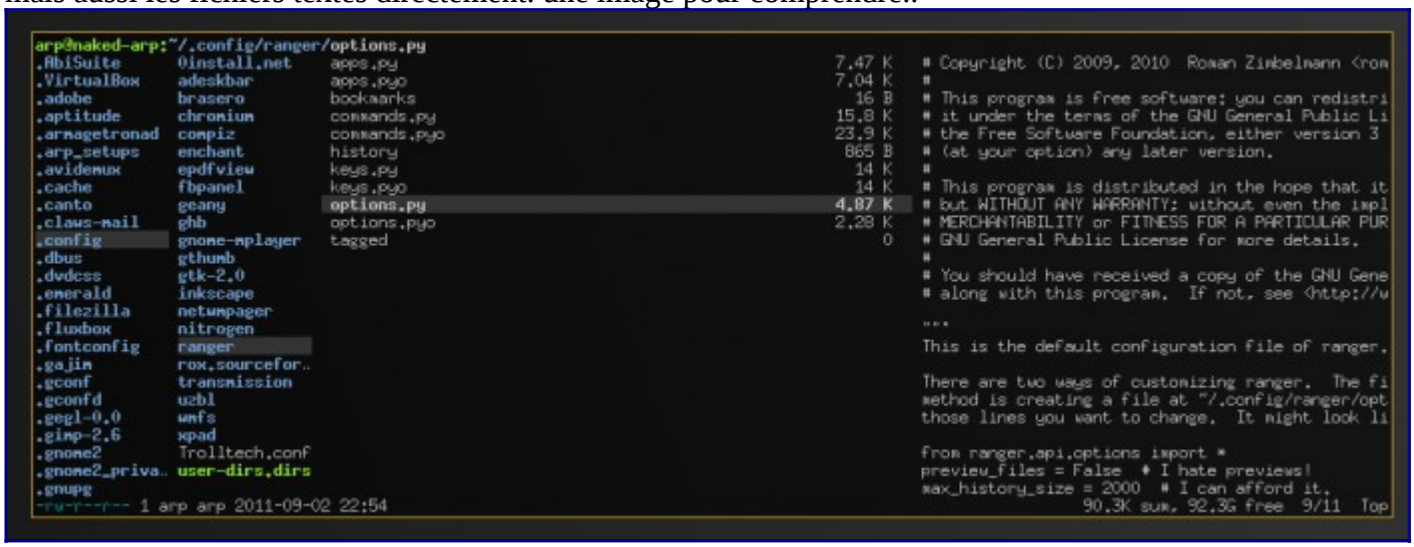
Ranger: le file-manager en python

ranger est un navigateur de fichier écrit en **python** adoptant les raccourcis clavier de **vim**. très léger, il n'en est pas moins hautement configurable: definitions des raccourcis, des applications associées aux types de fichiers, utilisation de scripts externes…
le *plus* de ranger est qu'il propose une organisation en colonnes qui permet de visualiser l'arborescence des dossiers mais aussi les fichiers textes directement. une image pour comprendre..



installation

ranger est présent dans la plupart des dépôts, mais vu sa fréquence de mise à jour, je vous conseille d'aller chercher ranger directement sur son site: les versions stable, testing et git sont disponibles. la dernière version stable est la 1.5.0 datant du 11 oct 2011.
note: les procédures d'installation décrites ci-dessous ont été réalisées avec une Debian Squeeze et python2.6 . cependant ranger est compatible avec python3.x.
au cas ou le serveur de ranger ne soit pas disponible, j'ai mis une archive de la version stable à disposition sur mon serveur ici

- installation de la **version stable** et récupération des fichiers de config:

```
$ wget http://nongnu.org/ranger/ranger-stable.tar.gz
$ tar xvf ranger-stable.tar.gz
$ cd ranger-1.5.0/
$ make
# make install
$ make clean
$ mkdir ~/.config/ranger
$ cp /usr/local/lib/python2.6/dist-packages/ranger/defaults/* /home/arp/.config/ranger/
$ cd
$ ranger
```

- l'installation de la **version testing** se fait de la même façon excepté les premières lignes:

```
$ wget http://git.savannah.gnu.org/cgit/ranger.git/snapshot/ranger-master.tar.gz
$ tar xvf ranger-master.tar.gz
$ cd ranger-master/
...
```

- l'installation de la **version git**: elle propose d'avoir les dernières sources et de pouvoir vérifier les mises à jour directement depuis le dossier ranger grâce à la commande *git pull*.

```
$ git clone git://git.savannah.nongnu.org/ranger.git
$ cd ranger/
...
```

configuration

ranger dispose de différents fichiers de configuration dans ~/.config/ranger que nous allons detailler.

- **ranger/apps.py**: pour déterminer avec quelles applications seront ouverts vos fichiers (choix par extension de fichier).

```
# -*- coding: utf-8 -*-
# Copyright (C) 2009, 2010, 2011  Roman Zimbelmann <romanz@lavabit.com>
# This configuration file is licensed under the same terms as ranger.
# ===================================================================
# This is the configuration file for file type detection and application
# handling.  It's all in python; lines beginning with # are comments.
#
# You can customize this in the file ~/.config/ranger/apps.py.
# It has the same syntax as this file.  In fact, you can just copy this
# file there with `ranger --copy-config=apps' and make your modifications.
# But make sure you update your configs when you update ranger.
#
# In order to add application definitions "on top of" the default ones
# in your ~/.config/ranger/apps.py, you should subclass the class defined
# here like this:
#
# from ranger.defaults.apps import CustomApplications as DefaultApps
# class CustomApplications(DeafultApps):
#     <your definitions here>
#
# ===================================================================
# This system is based on things called MODES and FLAGS.  You can read
# in the man page about them.  To remind you, here's a list of all flags.
# An uppercase flag inverts previous flags of the same name.
#     s   Silent mode.  Output will be discarded.
#     d   Detach the process.  (Run in background)
#     p   Redirect output to the pager
#     w   Wait for an Enter-press when the process is done
#     c   Run the current file only, instead of the selection
#
# To implement flags in this file, you could do this:
#     context.flags += "d"
# Another example:
#     context.flags += "Dw"
#
# To implement modes in this file, you can do something like:
#     if context.mode == 1:
#         <run in one way>
#     elif context.mode == 2:
#         <run in another way>
#
# ===================================================================
# The methods are called with a "context" object which provides some
# attributes that transfer information.  Relevant attributes are:
#
# mode -- a number, mainly used in determining the action in app_xyz()
# flags -- a string with flags which change the way programs are run
# files -- a list containing files, mainly used in app_xyz
# filepaths -- a list of the paths of each file
# file -- an arbitrary file from that list (or None)
# fm -- the filemanager instance
# popen_kws -- keyword arguments which are directly passed to Popen
#
# ===================================================================
# The return value of the functions should be either:
# 1. A reference to another app, like:
#     return self.app_editor(context)
#
# 2. A call to the "either" method, which uses the first program that
# is installed on your system.  If none are installed, None is returned.
#     return self.either(context, "libreoffice", "soffice", "ooffice")
#
```

```
# 3. A tuple of arguments that should be run.
#     return "mplayer", "-fs", context.file.path
# If you use lists instead of strings, they will be flattened:
#     args = ["-fs", "-shuf"]
#     return "mplayer", args, context.filepaths
# "context.filepaths" can, and will often be abbreviated with just "context":
#     return "mplayer", context
#
# 4. "None" to indicate that no action was found.
#     return None
#
# ===================================================================
# When using the "either" method, ranger determines which program to
# pick by looking at its dependencies.  You can set dependencies by
# adding the decorator "depends_on":
#     @depends_on("vim")
#     def app_vim(self, context):
#         ....
# There is a special keyword which you can use as a dependence: "X"
# This ensures that the program will only run when X is running.
# ===================================================================

import ranger
from ranger.api.apps import *
from ranger.ext.get_executables import get_executables

class CustomApplications(Applications):
        def app_default(self, c):
                """How to determine the default application?"""
                f = c.file

                if f.basename.lower() == 'makefile' and c.mode == 1:
                        made = self.either(c, 'make')
                        if made: return made

                if f.extension is not None:
                        if f.extension in ('pdf', ):
                                return self.either(c, 'evince', 'zathura', 'apvlv')
                        if f.extension == 'djvu':
                                return self.either(c, 'evince')
                        if f.extension in ('xml', ):
                                return self.either(c, 'editor')
                        if f.extension in ('html', 'htm', 'xhtml'):
                                return self.either(c, 'firefox', 'opera', 'jumanji',
                                                   'luakit', 'elinks', 'lynx')
                        if f.extension == 'swf':
                                return self.either(c, 'firefox', 'opera', 'jumanji',
'luakit')
                        if f.extension == 'nes':
                                return self.either(c, 'fceux')
                        if f.extension in ('swc', 'smc', 'sfc'):
                                return self.either(c, 'zsnes')
                        if f.extension in ('odt', 'ods', 'odp', 'odf', 'odg',
                                           'doc', 'xls'):
                                return self.either(c, 'libreoffice', 'soffice',
'ooffice')

                if f.mimetype is not None:
                        if INTERPRETED_LANGUAGES.match(f.mimetype):
                                return self.either(c, 'edit_or_run')

                if f.container:
                        return self.either(c, 'aunpack', 'file_roller')

                if f.video or f.audio:
                        if f.video:
                                c.flags += 'd'
                        return self.either(c, 'mplayer2', 'mplayer', 'smplayer', 'vlc',
                                           'totem')
```

```
                if f.image:
                        if c.mode in (11, 12, 13, 14):
                                return self.either(c, 'set_bg_with_feh')
                        else:
                                return self.either(c, 'sxiv', 'feh', 'eog', 'mirage')

                if f.document or f.filetype.startswith('text') or f.size == 0:
                        return self.either(c, 'editor')

                # You can put this at the top of the function and mimeopen will
                # always be used for every file.
                return self.either(c, 'mimeopen')


        # ----------------------------------------- application definitions
        # Note: Trivial application definitions are at the bottom
        def app_pager(self, c):
                return 'less', '-R', c

        def app_editor(self, c):
                try:
                        default_editor = os.environ['EDITOR']
                except KeyError:
                        pass
                else:
                        parts = default_editor.split()
                        exe_name = os.path.basename(parts[0])
                        if exe_name in get_executables():
                                return tuple(parts) + tuple(c)

                return self.either(c, 'vim', 'emacs', 'nano')

        def app_edit_or_run(self, c):
                if c.mode is 1:
                        return self.app_self(c)
                return self.app_editor(c)

        @depends_on('mplayer')
        def app_mplayer(self, c):
                if c.mode is 1:
                        return 'mplayer', '-fs', c

                elif c.mode is 2:
                        args = "mplayer -fs -sid 0 -vfm ffmpeg -lavdopts " \
                                        "lowres=1:fast:skiploopfilter=all:threads=8".spl
it()
                        args.extend(c)
                        return args

                elif c.mode is 3:
                        return 'mplayer', '-mixer', 'software', c

                else:
                        return 'mplayer', c

        @depends_on('mplayer2')
        def app_mplayer2(self, c):
                args = list(self.app_mplayer(c))
                args[0] += '2'
                return args

        # A dependence on "X" means: this programs requires a running X server!
        @depends_on('feh', 'X')
        def app_set_bg_with_feh(self, c):
                c.flags += 'd'
                arg = {11: '--bg-scale', 12: '--bg-tile', 13: '--bg-center',
                                14: '--bg-fill'}
                if c.mode in arg:
```

```
                return 'feh', arg[c.mode], c.file.path
        return 'feh', arg[11], c.file.path

@depends_on('feh', 'X')
def app_feh(self, c):
        c.flags += 'd'
        if c.mode is 0 and len(c.files) is 1: # view all files in the cwd
                images = [f.basename for f in self.fm.env.cwd.files if f.image]
                return 'feh', '--start-at', c.file.basename, images
        return 'feh', c

@depends_on('sxiv', 'X')
def app_sxiv(self, c):
        c.flags = 'd' + c.flags
        if len(c.files) is 1:
                images = [f.basename for f in self.fm.env.cwd.files if f.image]
                try:
                        position = images.index(c.file.basename) + 1
                except:
                        return None
                return 'sxiv', '-n', str(position), images
        return 'sxiv', c

@depends_on('aunpack')
def app_aunpack(self, c):
        if c.mode is 0:
                c.flags += 'p'
                return 'aunpack', '-l', c.file.path
        return 'aunpack', c.file.path

@depends_on('file-roller', 'X')
def app_file_roller(self, c):
        c.flags += 'd'
        return 'file-roller', c.file.path

@depends_on('make')
def app_make(self, c):
        if c.mode is 0:
                return "make"
        if c.mode is 1:
                return "make", "install"
        if c.mode is 2:
                return "make", "clear"

@depends_on('java')
def app_java(self, c):
        def strip_extensions(file):
                if '.' in file.basename:
                        return file.path[:file.path.index('.')]
                return file.path
        files_without_extensions = map(strip_extensions, c.files)
        return "java", files_without_extensions

@depends_on('totem', 'X')
def app_totem(self, c):
        if c.mode is 0:
                return "totem", c
        if c.mode is 1:
                return "totem", "--fullscreen", c

@depends_on('mimeopen')
def app_mimeopen(self, c):
        if c.mode is 0:
                return "mimeopen", c
        if c.mode is 1:
                # Will ask user to select program
                # aka "Open with..."
                return "mimeopen", "--ask", c
```

```
# Often a programs invocation is trivial.  For example:
#    vim test.py readme.txt [...]
#
# This could be implemented like:
#    @depends_on("vim")
#    def app_vim(self, context):
#        return "vim", context
#
# But this is redundant and ranger does this automatically.  However, sometimes
# you want to change some properties like flags or dependencies.
# The method "generic" defines a generic method for the given programs which
# looks like the one above, but you can add dependencies and flags here.
# Add programs (that are not defined yet) here if they should only run in X:
CustomApplications.generic('fceux', 'wine', 'zsnes', deps=['X'])

# Add those which should only run in X AND should be detached/forked here:
CustomApplications.generic('opera', 'firefox', 'apvlv', 'evince',
                'zathura', 'gimp', 'mirage', 'eog', 'jumanji',
                        flags='d', deps=['X'])

# What filetypes are recognized as scripts for interpreted languages?
# This regular expression is used in app_default()
INTERPRETED_LANGUAGES = re.compile(r'''
        ^(text|application)/x-(
                haskell|perl|python|ruby|sh
        )$''', re.VERBOSE)
```

- **ranger/commands.py**: les commandes passées à ranger (:delete, :rename…)

```
# -*- coding: utf-8 -*-
# Copyright (C) 2009, 2010, 2011  Roman Zimbelmann <romanz@lavabit.com>
# This configuration file is licensed under the same terms as ranger.
# ===================================================================
# This file contains ranger's commands.
# It's all in python; lines beginning with # are comments.
#
# Note that additional commands are automatically generated from the methods
# of the class ranger.core.actions.Actions.
#
# You can customize commands in the file ~/.config/ranger/commands.py.
# It has the same syntax as this file.  In fact, you can just copy this
# file there with `ranger --copy-config=commands' and make your modifications.
# But make sure you update your configs when you update ranger.
#
# ===================================================================
# Every class defined here which is a subclass of `Command' will be used as a
# command in ranger.  Several methods are defined to interface with ranger:
#   execute(): called when the command is executed.
#   cancel():  called when closing the console.
#   tab():     called when <TAB> is pressed.
#   quick():   called after each keypress.
#
# The return values for tab() can be either:
#   None: There is no tab completion
#   A string: Change the console to this string
#   A list/tuple/generator: cycle through every item in it
#
# The return value for quick() can be:
#   False: Nothing happens
#   True: Execute the command afterwards
#
# The return value for execute() and cancel() doesn't matter.
#
# ===================================================================
# Commands have certain attributes and methods that facilitate parsing of
# the arguments:
#
# self.line: The whole line that was written in the console.
```

```
# self.args: A list of all (space-separated) arguments to the command.
# self.quantifier: If this command was mapped to the key "X" and
#      the user pressed 6X, self.quantifier will be 6.
# self.arg(n): The n-th argument, or an empty string if it doesn't exist.
# self.rest(n): The n-th argument plus everything that followed.  For example,
#      If the command was "search foo bar a b c", rest(2) will be "bar a b c"
# self.start(n): The n-th argument and anything before it.  For example,
#      If the command was "search foo bar a b c", rest(2) will be "bar a b c"
#
# ========================================================================
# And this is a little reference for common ranger functions and objects:
#
# self.fm: A reference to the "fm" object which contains most information
#      about ranger.
# self.fm.notify(string): Print the given string on the screen.
# self.fm.notify(string, bad=True): Print the given string in RED.
# self.fm.reload_cwd(): Reload the current working directory.
# self.fm.env.cwd: The current working directory. (A File object.)
# self.fm.env.cf: The current file. (A File object too.)
# self.fm.env.cwd.get_selection(): A list of all selected files.
# self.fm.execute_console(string): Execute the string as a ranger command.
# self.fm.open_console(string): Open the console with the given string
#      already typed in for you.
# self.fm.move(direction): Moves the cursor in the given direction, which
#      can be something like down=3, up=5, right=1, left=1, to=6, ...
#
# File objects (for example self.fm.env.cf) have these useful attributes and
# methods:
#
# cf.path: The path to the file.
# cf.basename: The base name only.
# cf.load_content(): Force a loading of the directories content (which
#      obviously works with directories only)
# cf.is_directory: True/False depending on whether it's a directory.
#
# For advanced commands it is unavoidable to dive a bit into the source code
# of ranger.
# ========================================================================

from ranger.api.commands import *
from ranger.ext.get_executables import get_executables
from ranger.core.runner import ALLOWED_FLAGS

class alias(Command):
        """
        :alias <newcommand> <oldcommand>

        Copies the oldcommand as newcommand.
        """
        def execute(self):
                if not self.arg(1) or not self.arg(2):
                        self.fm.notify('Syntax: alias <newcommand> <oldcommand>',
bad=True)
                else:
                        self.fm.commands.alias(self.arg(1), self.arg(2))

class cd(Command):
        """
        :cd [-r] <dirname>

        The cd command changes the directory.
        The command 'cd -' is equivalent to typing ``.
        Using the option "-r" will get you to the real path.
        """

        def execute(self):
                if self.arg(1) == '-r':
                        import os.path
                        self.shift()
```

```
                        destination = os.path.realpath(self.rest(1))
                        if os.path.isfile(destination):
                                destination = os.path.dirname(destination)
                else:
                        destination = self.rest(1)

                if not destination:
                        destination = '~'

                if destination == '-':
                        self.fm.enter_bookmark('`')
                else:
                        self.fm.cd(destination)

        def tab(self):
                from os.path import dirname, basename, expanduser, join

                cwd = self.fm.env.cwd.path
                rel_dest = self.rest(1)

                bookmarks = [v.path for v in self.fm.bookmarks.dct.values()
                                if rel_dest in v.path ]

                # expand the tilde into the user directory
                if rel_dest.startswith('~'):
                        rel_dest = expanduser(rel_dest)

                # define some shortcuts
                abs_dest = join(cwd, rel_dest)
                abs_dirname = dirname(abs_dest)
                rel_basename = basename(rel_dest)
                rel_dirname = dirname(rel_dest)

                try:
                        # are we at the end of a directory?
                        if rel_dest.endswith('/') or rel_dest == '':
                                _, dirnames, _ = next(os.walk(abs_dest))

                        # are we in the middle of the filename?
                        else:
                                _, dirnames, _ = next(os.walk(abs_dirname))
                                dirnames = [dn for dn in dirnames \
                                                if dn.startswith(rel_basename)]
                except (OSError, StopIteration):
                        # os.walk found nothing
                        pass
                else:
                        dirnames.sort()
                        dirnames = bookmarks + dirnames

                        # no results, return None
                        if len(dirnames) == 0:
                                return

                        # one result. since it must be a directory, append a slash.
                        if len(dirnames) == 1:
                                return self.start(1) + join(rel_dirname, dirnames[0]) +
'/'

                        # more than one result. append no slash, so the user can
                        # manually type in the slash to advance into that directory
                        return (self.start(1) + join(rel_dirname, dirname) for dirname
in dirnames)


class chain(Command):
        """
        :chain <command1>; <command2>; ...
        Calls multiple commands at once, separated by semicolons.
```

```
        """
        def execute(self):
                for command in self.rest(1).split(";"):
                        self.fm.execute_console(command)


class search(Command):
        def execute(self):
                self.fm.search_file(self.rest(1), regexp=True)


class search_inc(Command):
        def quick(self):
                self.fm.search_file(self.rest(1), regexp=True, offset=0)


class shell(Command):
        escape_macros_for_shell = True

        def execute(self):
                if self.arg(1) and self.arg(1)[0] == '-':
                        flags = self.arg(1)[1:]
                        command = self.rest(2)
                else:
                        flags = ''
                        command = self.rest(1)

                if not command and 'p' in flags: command = 'cat %f'
                if command:
                        if '%' in command:
                                command = self.fm.substitute_macros(command)
                        self.fm.execute_command(command, flags=flags)

        def tab(self):
                if self.arg(1) and line.arg(1)[0] == '-':
                        command = self.rest(2)
                else:
                        command = self.rest(1)
                start = self.line[0:len(self.line) - len(command)]

                try:
                        position_of_last_space = command.rindex(" ")
                except ValueError:
                        return (start + program + ' ' for program \
                                        in get_executables() if
program.startswith(command))
                if position_of_last_space == len(command) - 1:
                        return self.line + '%s '
                else:
                        before_word, start_of_word = self.line.rsplit(' ', 1)
                        return (before_word + ' ' + file.shell_escaped_basename \
                                        for file in self.fm.env.cwd.files \
                                        if
file.shell_escaped_basename.startswith(start_of_word))

class open_with(Command):
        def execute(self):
                app, flags, mode = self._get_app_flags_mode(self.rest(1))
                self.fm.execute_file(
                                files = [f for f in self.fm.env.cwd.get_selection()],
                                app = app,
                                flags = flags,
                                mode = mode)

        def _get_app_flags_mode(self, string):
                """
                Extracts the application, flags and mode from a string.

                examples:
```

```
                "mplayer d 1" => ("mplayer", "d", 1)
                "aunpack 4" => ("aunpack", "", 4)
                "p" => ("", "p", 0)
                "" => None
                """

                app = ''
                flags = ''
                mode = 0
                split = string.split()

                if len(split) == 0:
                        pass

                elif len(split) == 1:
                        part = split[0]
                        if self._is_app(part):
                                app = part
                        elif self._is_flags(part):
                                flags = part
                        elif self._is_mode(part):
                                mode = part

                elif len(split) == 2:
                        part0 = split[0]
                        part1 = split[1]

                        if self._is_app(part0):
                                app = part0
                                if self._is_flags(part1):
                                        flags = part1
                                elif self._is_mode(part1):
                                        mode = part1
                        elif self._is_flags(part0):
                                flags = part0
                                if self._is_mode(part1):
                                        mode = part1
                        elif self._is_mode(part0):
                                mode = part0
                                if self._is_flags(part1):
                                        flags = part1

                elif len(split) >= 3:
                        part0 = split[0]
                        part1 = split[1]
                        part2 = split[2]

                        if self._is_app(part0):
                                app = part0
                                if self._is_flags(part1):
                                        flags = part1
                                        if self._is_mode(part2):
                                                mode = part2
                                elif self._is_mode(part1):
                                        mode = part1
                                        if self._is_flags(part2):
                                                flags = part2
                        elif self._is_flags(part0):
                                flags = part0
                                if self._is_mode(part1):
                                        mode = part1
                        elif self._is_mode(part0):
                                mode = part0
                                if self._is_flags(part1):
                                        flags = part1

                return app, flags, int(mode)

        def _get_tab(self):
```

```
                data = self.rest(1)
                if ' ' not in data:
                        all_apps = self.fm.apps.all()
                        if all_apps:
                                return (app for app in all_apps if app.startswith(data))

                return None

        def _is_app(self, arg):
                return self.fm.apps.has(arg) or \
                        (not self._is_flags(arg) and arg in get_executables())

        def _is_flags(self, arg):
                return all(x in ALLOWED_FLAGS for x in arg)

        def _is_mode(self, arg):
                return all(x in '0123456789' for x in arg)


class find(Command):
        """
        :find <string>

        The find command will attempt to find a partial, case insensitive
        match in the filenames of the current directory and execute the
        file automatically.
        """

        count = 0
        tab = Command._tab_directory_content

        def execute(self):
                if self.quick():
                        self.fm.move(right=1)
                        self.fm.block_input(0.5)
                else:
                        self.fm.cd(self.rest(1))

        def quick(self):
                self.count = 0
                cwd = self.fm.env.cwd
                arg = self.rest(1)
                if not arg:
                        return False

                if arg == '.':
                        return False
                if arg == '..':
                        return True

                deq = deque(cwd.files)
                deq.rotate(-cwd.pointer)
                i = 0
                case_insensitive = arg.lower() == arg
                for fsobj in deq:
                        if case_insensitive:
                                filename = fsobj.basename_lower
                        else:
                                filename = fsobj.basename
                        if arg in filename:
                                self.count += 1
                                if self.count == 1:
                                        cwd.move(to=(cwd.pointer + i) % len(cwd.files))
                                        self.fm.env.cf = cwd.pointed_obj
                        if self.count > 1:
                                return False
                        i += 1

                return self.count == 1
```

```
class set_(Command):
        """
        :set <option name>=<python expression>

        Gives an option a new value.
        """
        name = 'set'  # don't override the builtin set class
        def execute(self):
                name = self.arg(1)
                name, value, _ = self.parse_setting_line()
                if name and value:
                        from re import compile as regexp
                        try:
                                value = eval(value)
                        except:
                                pass
                        self.fm.settings[name] = value

        def tab(self):
                name, value, name_done = self.parse_setting_line()
                settings = self.fm.settings
                if not name:
                        return (self.firstpart + setting for setting in settings)
                if not value and not name_done:
                        return (self.firstpart + setting for setting in settings \
                                        if setting.startswith(name))
                if not value:
                        return self.firstpart + repr(settings[name])
                if bool in settings.types_of(name):
                        if 'true'.startswith(value.lower()):
                                return self.firstpart + 'True'
                        if 'false'.startswith(value.lower()):
                                return self.firstpart + 'False'


class quit(Command):
        """
        :quit

        Closes the current tab.  If there is only one tab, quit the program.
        """

        def execute(self):
                if len(self.fm.tabs) <= 1:
                        self.fm.exit()
                self.fm.tab_close()


class quitall(Command):
        """
        :quitall

        Quits the program immediately.
        """

        def execute(self):
                self.fm.exit()


class quit_bang(quitall):
        """
        :quit!

        Quits the program immediately.
        """
        name = 'quit!'
        allow_abbrev = False
```

```
class terminal(Command):
        """
        :terminal

        Spawns an "x-terminal-emulator" starting in the current directory.
        """
        def execute(self):
                self.fm.run('x-terminal-emulator', flags='d')


class delete(Command):
        """
        :delete

        Tries to delete the selection.

        "Selection" is defined as all the "marked files" (by default, you
        can mark files with space or v). If there are no marked files,
        use the "current file" (where the cursor is)

        When attempting to delete non-empty directories or multiple
        marked files, it will require a confirmation: The last word in
        the line has to start with a 'y'.  This may look like:
        :delete yes
        :delete seriously? yeah!
        """

        allow_abbrev = False

        def execute(self):
                lastword = self.arg(-1)

                if lastword.startswith('y'):
                        # user confirmed deletion!
                        return self.fm.delete()
                elif self.line.startswith(DELETE_WARNING):
                        # user did not confirm deletion
                        return

                cwd = self.fm.env.cwd
                cf = self.fm.env.cf

                if cwd.marked_items or (cf.is_directory and not cf.is_link \
                                and len(os.listdir(cf.path)) > 0):
                        # better ask for a confirmation, when attempting to
                        # delete multiple files or a non-empty directory.
                        return self.fm.open_console(DELETE_WARNING)

                # no need for a confirmation, just delete
                self.fm.delete()


class mark(Command):
        """
        :mark <regexp>

        Mark all files matching a regular expression.
        """
        do_mark = True

        def execute(self):
                import re
                cwd = self.fm.env.cwd
                input = self.rest(1)
                searchflags = re.UNICODE
                if input.lower() == input: # "smartcase"
                        searchflags |= re.IGNORECASE
```

```
                pattern = re.compile(input, searchflags)
                for fileobj in cwd.files:
                        if pattern.search(fileobj.basename):
                                cwd.mark_item(fileobj, val=self.do_mark)
                self.fm.ui.status.need_redraw = True
                self.fm.ui.need_redraw = True


class console(Command):
        """
        :console <command>

        Open the console with the given command.
        """
        def execute(self):
                position = None
                if self.arg(1)[0:2] == '-p':
                        try:
                                position = int(self.arg(1)[2:])
                                self.shift()
                        except:
                                pass
                self.fm.open_console(self.rest(1), position=position)


class load_copy_buffer(Command):
        """
        :load_copy_buffer

        Load the copy buffer from confdir/copy_buffer
        """
        copy_buffer_filename = 'copy_buffer'
        def execute(self):
                from ranger.fsobject import File
                from os.path import exists
                try:
                        fname = self.fm.confpath(self.copy_buffer_filename)
                        f = open(fname, 'r')
                except:
                        return self.fm.notify("Cannot open %s" % \
                                        (fname or self.copy_buffer_filename), bad=True)
                self.fm.env.copy = set(File(g) \
                        for g in f.read().split("\n") if exists(g))
                f.close()
                self.fm.ui.redraw_main_column()


class save_copy_buffer(Command):
        """
        :save_copy_buffer

        Save the copy buffer to confdir/copy_buffer
        """
        copy_buffer_filename = 'copy_buffer'
        def execute(self):
                fname = None
                try:
                        fname = self.fm.confpath(self.copy_buffer_filename)
                        f = open(fname, 'w')
                except:
                        return self.fm.notify("Cannot open %s" % \
                                        (fname or self.copy_buffer_filename), bad=True)
                f.write("\n".join(f.path for f in self.fm.env.copy))
                f.close()


class unmark(mark):
        """
        :unmark <regexp>
```

```
        Unmark all files matching a regular expression.
        """
        do_mark = False


class mkdir(Command):
        """
        :mkdir <dirname>

        Creates a directory with the name <dirname>.
        """

        def execute(self):
                from os.path import join, expanduser, lexists
                from os import mkdir

                dirname = join(self.fm.env.cwd.path, expanduser(self.rest(1)))
                if not lexists(dirname):
                        mkdir(dirname)
                else:
                        self.fm.notify("file/directory exists!", bad=True)


class touch(Command):
        """
        :touch <fname>

        Creates a file with the name <fname>.
        """

        def execute(self):
                from os.path import join, expanduser, lexists

                fname = join(self.fm.env.cwd.path, expanduser(self.rest(1)))
                if not lexists(fname):
                        open(fname, 'a').close()
                else:
                        self.fm.notify("file/directory exists!", bad=True)


class edit(Command):
        """
        :edit <filename>

        Opens the specified file in vim
        """

        def execute(self):
                if not self.arg(1):
                        self.fm.edit_file(self.fm.env.cf.path)
                else:
                        self.fm.edit_file(self.rest(1))

        def tab(self):
                return self._tab_directory_content()


class eval_(Command):
        """
        :eval [-q] <python code>

        Evaluates the python code.
        `fm' is a reference to the FM instance.
        To display text, use the function `p'.

        Examples:
        :eval fm
        :eval len(fm.env.directories)
```

```
        :eval p("Hello World!")
        """
        name = 'eval'
        resolve_macros = False

        def execute(self):
                if self.arg(1) == '-q':
                        code = self.rest(2)
                        quiet = True
                else:
                        code = self.rest(1)
                        quiet = False
                import ranger
                global cmd, fm, p, quantifier
                fm = self.fm
                cmd = self.fm.execute_console
                p = fm.notify
                quantifier = self.quantifier
                try:
                        try:
                                result = eval(code)
                        except SyntaxError:
                                exec(code)
                        else:
                                if result and not quiet:
                                        p(result)
                except Exception as err:
                        p(err)


class rename(Command):
        """
        :rename <newname>

        Changes the name of the currently highlighted file to <newname>
        """

        def execute(self):
                from ranger.fsobject import File
                from os import access

                new_name = self.rest(1)

                if not new_name:
                        return self.fm.notify('Syntax: rename <newname>', bad=True)

                if new_name == self.fm.env.cf.basename:
                        return

                if access(new_name, os.F_OK):
                        return self.fm.notify("Can't rename: file already exists!",
bad=True)

                self.fm.rename(self.fm.env.cf, new_name)
                f = File(new_name)
                self.fm.env.cwd.pointed_obj = f
                self.fm.env.cf = f

        def tab(self):
                return self._tab_directory_content()


class chmod(Command):
        """
        :chmod <octal number>

        Sets the permissions of the selection to the octal number.

        The octal number is between 0 and 777. The digits specify the
```

```
        permissions for the user, the group and others.

        A 1 permits execution, a 2 permits writing, a 4 permits reading.
        Add those numbers to combine them. So a 7 permits everything.
        """

        def execute(self):
                mode = self.rest(1)
                if not mode:
                        mode = str(self.quantifier)

                try:
                        mode = int(mode, 8)
                        if mode < 0 or mode > 0o777:
                                raise ValueError
                except ValueError:
                        self.fm.notify("Need an octal number between 0 and 777!",
bad=True)
                        return

                for file in self.fm.env.get_selection():
                        try:
                                os.chmod(file.path, mode)
                        except Exception as ex:
                                self.fm.notify(ex)

                try:
                        # reloading directory.  maybe its better to reload the selected
                        # files only.
                        self.fm.env.cwd.load_content()
                except:
                        pass


class bulkrename(Command):
        """
        :bulkrename

        This command opens a list of selected files in an external editor.
        After you edit and save the file, it will generate a shell script
        which does bulk renaming according to the changes you did in the file.

        This shell script is opened in an editor for you to review.
        After you close it, it will be executed.
        """
        def execute(self):
                import sys
                import tempfile
                from ranger.fsobject.file import File
                from ranger.ext.shell_escape import shell_escape as esc
                py3 = sys.version > "3"

                # Create and edit the file list
                filenames = [f.basename for f in self.fm.env.get_selection()]
                listfile = tempfile.NamedTemporaryFile()

                if py3:
                        listfile.write("\n".join(filenames).encode("utf-8"))
                else:
                        listfile.write("\n".join(filenames))
                listfile.flush()
                self.fm.execute_file([File(listfile.name)], app='editor')
                listfile.seek(0)
                if py3:
                        new_filenames = listfile.read().decode("utf-8").split("\n")
                else:
                        new_filenames = listfile.read().split("\n")
                listfile.close()
                if all(a == b for a, b in zip(filenames, new_filenames)):
```

```
                        self.fm.notify("No renaming to be done!")
                        return

                # Generate and execute script
                cmdfile = tempfile.NamedTemporaryFile()
                cmdfile.write(b"# This file will be executed when you close the
editor.\n")
                cmdfile.write(b"# Please double-check everything, clear the file to
abort.\n")
                if py3:
                        cmdfile.write("\n".join("mv -vi " + esc(old) + " " + esc(new) \
                                for old, new in zip(filenames, new_filenames) \
                                if old != new).encode("utf-8"))
                else:
                        cmdfile.write("\n".join("mv -vi " + esc(old) + " " + esc(new) \
                                for old, new in zip(filenames, new_filenames) if old !=
new))
                cmdfile.flush()
                self.fm.execute_file([File(cmdfile.name)], app='editor')
                self.fm.run(['/bin/sh', cmdfile.name], flags='w')
                cmdfile.close()


class help_(Command):
        """
        :help

        Display ranger's manual page.
        """
        name = 'help'
        def execute(self):
                if self.quantifier == 1:
                        self.fm.dump_keybindings()
                elif self.quantifier == 2:
                        self.fm.dump_commands()
                elif self.quantifier == 3:
                        self.fm.dump_settings()
                else:
                        self.fm.display_help()


class copymap(Command):
        """
        :copymap <keys> <newkeys1> [<newkeys2>...]
        Copies a "browser" keybinding from <keys> to <newkeys>
        """
        context = 'browser'

        def execute(self):
                if not self.arg(1) or not self.arg(2):
                        return self.notify("Not enough arguments", bad=True)

                for arg in self.args[2:]:
                        self.fm.env.keymaps.copy(self.context, self.arg(1), arg)


class copypmap(copymap):
        """
        :copypmap <keys> <newkeys1> [<newkeys2>...]
        Copies a "pager" keybinding from <keys> to <newkeys>
        """
        context = 'pager'


class copycmap(copymap):
        """
        :copycmap <keys> <newkeys1> [<newkeys2>...]
        Copies a "console" keybinding from <keys> to <newkeys>
        """
```

```
        context = 'console'


class copytmap(copymap):
        """
        :copycmap <keys> <newkeys1> [<newkeys2>...]
        Copies a "taskview" keybinding from <keys> to <newkeys>
        """
        context = 'taskview'


class unmap(Command):
        """
        :unmap <keys> [<keys2>, ...]
        Remove the given "browser" mappings
        """
        context = 'browser'

        def execute(self):
                for arg in self.args[1:]:
                        self.fm.env.keymaps.unbind(self.context, arg)


class cunmap(unmap):
        """
        :cunmap <keys> [<keys2>, ...]
        Remove the given "console" mappings
        """
        context = 'browser'


class punmap(unmap):
        """
        :punmap <keys> [<keys2>, ...]
        Remove the given "pager" mappings
        """
        context = 'pager'


class tunmap(unmap):
        """
        :tunmap <keys> [<keys2>, ...]
        Remove the given "taskview" mappings
        """
        context = 'taskview'


class map_(Command):
        """
        :map <keysequence> <command>
        Maps a command to a keysequence in the "browser" context.

        Example:
        map j move down
        map J move down 10
        """
        name = 'map'
        context = 'browser'
        resolve_macros = False

        def execute(self):
                self.fm.env.keymaps.bind(self.context, self.arg(1), self.rest(2))


class cmap(map_):
        """:cmap <keysequence> <command>
        Maps a command to a keysequence in the "console" context.

        Example:
```

```
        cmap <ESC> console_close
        cmap <C-x> console_type test
        """
        context = 'console'


class tmap(map_):
        """:tmap <keysequence> <command>
        Maps a command to a keysequence in the "taskview" context.
        """
        context = 'taskview'


class pmap(map_):
        """:pmap <keysequence> <command>
        Maps a command to a keysequence in the "pager" context.
        """
        context = 'pager'


class filter(Command):
        """
        :filter <string>

        Displays only the files which contain <string> in their basename.
        """

        def execute(self):
                self.fm.set_filter(self.rest(1))
                self.fm.reload_cwd()


class grep(Command):
        """
        :grep <string>

        Looks for a string in all marked files or directories
        """

        def execute(self):
                if self.rest(1):
                        action = ['grep', '--color=always', '--line-number']
                        action.extend(['-e', self.rest(1), '-r'])
                        action.extend(f.path for f in self.fm.env.get_selection())
                        self.fm.execute_command(action, flags='p')
```

- **ranger/options.py**: les options de ranger. **note** si vous désirez modifier les options de ranger, il faudra *impérativement* éditer la ligne 21 du fichier 'options.py' et remplacer

```
load_default_rc = True
```

par

```
load_default_rc = False

# -*- coding: utf-8 -*-
# Copyright (C) 2009, 2010, 2011  Roman Zimbelmann <romanz@lavabit.com>
# This configuration file is licensed under the same terms as ranger.
# ===================================================================
# This is the main configuration file of ranger.  It consists of python
# code, but fear not, you don't need any python knowledge for changing
# the settings.
#
# Lines beginning with # are comments.  To enable a line, remove the #.
#
# You can customize ranger in the file ~/.config/ranger/options.py.
# It has the same syntax as this file.  In fact, you can just copy this
# file there with `ranger --copy-config=options' and make your modifications.
# But make sure you update your configs when you update ranger.
```

```
# ===================================================================

from ranger.api.options import *

# Load the deault rc.conf file?  If you've copied it to your configuration
# direcory, then you should deactivate this option.
load_default_rc = True

# How many columns are there, and what are their relative widths?
column_ratios = (1, 3, 4)

# Which files should be hidden?  Toggle this by typing `zh' or
# changing the setting `show_hidden'
hidden_filter = regexp(
        r'^\.|\.(?:pyc|pyo|bak|swp)$|^lost\+found$|^__cache__$')
show_hidden = False

# Which script is used to generate file previews?
# ranger ships with scope.sh, a script that calls external programs (see
# README for dependencies) to preview images, archives, etc.
preview_script = '~/.config/ranger/scope.sh'

# Use that external preview script or display internal plain text previews?
use_preview_script = True

# Use a unicode "..." character to mark cut-off filenames?
unicode_ellipsis = False

# Show dotfiles in the bookmark preview box?
show_hidden_bookmarks = True

# Which colorscheme to use?  These colorschemes are available by default:
# default, default88, jungle, snow
# Snow is monochrome and default88 uses 88 colors.
colorscheme = 'default'

# Preview files on the rightmost column?
# And collapse (shrink) the last column if there is nothing to preview?
preview_files = True
preview_directories = True
collapse_preview = True

# Save the console history on exit?
save_console_history = True

# Draw borders around columns?
draw_borders = False
draw_bookmark_borders = True

# Display the directory name in tabs?
dirname_in_tabs = False

# Enable the mouse support?
mouse_enabled = True

# Display the file size in the main column or status bar?
display_size_in_main_column = True
display_size_in_status_bar = False

# Display files tags in all columns or only in main column?
display_tags_in_all_columns = True

# Set a title for the window?
update_title = True

# Shorten the title if it gets long?  The number defines how many
# directories are displayed at once, False turns off this feature.
shorten_title = 3
```

```
# Abbreviate $HOME with ~ in the titlebar (first line) of ranger?
tilde_in_titlebar = True

# How many directory-changes or console-commands should be kept in history?
max_history_size = 20
max_console_history_size = 50

# Try to keep so much space between the top/bottom border when scrolling:
scroll_offset = 8

# Flush the input after each key hit?  (Noticable when ranger lags)
flushinput = True

# Padding on the right when there's no preview?
# This allows you to click into the space to run the file.
padding_right = True

# Save bookmarks (used with mX and `X) instantly?
# This helps to synchronize bookmarks between multiple ranger
# instances but leads to *slight* performance loss.
# When false, bookmarks are saved when ranger is exited.
autosave_bookmarks = True

# Makes sense for screen readers:
show_cursor = False

# One of: size, basename, mtime, type
sort = 'natural'
sort_reverse = False
sort_case_insensitive = False
sort_directories_first = True

# Enable this if key combinations with the Alt Key don't work for you.
# (Especially on xterm)
xterm_alt_key = False

# The color scheme overlay.  Explained below.
colorscheme_overlay = None

## Apply an overlay function to the colorscheme.  It will be called with
## 4 arguments: the context and the 3 values (fg, bg, attr) returned by
## the original use() function of your colorscheme.  The return value
## must be a 3-tuple of (fg, bg, attr).
## Note: Here, the colors/attributes aren't directly imported into
## the namespace but have to be accessed with color.xyz.

#from ranger.gui import color
#def colorscheme_overlay(context, fg, bg, attr):
#       if context.directory and attr & color.bold and \
#                       not any((context.marked, context.selected)):
#               attr ^= color.bold  # I don't like bold directories!
#
#       if context.main_column and context.selected:
#               fg, bg = color.red, color.default  # To highlight the main column!
#
#       return fg, bg, attr


# ===================================================================
# Beware: from here on, you are on your own.  This part requires python
# knowledge.
#
# Since python is a dynamic language, it gives you the power to replace any
# part of ranger without touching the code.  This is commonly referred to as
# Monkey Patching and can be helpful if you, for some reason, don't want to
# modify rangers code directly.  Just remember: the more you mess around, the
# more likely it is to break when you switch to another version.
#
# Here are some practical examples of monkey patching.
```

```
#
# Technical information:  This file is imported as a python module.  If a
# variable has the name of a setting, ranger will attempt to use it to change
# that setting.  You can write "del <variable-name>" to avoid that.
# ===================================================================
# Add a new sorting algorithm: Random sort.
# Enable this with :set sort=random

#from ranger.fsobject.directory import Directory
#from random import random
#Directory.sort_dict['random'] = lambda path: random()

# ===================================================================
# A function that changes which files are displayed.  This is more powerful
# than the hidden_filter setting since this function has more information.

## Save the original filter function
#import ranger.fsobject.directory
#old_accept_file = ranger.fsobject.directory.accept_file
#
## Define a new one
#def accept_file_MOD(fname, mypath, hidden_filter, name_filter):
#       if hidden_filter and mypath == '/' and fname in ('boot', 'sbin', 'proc', 'sys'):
#               return False
#       else:
#               return old_accept_file(fname, mypath, hidden_filter, name_filter)
#
## Overwrite the old function
#import ranger.fsobject.directory
#ranger.fsobject.directory.accept_file = accept_file_MOD

# ===================================================================
# A function that adds an additional macro.  Test this with :shell -p echo %date

## Save the original macro function
#import ranger.core.actions
#old_get_macros = ranger.core.actions.Actions._get_macros
#
## Define a new macro function
#import time
#def get_macros_MOD(self):
#       macros = old_get_macros(self)
#       macros['date'] = time.strftime('%m/%d/%Y')
#       return macros
#
## Overwrite the old one
#ranger.core.actions.Actions._get_macros = get_macros_MOD
```

- **ranger/rc.conf**: les raccourcis clavier et différents réglages.

```
# ===================================================================
# This file contains the default startup commands for ranger.
# To change them, it is recommended to create the file
# ~/.config/ranger/rc.conf and add your custom commands there.
#
# If you copy this whole file there, add this line to your options.py
# so it is not loaded twice:
#
#     load_default_rc = False
#
# The purpose of this file is mainly to define keybindings.  For
# changing settings or running more complex python code, use the
# configuation file "options.py" or define commands in "commands.py".
#
# Each line is a command that will be run before the user interface
# is initialized.  As a result, you can not use commands which rely
# on the UI such as :delete or :mark.
# ===================================================================
```

```
# ============================================================================
# == Command Aliases in the Console
# ============================================================================

alias e    edit
alias q    quit
alias q!   quitall
alias qall quitall

# ============================================================================
# == Define keys for the browser
# ============================================================================

# Basic
map     Q quit!
map     q quit
copymap q ZZ ZQ

map R     reload_cwd
map <C-r> reset
map <C-l> redraw_window
map <C-c> abort

map i display_file
map ? help
map W display_log
map w taskview_open
map S shell $SHELL

map :  console
map ;  console
map !  console shell
map @  console -p6 shell  %%s
map #  console shell -p
map s  console shell
map r  console open_with
map f  console find
map cd console cd

# Tagging / Marking
map t       tag_toggle
map T       tag_remove
map "<any>  tag_toggle tag=%any
map <Space> mark_files toggle=True
map v       mark_files all=True toggle=True
map V       mark_files all=True val=False
map uv      mark_files all=True val=False

# For the nostalgics: Midnight Commander bindings
map <F1> help
map <F3> display_file
map <F4> edit
map <F5> copy
map <F6> cut
map <F7> console mkdir
map <F8> console delete seriously?
map <F10> exit

# In case you work on a keyboard with dvorak layout
map <UP>       move up=1
map <DOWN>     move down=1
map <LEFT>     move left=1
map <RIGHT>    move right=1
map <HOME>     move to=0
map <END>      move to=-1
map <PAGEDOWN> move down=1   pages=True
map <PAGEUP>   move up=1     pages=True
map <CR>       move right=1
map <DELETE>   console delete
```

```
map <INSERT>   console touch

# VIM-like
copymap <UP>       k
copymap <DOWN>     j
copymap <LEFT>     h
copymap <RIGHT>    l
copymap <HOME>     gg
copymap <END>      G
copymap <PAGEDOWN> <C-F>
copymap <PAGEUP>   <C-B>

map J  move down=0.5  pages=True
map K  move up=0.5    pages=True
copymap J <C-D>
copymap K <C-U>

# Jumping around
map H     history_go -1
map L     history_go 1
map ]     move_parent 1
map [     move_parent -1
map }     traverse

map gh cd ~
map ge cd /etc
map gu cd /usr
map gd cd /dev
map gl cd -r .
map gL cd -r %f
map go cd /opt
map gv cd /var
map gm cd /media
map gM cd /mnt
map gs cd /srv
map gr cd /
map gR eval fm.cd(ranger.RANGERDIR)
map g/ cd /

# External Programs
map E  edit
map du shell -p du --max-depth=1 -h --apparent-size
map dU shell -p du --max-depth=1 -h --apparent-size | sort -rh
map yp shell -d echo -n %d/%f | xsel -i
map yd shell -d echo -n %d    | xsel -i
map yn shell -d echo -n %f    | xsel -i

# Filesystem Operations
map =  chmod

map cw console rename
map A  eval fm.open_console('rename ' + fm.env.cf.basename)
map I  eval fm.open_console('rename ' + fm.env.cf.basename, position=7)

map pp paste
map po paste overwrite=True
map pl paste_symlink relative=False
map pL paste_symlink relative=True
map phl paste_hardlink

map dd cut
map ud uncut
map da cut mode=add
map dr cut mode=remove

map yy copy
map uy uncut
map ya copy mode=add
map yr copy mode=remove
```

```
# Temporary workarounds
map dgg eval fm.cut(dirarg=dict(to=0), narg=quantifier)
map dG  eval fm.cut(dirarg=dict(to=-1), narg=quantifier)
map dj  eval fm.cut(dirarg=dict(down=1), narg=quantifier)
map dk  eval fm.cut(dirarg=dict(up=1), narg=quantifier)
map ygg eval fm.copy(dirarg=dict(to=0), narg=quantifier)
map yG  eval fm.copy(dirarg=dict(to=-1), narg=quantifier)
map yj  eval fm.copy(dirarg=dict(down=1), narg=quantifier)
map yk  eval fm.copy(dirarg=dict(up=1), narg=quantifier)

# Searching
map /  console search
map n  search_next
map N  search_next forward=False
map ct search_next order=tag
map cs search_next order=size
map ci search_next order=mimetype
map cc search_next order=ctime
map cm search_next order=mtime
map ca search_next order=atime

# Tabs
map <C-n>     tab_new ~
map <C-w>     tab_close
map <TAB>     tab_move 1
map <S-TAB>   tab_move -1
map <A-Right> tab_move 1
map <A-Left>  tab_move -1
map gt        tab_move 1
map gT        tab_move -1
map gn        tab_new ~
map gc        tab_close
map <a-1>     tab_open 1
map <a-2>     tab_open 2
map <a-3>     tab_open 3
map <a-4>     tab_open 4
map <a-5>     tab_open 5
map <a-6>     tab_open 6
map <a-7>     tab_open 7
map <a-8>     tab_open 8
map <a-9>     tab_open 9

# Sorting
map or eval fm.settings.sort_reverse ^= True
map os chain set sort=size;      set sort_reverse=False
map ob chain set sort=basename;  set sort_reverse=False
map on chain set sort=natural;   set sort_reverse=False
map om chain set sort=mtime;     set sort_reverse=False
map oc chain set sort=ctime;     set sort_reverse=False
map oa chain set sort=atime;     set sort_reverse=False
map ot chain set sort=type;      set sort_reverse=False

map oS chain set sort=size;      set sort_reverse=True
map oB chain set sort=basename;  set sort_reverse=True
map oN chain set sort=natural;   set sort_reverse=True
map oM chain set sort=mtime;     set sort_reverse=True
map oC chain set sort=ctime;     set sort_reverse=True
map oA chain set sort=atime;     set sort_reverse=True
map oT chain set sort=type;      set sort_reverse=True

# Settings
map zc    toggle_option collapse_preview
map zd    toggle_option sort_directories_first
map zh    toggle_option show_hidden
map <C-h> toggle_option show_hidden
map zi    toggle_option flushinput
map zm    toggle_option mouse_enabled
map zp    toggle_option preview_files
```

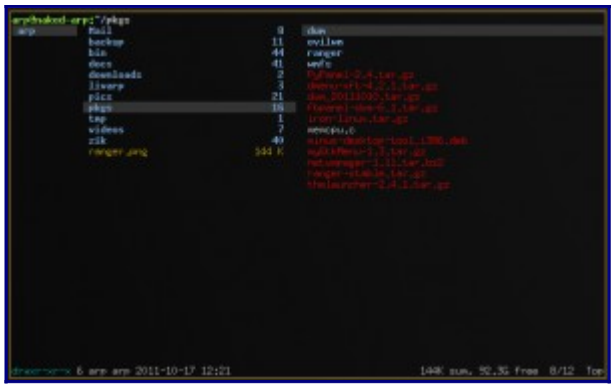
```
map zP    toggle_option preview_directories
map zs    toggle_option sort_case_insensitive
map zv    toggle_option use_preview_script
map zf    console filter

# Bookmarks
map `<any>  enter_bookmark %any
map '<any>  enter_bookmark %any
map m<any>  set_bookmark %any
map um<any> unset_bookmark %any

map m<bg>   draw_bookmarks
copymap m<bg>  um<bg> `<bg> '<bg>

# Beware. I haven't figured out how to make these keybindings pretty yet:

# map +ow shell -d chmod o+w (one mapping for each combination)
eval import itertools; [cmd("map +%s%s shell -d chmod %s+%s %%s" % (mode+mode)) for mode
in itertools.product("ugoa", "rwxXst")]

# map -ow shell -d chmod o+w (one mapping for each combination)
eval import itertools; [cmd("map -%s%s shell -d chmod %s-%s %%s" % (mode+mode)) for mode
in itertools.product("ugoa", "rwxXst")]

# ===================================================================
# == Define keys for the console
# ===================================================================
# Note: Unmapped keys are passed directly to the console.

# Basic
cmap <tab>   eval fm.ui.console.tab()
cmap <s-tab> eval fm.ui.console.tab(-1)
cmap <ESC>   eval fm.ui.console.close()
cmap <CR>    eval fm.ui.console.execute()
cmap <C-l>   redraw_window

copycmap <ESC> <C-c>
copycmap <CR>  <C-j>

# Move around
cmap <up>    eval fm.ui.console.history_move(-1)
cmap <down>  eval fm.ui.console.history_move(1)
cmap <left>  eval fm.ui.console.move(left=1)
cmap <right> eval fm.ui.console.move(right=1)
cmap <home>  eval fm.ui.console.move(right=0, absolute=True)
cmap <end>   eval fm.ui.console.move(right=-1, absolute=True)

# Line Editing
cmap <backspace>  eval fm.ui.console.delete(-1)
cmap <delete>     eval fm.ui.console.delete(0)
cmap <C-w>        eval fm.ui.console.delete_word()
cmap <C-k>        eval fm.ui.console.delete_rest(1)
cmap <C-u>        eval fm.ui.console.delete_rest(-1)
cmap <C-y>        eval fm.ui.console.paste()

# And of course the emacs way
copycmap <up>        <C-p>
copycmap <down>      <C-n>
copycmap <left>      <C-b>
copycmap <right>     <C-f>
copycmap <home>      <C-a>
copycmap <end>       <C-e>
copycmap <delete>    <C-d>
copycmap <backspace> <C-h>

# Note: There are multiple ways to express backspaces.  <backspace> (code 263)
# and <backspace2> (code 127).  To be sure, use both.
copycmap <backspace> <backspace2>
```

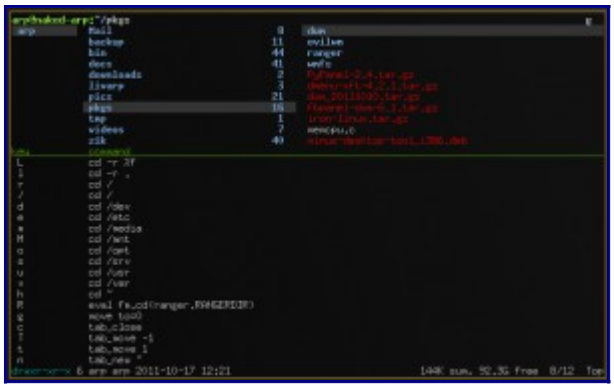
```
# This special expression allows typing in numerals:
cmap <allow_quantifiers> false

# ================================================================
# == Pager Keybindings
# ================================================================

# Movement
pmap  <down>      pager_move  down=1
pmap  <up>        pager_move  up=1
pmap  <left>      pager_move  left=4
pmap  <right>     pager_move  right=4
pmap  <home>      pager_move  to=0
pmap  <end>       pager_move  to=-1
pmap  <pagedown>  pager_move  down=1.0  pages=True
pmap  <pageup>    pager_move  up=1.0    pages=True
pmap  <C-d>       pager_move  down=0.5  pages=True
pmap  <C-u>       pager_move  up=0.5    pages=True

copypmap <UP>       k  <C-p>
copypmap <DOWN>     j  <C-n> <CR>
copypmap <LEFT>     h
copypmap <RIGHT>    l
copypmap <HOME>     g
copypmap <END>      G
copypmap <C-d>      d
copypmap <C-u>      u
copypmap <PAGEDOWN> n  f  <C-F>  <Space>
copypmap <PAGEUP>   p  b  <C-B>

# Basic
pmap     <ESC> pager_close
copypmap <ESC> q Q i <F3>
pmap E      edit_file

# ================================================================
# == Taskview Keybindings
# ================================================================

# Movement
tmap <up>        taskview_move up=1
tmap <down>      taskview_move down=1
tmap <home>      taskview_move to=0
tmap <end>       taskview_move to=-1
tmap <pagedown>  taskview_move down=1.0  pages=True
tmap <pageup>    taskview_move up=1.0    pages=True
tmap <C-d>       taskview_move down=0.5  pages=True
tmap <C-u>       taskview_move up=0.5    pages=True

copytmap <UP>       k  <C-p>
copytmap <DOWN>     j  <C-n> <CR>
copytmap <HOME>     g
copytmap <END>      G
copytmap <C-u>      u
copytmap <PAGEDOWN> n  f  <C-F>  <Space>
copytmap <PAGEUP>   p  b  <C-B>

# Changing priority and deleting tasks
tmap J          eval -q fm.ui.taskview.task_move(-1)
tmap K          eval -q fm.ui.taskview.task_move(0)
tmap dd         eval -q fm.ui.taskview.task_remove()
tmap <pagedown> eval -q fm.ui.taskview.task_move(-1)
tmap <pageup>   eval -q fm.ui.taskview.task_move(0)
tmap <delete>   eval -q fm.ui.taskview.task_remove()

# Basic
tmap <ESC> taskview_close
copytmap <ESC> q Q w <C-c>
```

utilisation

même si tout cela peut parraitre complexe, ranger est utilisable immédiatement sans aucune configuration majeure. de plus, ranger dispose d'une série d'outils facilitant grandement sa prise en main:

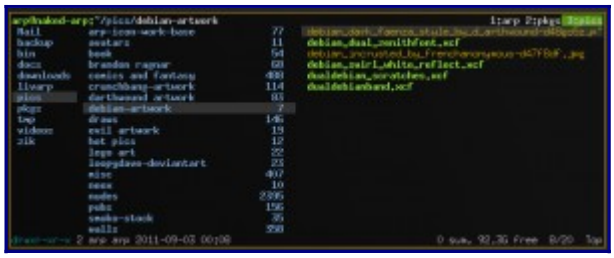
- la **fenêtre pop-up** (en console) qui apparait lorsqu'un raccoucis clavier est lancé: par exemple, le raccoucis **gh** (**g**o **h**ome) vous ramène à votre dossier utilisateur. si vous commencez à taper "**g**", la liste des commandes disponibles apparait:





- ranger gère **les onglets**: pour ouvrir un onglet "**gn**" pour naviguer entre les onglets "**g'n'**" ou [Tab], pour avoir le nom du dossier dans l'onglet, éditer la ligne 65 du ranger/options.py pour obtenir

```
# Display the directory name in tabs?
dirname_in_tabs = True
```



- ranger utilise les **raccourcis de mc** (pour les nostalgiques).
  - <F1> help
  - <F3> display_file
  - <F4> edit
  - <F5> copy
  - <F6> cut
  - <F7> console mkdir
  - <F8> console delete seriously?
  - <F10> exit
- ranger utilise les **raccourcis vim** pour la navigation mais aussi les flèches directionnelle et la souris. la liste complète est contenue dans le fichier ranger/rc.conf.
- ranger utilise les raccourcis vim pour les opérations de fichiers
  - yy copy
  - pp paste
  - dd cut
  - / recherche
  - n rechercher le suivant
  - ….

conclusion

un **file-manager en console** qui pourrait bien réconcilier certains utilisateurs avec le terminal :) . rapide, très facile à prendre en main, pourvu d'un excellent système d'aide, le seul défaut de ranger serait qu'il est anglophone… à part ça, je ne peux que vous inviter à essayer ce petit file-manager.