NeoTree Theme You can change the NeoTree theme by using the setting `neo-theme`. Possible values are:

Setting	Description
`classic`	Use an icon to display items - only suitable for gui mode.
`ascii`	The simplest style, it will use `x`, `-` to display fold status.
`arrow`	Use unicode arrows to display fold status.
`nerd`	Use the NERDTree indentation mode and arrows.

The default is `classic`.

Use `nerd` if you want it to look most like NERDTree in VIM. For example:

```
(setq neo-theme 'nerd)
```

14.5.10 Bookmarks

Bookmarks can be set anywhere in a file. Bookmarks are persistent. They are very useful to jump to/open a known project. Spacemacs uses `helm-bookmarks` to manage them.

Open an `helm` window with the current bookmarks by pressing: `SPC f b`

Then in the `helm-bookmarks` buffer:

Key Binding	Description
`C-d`	delete the selected bookmark
`C-e`	edit the selected bookmark
`C-f`	toggle filename location
`C-o`	open the selected bookmark in another window

To save a new bookmark, just type the name of the bookmark and press `RET`.

14.5.11 DocView mode

`doc-view-mode` is a built-in major mode to view DVI, PostScript (PS), PDF, OpenDocument, and Microsoft Office documents.

Key Binding	Description
`/`	search forward
`?`	search backward
`+`	enlarge
`-`	shrink
`gg`	go to first page
`G`	go to last page
`gt`	go to page number
`h`	previous page
`H`	adjust to height
`j`	next line
`k`	previous line
`K`	kill proc and buffer
`l`	next page
`n`	go to next search occurrence
`N`	go to previous search occurrence
`P`	fit page to window
`r`	revert
`W`	adjust to width
`C-d`	scroll down
`C-k`	kill proc
`C-u`	scroll up
`C-c C-c`	toggle display text and image display
`C-c C-t`	open new buffer with doc's text contents

14.6 Auto-saving

14.6.1 Frequency of auto-saving

By default auto-saving of files is performed every 300 characters and every 30 seconds of idle time which can be changed by setting to a new value the variables `auto-save-interval` and `auto-save-timeout` respectively.

14.6.2 Location of auto-saved files

Auto-save of modified files can be performed in-place on the original file itself *or* in the cache directory (in this case the original file will remain unsaved).

By default Spacemacs auto-save the file in the cache directory.

To modify the location set the variable `dotspacemacs-auto-save-file-location` to `original` or `cache`.

Local files are auto-saved in a sub-directory called `site` in the `cache` directory whereas remote files (i.e. files edited over TRAMP) are auto-saved in a sub-directory called `dist`.

14.6.3 Disable auto-save

To disable auto-saving set the variable `dotspacemacs-auto-save-file-location` to `nil`.

You can toggle auto-save in a buffer by calling the command `auto-save-mode`.

14.7 Searching

14.7.1 With an external tool

Spacemacs can be interfaced with different search utilities like:

- ack
- grep
- ag
- pt

The search commands in Spacemacs are organized under the `SPC s` prefix with the next key is the tool to use and the last key is the scope. For instance `SPC s a b` will search in all opened buffers using `ag`.

If the last key (determining the scope) is uppercase then the current region or symbol under point is used as default input for the search. For instance `SPC s a B` will search with symbol under point (if there is no active region).

If the tool key is omitted then a default tool will be automatically selected for the search. This tool corresponds to the first tool found on the system of the list `dotspacemacs-search-tools`, the default order is `ag`, `pt`, `ack` then `grep`. For instance `SPC s b` will search in the opened buffers using `pt` if `ag` has not been found on the system.

The tool keys are:

Tool	Key
ag	a
grep	g
ack	k
pt	t

The available scopes and corresponding keys are:

Scope	Key
opened buffers	b
files in a given directory	f
current project	p

It is possible to search in the current file by double tapping the second key of the sequence, for instance `SPC s a a` will search in the current file with `ag`.

Notes:

- `ag` and `pt` are optimized to be used in a source control repository but they can be used in an arbitrary directory as well.
- It is also possible to search in several directories at once by marking them in the helm buffer.

Beware if you use `pt`, TCL parser tools also install a command line tool called `pt`.

1. Useful key bindings

Key Binding	Description
`F3`	in a `helm` or `ivy` buffer, save results to a regular buffer
`SPC r l`	resume the last `completion` buffer
`SPC r s` or `SPC s l`	resume search buffer (completion or converted search buffer)
`SPC s ``	go back to the previous place reached with `helm-ag`
Prefix argument	will ask for file extensions

When results have been saved in a regular buffer with `F3`, that buffer supports browsing through the matches with Spacemacs' `next-error` and `previous-error` bindings (`SPC e n` and `SPC e p`) as well as the error transient state (`SPC e`).

2. Searching in current file

Key Binding	Description
`SPC s s`	search with the first found tool
`SPC s S`	search with the first found tool with default input
`SPC s a a`	`ag`
`SPC s a A`	`ag` with default input
`SPC s g g`	`grep`
`SPC s g G`	`grep` with default input

3. Searching in all open buffers visiting files

Key Binding	Description
`SPC s b`	search with the first found tool
`SPC s B`	search with the first found tool with default input
`SPC s a b`	`ag`
`SPC s a B`	`ag` with default text
`SPC s g b`	`grep`
`SPC s g B`	`grep` with default text
`SPC s k b`	`ack`
`SPC s k B`	`ack` with default text
`SPC s t b`	`pt`
`SPC s t B`	`pt` with default text

4. Searching in files in an arbitrary directory

Key Binding	Description
`SPC s f`	search with the first found tool
`SPC s F`	search with the first found tool with default input
`SPC s a f`	`ag`
`SPC s a F`	`ag` with default text
`SPC s g f`	`grep`
`SPC s g F`	`grep` with default text
`SPC s k f`	`ack`
`SPC s k F`	`ack` with default text
`SPC s t f`	`pt`
`SPC s t F`	`pt` with default text

5. Searching in a project

Key Binding	Description
`SPC /` or `SPC s p`	search with the first found tool
`SPC *` or `SPC s P`	search with the first found tool with default input
`SPC s a p`	`ag`
`SPC s a P`	`ag` with default text
`SPC s g p`	`grep` with default text
`SPC s k p`	`ack`
`SPC s k P`	`ack` with default text
`SPC s t p`	`pt`
`SPC s t P`	`pt` with default text

Hint: It is also possible to search in a project without needing to open a file beforehand. To do so use `SPC p p` and then `C-s` on a given project to directly search into it like with `SPC s p`.

6. Searching the web

Key Binding	Description
`SPC s w g`	Get Google suggestions in emacs. Opens Google results in Browser.
`SPC s w w`	Get Wikipedia suggestions in emacs. Opens Wikipedia page in Browser.

14.7.2 Persistent highlighting

Spacemacs uses `evil-search-highlight-persist` to keep the searched expression highlighted until the next search. It is also possible to clear the highlighting by pressing `SPC s c` or executing the ex command `:noh`.

14.7.3 Highlight current symbol

Spacemacs supports highlighting of the current symbol on demand (provided by auto-highlight-symbol mode) and adds a transient state to easily navigate and rename this symbol.

It is also possible to change the range of the navigation on the fly to:

- buffer
- function
- visible area

To initiate the highlighting of the current symbol under point press `SPC s h`.

Navigation between the highlighted symbols can be done with the commands:

Key Binding	Description
`*`	initiate navigation transient state on current symbol and jump forwards
`#`	initiate navigation transient state on current symbol and jump backwards
`SPC s e`	edit all occurrences of the current symbol(/)
`SPC s h`	highlight the current symbol and all its occurrence within the current range
`SPC s H`	go to the last searched occurrence of the last highlighted symbol
`SPC t h a`	toggle automatic highlight of symbol under point after `ahs-idle-interval` seconds

In 'Spacemacs' highlight symbol transient state:

Key Binding	Description
`e`	edit occurrences (*)
`n`	go to next occurrence
`N`	go to previous occurrence
`d`	go to next definition occurrence
`D`	go to previous definition occurrence
`r`	change range (`function`, `display area`, `whole buffer`)
`R`	go to home occurrence (reset position to starting occurrence)
Any other key	leave the navigation transient state

(*) using iedit or the default implementation of `auto-highlight-symbol`

The transient state text in minibuffer display the following information:

```
<M> [6/11]* press (n/N) to navigate, (e) to edit, (r) to change range or (R)
for reset
```

Where `<M> [x/y]*` is:

- M: the current range mode
- `<B>`: whole buffer range
- `<D>`: current display range
- `<F>`: current function range

- `x`: the index of the current highlighted occurrence
- `y`: the total number of occurrences
- `*`: appears if there is at least one occurrence which is not currently visible.

14.7.4 Visual Star

With evil-visualstar you can search for the next occurrence of the current selection.

It is pretty useful combined with the expand-region bindings.

Note: If the current state is not the `visual state` then pressing `*` uses auto-highlight-symbol and its transient state.

14.7.5 Listing symbols by semantic

Use `helm-semantic-or-imenu` command from `Helm` to quickly navigate between the symbols in a buffer.

To list all the symbols of a buffer press: `SPC s j`

14.7.6 Helm-swoop

This is very similar to `moccur`, it displays a `helm` buffer with all the occurrences of the word under point. You can then change the search query in real-time and navigate between them easily.

You can even edit the occurrences directly in the `helm` buffer and apply the modifications to the buffer.

Key Binding	Description
`SPC s s`	execute `helm-swoop`
`SPC s S`	execute `helm-multi-swoop`
`SPC s C-s`	execute `helm-multi-swoop-all`

14.8 Editing

14.8.1 Paste text

1. Paste Transient-state The paste transient state can be enabled by settings the variable `dotspacemacs-enable-paste-transient-state` to `t`. By default it is disabled.

   When the transient state is enabled, pressing `p` again will replace the pasted text with the previous yanked (copied) text on the kill ring.

For example if you copy `foo` and `bar` then press `p` the text `bar` will be pasted, pressing `p` again will replace `bar` with `foo`.

Key Binding	Description
`p` or `P`	paste the text before or after point and initiate the `paste` transient state
`C-j`	in transient state: replace paste text with the previously copied one
`C-k`	in transient state: replace paste text with the next copied one
Any other key	leave the transient state

2. Auto-indent pasted text By default any pasted text will be auto-indented. To paste text un-indented use the universal argument.

   It is possible to disable the auto-indentation for specific major-modes by adding a major-mode to the variable `spacemacs-indent-sensitive-modes` in your `dotspacemacs/user-config` function.

14.8.2 Text manipulation commands

Text related commands (start with `x`):

Key Binding	Description
`SPC x a &`	align region at &
`SPC x a (`	align region at (
`SPC x a )`	align region at)
`SPC x a ,`	align region at ,
`SPC x a .`	align region at . (for numeric tables)
`SPC x a :`	align region at :
`SPC x a ;`	align region at ;
`SPC x a =`	align region at =
`SPC x a a`	align region (or guessed section) using default rules
`SPC x a c`	align current indentation region using default rules
`SPC x a r`	align region using user-specified regexp
`SPC x a m`	align region at arithmetic operators (+-*/)
`SPC x a ¦`	align region at ¦
`SPC x c`	count the number of chars/words/lines in the selection region
`SPC x d w`	delete trailing whitespaces
`SPC x g l`	set languages used by translate commands
`SPC x g t`	translate current word using Google Translate
`SPC x g T`	reverse source and target languages
`SPC x j c`	set the justification to center
`SPC x j f`	set the justification to full
`SPC x j l`	set the justification to left
`SPC x j n`	set the justification to none
`SPC x j r`	set the justification to right
`SPC x J`	move down a line of text (enter transient state)
`SPC x K`	move up a line of text (enter transient state)
`SPC x l d`	duplicate line or region
`SPC x l s`	sort lines
`SPC x l u`	uniquify lines
`SPC x o`	use avy to select a link in the frame and open it
`SPC x O`	use avy to select multiple links in the frame and open them
`SPC x t c`	swap (transpose) the current character with the previous one
`SPC x t w`	swap (transpose) the current word with the previous one
`SPC x t l`	swap (transpose) the current line with the previous one
`SPC x u`	set the selected text to lower case
`SPC x U`	set the selected text to upper case
`SPC x w c`	count the number of occurrences per word in the select region
`SPC x w d`	show dictionary entry of word from wordnik.com
`SPC x TAB`	indent or dedent a region rigidly

14.8.3 Text insertion commands

Text insertion commands (start with `i`):

Key binding	Description
`SPC i l l`	insert lorem-ipsum list
`SPC i l p`	insert lorem-ipsum paragraph
`SPC i l s`	insert lorem-ipsum sentence
`SPC i u`	Search for Unicode characters and insert them into the active buffer.
`SPC i U 1`	insert UUIDv1 (use universal argument to insert with CID format)
`SPC i U 4`	insert UUIDv4 (use universal argument to insert with CID format)
`SPC i U U`	insert UUIDv4 (use universal argument to insert with CID format)

14.8.4 Smartparens Strict mode

Smartparens comes with a strict mode which prevents deletion of parenthesis if the result is unbalanced.

This mode can be frustrating for novices, this is why it is not enabled by default.

It is possible to enable it easily for *all programming modes* with the variable `dotspacemacs-smartparens-strict-mode` of you `~/.spacemacs`.

```
(setq-default dotspacemacs-smartparens-strict-mode t)
```

14.8.5 Zooming

1. Text The font size of the current buffer can be adjusted with the commands:

Key Binding	Description
`SPC z x +`	scale up the font and initiate the font scaling transient state
`SPC z x =`	scale up the font and initiate the font scaling transient state
`SPC z x -`	scale down the font and initiate the font scaling transient state
`SPC z x 0`	reset the font size (no scaling) and initiate the font scaling transient state
`+`	increase the font size
`=`	increase the font size
`-`	decrease the font size
`0`	reset the font size
Any other key	leave the font scaling transient state

Note that *only* the text of the current buffer is scaled, the other buffers, the mode-line and the minibuffer are not affected. To zoom the whole content of a frame use the `zoom frame` bindings (see next section).

2. Frame You can zoom in and out the whole content of the frame with the commands:

Key Binding	Description
`SPC z f +`	zoom in the frame content and initiate the frame scaling transient state
`SPC z f =`	zoom in the frame content and initiate the frame scaling transient state
`SPC z f -`	zoom out the frame content and initiate the frame scaling transient state
`SPC z f 0`	reset the frame content size and initiate the frame scaling transient state
`+`	zoom in
`=`	zoom in
`-`	zoom out
`0`	reset zoom
Any other key	leave the zoom frame transient state

14.8.6 Increase/Decrease numbers

Spacemacs uses evil-numbers to easily increase or decrease numbers.

Key Binding	Description
`SPC n +`	increase the number under point by one and initiate transient state
`SPC n -`	decrease the number under point by one and initiate transient state

In transient state:

Key Binding	Description
`+`	increase the number under point by one
`-`	decrease the number under point by one
Any other key	leave the transient state

Tips: you can increase or decrease a value by more that once by using a prefix argument (i.e. `10 SPC n +` will add 10 to the number under point).

14.8.7 Spell checking

Spell checking is enabled by including the spell checking layer in your dotfile.

Keybindings are listed in the layer documentation.

14.8.8 Region selection

Vi `Visual` modes are all supported by `evil`.

1. Expand-region Spacemacs adds another `Visual` mode via the expand-region mode.

Key Binding	Description
`SPC v`	initiate expand-region mode then...
`v`	expand the region by one semantic unit
`V`	contract the region by one semantic unit
`r`	reset the region to initial selection
`ESC`	leave expand-region mode

2. Indent text object With evil-indent-plus the following text objects are available:

   - `ii` - Inner Indentation: the surrounding textblock with the same indentation
   - `iI` - Above and Indentation: `ii` + the line above with a different indentation
   - `iJ` - Above, Below and Indentation+: `iI` + the line below with a different indentation

   There are also `a` variants that include whitespace. Example (`|` indicates point):

   ```
   (while (not done)
     (messa|ge "All work and no play makes Jack a dull boy."))
   (1+ 41)
   ```

   - `vii` will select the line with message
   - `viI` will select the whole while loop
   - `viJ` will select the whole fragment

14.8.9 Region narrowing

The displayed text of a buffer can be narrowed with the commands (start with `n`):

Key Binding	Description
`SPC n f`	narrow the buffer to the current function
`SPC n p`	narrow the buffer to the visible page
`SPC n r`	narrow the buffer to the selected text
`SPC n w`	widen, i.e. show the whole buffer again

14.8.10 Replacing text with iedit

Spacemacs uses the powerful iedit mode through evil-iedit-state to quickly edit multiple occurrences of a symbol or selection.

`evil-iedit-state` defines two new evil states:

- `iedit state`
- `iedit-insert state`

The color code for these states is `red`.

`evil-iedit-state` has also a nice integration with expand-region for quick editing of the currently selected text by pressing `e`.

1. iedit states key bindings

   (a) State transitions

Key Binding	From	To
`SPC s e`	normal or visual	iedit
`e`	expand-region	iedit
`ESC`	iedit	normal
`C-g`	iedit	normal
`fd`	iedit	normal
`ESC`	iedit-insert	iedit
`C-g`	iedit-insert	normal
`fd`	iedit-insert	normal

   To sum-up, in `iedit-insert state` you have to press ESC twice to go back to the `normal state`. You can also at any time press `C-g` or `fd` to return to `normal state`.

   Note: evil commands which switch to `insert state` will switch in `iedit-insert state`.

   (b) In iedit state `iedit state` inherits from `normal state`, the following key bindings are specific to `iedit state`.

Key Binding	Description
`ESC`	go back to `normal state`
`TAB`	toggle current occurrence
`0`	go to the beginning of the current occurrence
`$`	go to the end of the current occurrence
`#`	prefix all occurrences with an increasing number (SPC u to choose the star
`A`	go to the end of the current occurrence and switch to `iedit-insert state`
`D`	delete the occurrences
`F`	restrict the scope to the function
`gg`	go to first occurrence
`G`	go to last occurrence
`I`	go to the beginning of the current occurrence and switch to `iedit-insert`
`J`	increase the editing scope by one line below
`K`	increase the editing scope by one line above
`L`	restrict the scope to the current line
`n`	go to next occurrence
`N`	go to previous occurrence
`p`	replace occurrences with last yanked (copied) text
`S`	(substitute) delete the occurrences and switch to `iedit-insert state`
`V`	toggle visibility of lines with no occurrence
`U`	Up-case the occurrences
`C-U`	down-case the occurrences

Note: `0`, `$`, `A` and `I` have the default Vim behavior when used outside of an `occurrence`.

(c) In iedit-insert state

Key Binding	Description
`ESC`	go back to `iedit state`
`C-g`	go back to `normal state`

2. Examples

- manual selection of several words then replace: `v w w SPC s e S "toto" ESC ESC`
- append text to a word on two lines: `v i w SPC s e J i "toto" ESC ESC`
- substitute symbol *with expand-region*: `SPC v v e S "toto" ESC ESC`

- replace symbol with yanked (copied) text *with expand region*: `SPC v e p ESC ESC`

14.8.11 Replacing text in several files

If you have `ag`, `pt` or `ack` installed, replacing an occurrence of text in several files can be performed via helm-ag.

Say you want to replace all `foo` occurrences by `bar` in your current project:

- initiate a search with `SPC /`
- enter in edit mode with `C-c C-e`
- go to the occurrence and enter in `iedit state` with `SPC s e`
- edit the occurrences then leave the `iedit state`
- press `C-c C-c`

Note: In Spacemacs, `helm-ag` despite its name works with `ack` and `pt` as well (but not with `grep`).

14.8.12 Renaming files in a directory

It is possible to batch rename files in a directory using `wdired` from an `helm` session:

- browse for a directory using `SPC f f`
- enter `wdired` with `C-c C-e`
- edit the file names and use `C-c C-c` to confirm the changes
- use `C-c C-k` to abort any changes

14.8.13 Commenting

Comments are handled by evil-nerd-commenter, it's bound to the following keys.

Key Binding	Description
`SPC ;`	comment operator
`SPC c h`	hide/show comments
`SPC c l`	comment lines
`SPC c L`	invert comment lines
`SPC c p`	comment paragraphs
`SPC c P`	invert comment paragraphs
`SPC c t`	comment to line
`SPC c T`	invert comment to line
`SPC c y`	comment and yank
`SPC c Y`	invert comment and yank

Tips: To comment efficiently a block of line use the combo `SPC ; SPC j l`

14.8.14 Regular expressions

Spacemacs uses the packages pcre2el to manipulate regular expressions. It is useful when working with `Emacs Lisp` buffers since it allows to easily converts `PCRE` (Perl Compatible RegExp) to Emacs RegExp or `rx`. It can also be used to "explain" a PCRE RegExp around point in `rx` form.

The key bindings start with `SPC x r` and have the following mnemonic structure:

- `SPC x r <source> <target>` convert from source to target
- `SPC x r` do what I mean

Key Binding	Function
`SPC x r /`	Explain the regexp around point with `rx`
`SPC x r '`	Generate strings given by a regexp given this list is finite
`SPC x r t`	Replace regexp around point by the `rx` form or vice versa
`SPC x r x`	Convert regexp around point in `rx` form and display the result in the minibuffer
`SPC x r c`	Convert regexp around point to the other form and display the result in the minibuffe
`SPC x r e /`	Explain Emacs Lisp regexp
`SPC x r e '`	Generate strings from Emacs Lisp regexp
`SPC x r e p`	Convert Emacs Lisp regexp to PCRE
`SPC x r e t`	Replace Emacs Lisp regexp by `rx` form or vice versa
`SPC x r e x`	Convert Emacs Lisp regexp to `rx` form
`SPC x r p /`	Explain PCRE regexp
`SPC x r p '`	Generate strings from PCRE regexp
`SPC x r p e`	Convert PCRE regexp to Emacs Lisp
`SPC x r p x`	Convert PCRE to `rx` form

14.8.15 Deleting files

Deletion is configured to send deleted files to system trash.

On OS X the `trash` program is required. It can be installed with homebrew with the following command:

```
$ brew install trash
```

To disable the trash you can set the variable `delete-by-moving-to-trash` to `nil` in your `~/.spacemacs`.

14.8.16 Editing Lisp code

Editing of lisp code is provided by evil-lisp-state.

Commands will set the current state to `lisp state` where different commands combo can be repeated without pressing on `SPC k`.

When in `lisp state` the color of the mode-line changes to pink.

Examples:

- to slurp three times while in normal state: `SPC k 3 s`
- to wrap a symbol in parentheses then slurp two times: `SPC k w 2 s`

Note: The `lisp state` commands are available in *any* modes! Try it out.

1. Lisp Key Bindings

(a) Lisp state key bindings These commands automatically switch to `lisp state`.

Key Binding	Function
`SPC k %`	evil jump item
`SPC k :`	ex command
`SPC k (`	insert expression before (same level as current one)
`SPC k )`	insert expression after (same level as current one)
`SPC k $`	go to the end of current sexp
`SPC k ‘ k`	hybrid version of push sexp (can be used in non lisp dialects)
`SPC k ‘ p`	hybrid version of push sexp (can be used in non lisp dialects)
`SPC k ‘ s`	hybrid version of slurp sexp (can be used in non lisp dialects)
`SPC k ‘ t`	hybrid version of transpose sexp (can be used in non lisp dialects)
`SPC k 0`	go to the beginning of current sexp
`SPC k a`	absorb expression
`SPC k b`	forward barf expression
`SPC k B`	backward barf expression
`SPC k c`	convolute expression
`SPC k ds`	delete symbol
`SPC k Ds`	backward delete symbol
`SPC k dw`	delete word
`SPC k Dw`	backward delete word
`SPC k dx`	delete expression
`SPC k Dx`	backward delete expression
`SPC k e`	unwrap current expression and kill all symbols after point
`SPC k E`	unwrap current expression and kill all symbols before point
`SPC k h`	previous symbol
`SPC k H`	go to previous sexp
`SPC k i`	switch to `insert state`
`SPC k I`	go to beginning of current expression and switch to `insert state`
`SPC k j`	next closing parenthesis
`SPC k J`	join expression
`SPC k k`	previous opening parenthesis
`SPC k l`	next symbol
`SPC k L`	go to next sexp
`SPC k p`	paste after
`SPC k P`	paste before
`SPC k r`	raise expression (replace parent expression by current one)
`SPC k s`	forward slurp expression
`SPC k S`	backward slurp expression
`SPC k t`	transpose expression
`SPC k u`	undo
`SPC k U`	got to parent sexp backward
`SPC k C-r`	redo
`SPC k v`	switch to `visual state`
`SPC k V`	switch to `visual line state`
`SPC k C-v`	switch to `visual block state`
`SPC k w`	wrap expression with parenthesis
`SPC k W`	unwrap expression
`SPC k y`	copy expression

(b) Emacs lisp specific key bindings

Key Binding	Function
`SPC m e $`	go to end of line and evaluate last sexp
`SPC m e b`	evaluate buffer
`SPC m e c`	evaluate current form (a `def` or a `set`)
`SPC m e e`	evaluate last sexp
`SPC m e f`	evaluate current defun
`SPC m e l`	go to end of line and evaluate last sexp
`SPC m e r`	evaluate region

Key Binding	Function
`SPC m g g`	go to definition
`SPC m g G`	go to definition in another window
`SPC m h h`	describe elisp thing at point (show documentation)
`SPC m t b`	execute buffer tests
`SPC m t q`	ask for test function to execute

14.8.17 Mouse usage

There are some added mouse features set for the line number margin (if shown):

- single click in line number margin visually selects the entire line
- drag across line number margin visually selects the region
- double click in line number margin visually select the current code block

14.9 Managing projects

Projects in Spacemacs are managed with projectile. In `projectile` projects are defined implicitly, for instance the root of a project is found when a `.git` repository or `.projectile` file is encountered in the file tree.

`Helm` is used whenever it is possible.

To search in a project see project searching.

`projectile` commands start with p:

Key Binding	Description
`SPC p '`	open a shell in project's root (with the `shell` layer)
`SPC p !`	run shell command in project's root
`SPC p &`	run async shell command in project's root
`SPC p %`	replace a regexp
`SPC p a`	toggle between implementation and test
`SPC p b`	switch to project buffer
`SPC p c`	compile project using `projectile`
`SPC p d`	find directory
`SPC p D`	open project root in `dired`
`SPC p f`	find file
`SPC p F`	find file based on path around point
`SPC p g`	find tags
`SPC p G`	regenerate the project's `etags` / `gtags`
`SPC p h`	find file using `helm`
`SPC p I`	invalidate the projectile cache
`SPC p k`	kill all project buffers
`SPC p o`	run `multi-occur`
`SPC p p`	switch project
`SPC p r`	open a recent file
`SPC p R`	replace a string
`SPC p t`	open `NeoTree` in `projectile` root
`SPC p T`	test project
`SPC p v`	open project root in `vc-dir` or `magit`
`SPC /`	search in project with the best search tool available
`SPC s p`	see searching in a project
`SPC s a p`	run `ag`
`SPC s g p`	run `grep`
`SPC s k p`	run `ack`
`SPC s t p`	run `pt`

Note for Windows Users: To enable fast indexing the GNU `find` or Cygwin `find` must be in your `PATH`.

14.10 Registers

Access commands to the various registers start with `r`:

Key Binding	Description
`SPC r e`	show evil yank and named registers
`SPC r m`	show marks register
`SPC r r`	show helm register
`SPC r y`	show kill ring

14.11 Errors handling

Spacemacs uses Flycheck to gives error feedback on the fly. The checks are only performed at save time by default.

Errors management commands (start with `e`):

Key Binding	Description
`SPC t s`	toggle flycheck
`SPC e c`	clear all errors
`SPC e h`	describe a flycheck checker
`SPC e l`	toggle the display of the `flycheck` list of errors/warnings
`SPC e n`	go to the next error
`SPC e p`	go to the previous error
`SPC e v`	verify flycheck setup (useful to debug 3rd party tools configuration)
`SPC e .`	error transient state

The next/previous error bindings and the error transient state can be used to browse errors from flycheck as well as errors from compilation buffers, and indeed anything that supports Emacs' `next-error` API. This includes for example search results that have been saved to a separate buffer.

Custom fringe bitmaps:

Symbol	Description
🟥	Error
🟨	

14.12 Compiling

Spacemacs binds a few commands to support compiling a project.

Key Binding	Description
`SPC c c`	use `helm-make` via projectile
`SPC c C`	compile
`SPC c d`	close compilation window
`SPC c k`	kill compilation
`SPC c m`	`helm-make`
`SPC c r`	recompile

14.13 Modes

14.13.1 Major Mode leader key

Key bindings specific to the current `major mode` start with `SPC m`. For convenience a shortcut key called the major mode leader key is set by default on `,` which saves one precious keystroke.

It is possible to change the major mode leader key by defining the variable `dotspacemacs-major-mode-leader-key` in your `~/.spacemacs`. For example to setup the key on tabulation:

```
(setq-default dotspacemacs-major-mode-leader-key "<tab>")
```

14.13.2 Helm

Spacemacs add `hjkl` navigation to `helm` buffers:

Key Binding	Description
`C-h`	go to next source
`C-H`	describe key (replace `C-h`)
`C-j`	go to previous candidate
`C-k`	go to next candidate
`C-l`	same as `return`

14.14 Emacs Server

Spacemacs starts a server at launch. This server is killed whenever you close your Emacs windows.

14.14.1 Connecting to the Emacs server

You can open a file in Emacs from the terminal using `emacsclient`. Use `emacsclient -c` to open the file in Emacs GUI. Use `emacsclient -t` to open the file in Emacs within the terminal.

If you want your Linux/OS X system to use Emacs by default for any prompt, you need to set it in your shell configuration, e.g. `~/.bashrc` or `~/.zshrc`:

```
export EDITOR="emacsclient -c"
```

Note that if you're on OS X, you may have to refer to the emacsclient that comes with your GUI Emacs, e.g.:

```
export EDITOR="/Applications/Emacs.app/Contents/MacOS/bin/emacsclient -c"
```

Tip: Remember to use `:wq` or `C-x #` after you are done editing the file in Emacs.

See Emacs as a Server in the official Emacs manual for more details.

14.15 Keeping the server alive

It is possible to keep the server alive when you close Emacs by setting the variable `dotspacemacs-persistent-server` to `t` in your `~./spacemacs`.

```
(setq-default dotspacemacs-persistent-server t)
```

When this variable is set to `t`, the only way to quit Emacs *and* kill the server is to use the following bindings:

Keybinding	Description
`SPC q q`	Quit Emacs and kill the server, prompt for changed buffers to save
`SPC q Q`	Quit Emacs and kill the server, lose all unsaved changes.
`SPC q r`	Restart both Emacs and the server, prompting to save any changed buffers
`SPC q s`	Save the buffers, quit Emacs and kill the server
`SPC q z`	Kill the current frame

14.16 Troubleshoot

14.16.1 Loading fails

If any errors happen during the loading the mode-line will turn red and the errors should appear inline in the startup buffer. Spacemacs should still be usable; if it is not then restart Emacs with `emacs --debug-init` and open a Github issue with the backtrace.

14.16.2 Upgrading/Downgrading Emacs version

To ensure that packages are correctly compiled for the new Emacs version you installed, be sure to run the interactive command `spacemacs/recompile-elpa` with `SPC SPC spacemacs/recompile-elpa`.

15 Achievements

15.1 Issues

Achievements	Account
100th issue (PR)	danielwuz
200th issue (question)	justrajdeep
300th issue (PR)	danielwuz
400th issue (PR)	CestDiego
500th issue (PR)	bjarkevad
600th issue (PR)	bjarkevad
700th issue (enhancement)	jcpetkovich
800th issue (PR)	ryansroberts
900th issue (PR)	jcpetkovich
1000th issue (PR)	tuhdo
2000th issue (PR)	IvanMalison
3000th issue (issue)	malchmih
4000th issue (issue)	icymist
5000th issue (issue)	justbur
6000th issue (issue)	d12frosted
7000th issue (issue)	deb0ch

15.2 Merged Pull Requests

Achievements	Account
100th pull request	bru
200th pull request	smt
300th pull request	BrianHicks
400th pull request	cpaulik
500th pull request	tuhdo
600th pull request	trishume
1000th pull request	justbur
2000th pull request	channingwalton
3000th pull request	darkfeline

15.3 Stars, forks and watchers

Achievements	Account
100th watcher	adouzzy
100th fork	balajisivaraman
200th fork	alcol80
300th fork	mlopes
2000th fork	Gameguykiler
100th star	Jackneill
200th star	jb55
400th star	dbohdan
600th star	laat
700th star	kendall
800th star	urso
900th star	luisgerhorst
1000th star!	rashly
2000th star!!	stshine
3000th star!!!	TheBB
4000th star!!!!	nixmaniack
5000th star!!!!!	StreakyCobra
6000th star!!!!!!	NJBS
7000th star!!!!!!!	mukhali
8000th star!!!!!!!!	shsteven
9000th star!!!!!!!!!	deb0ch
10000th star :star:	colt365

15.4 Gitter chat

Achievements	Account
First joiner on the Gitter Chat	trishume
1000th joiner	gabrielpoca

15.5 First times

Achievements	Account
First contribution	trishume
First contribution layer	trishume
First blog article on Spacemacs	Wolfy87
First contributed banner	chrisbarrett

15.6 Special Mentions

Reason	Account
Autumnal Cleanup 2015 (wiki)	StreakyCobra
Test and debug tools	justbur
Integration of Ivy	justbur
Transient States	justbur
Integration of Persp-mode	CestDiego
Cleanest PR (PR #5545)	JAremko
Documentation tools and GitHub support	JAremko
Code navigation improvement (jump handlers, generalized next error)	TheBB
Better support for GUI using an Emacs daemon (after-display macro)	travisbhartwell

15.7 Special Titles

Achievements	Account
The Gunner (18 PRs in a row)	ralesi
The Saint (unlocked the holy-mode)	trishume
The Artist (logo and theme)	nashamri
The Meme Master (doge banner)	chrisbarrett
The Helm captain (helm guide)	tuhdo
The Master of the Keys (which-key and bind-map)	justbur
The PR Patrol Officer	robbyoconnor
The Expert in Latin Language (PR)	vijaykiran
The Tiler (eyebrowse integration)	bmag
The Geometer (spaceline)	TheBB
The Librarian (doc-fmt tool and space-doc mode)	JAremko

16 Thank you

Thank you Richard for this great piece of software.

Thank you to all the contributors and the whole Emacs community from core developers to elisp hackers!