mk-project Documentation

Release 2.1

Brüggemann Eddie

Nov 15, 2017

CONTENTS:

1 mk-project **1**
- 1.1 Presentation . . . 1
- 1.2 Hackme . . . 2
- 1.3 What provide **mk-project** . . . 2
- 1.4 How **mk-project** works . . . 4
- 1.5 Author(s) . . . 4
- 1.6 Contributor(s) . . . 4
- 1.7 The future of **mk-project** . . . 5

2 Building a mk-project project **7**
- 2.1 Starting . . . 7
- 2.2 **C/C++** Compiler settings . . . 8
- 2.3 Files . . . 9
- 2.4 Disassembling . . . 10
- 2.5 Profiling . . . 10
- 2.6 **C/C++** code formatters . . . 11
- 2.7 Documentation . . . 11
- 2.8 About informations . . . 11
- 2.9 Others Informations . . . 12
- 2.10 Licensing . . . 12
- 2.11 *.desktop file . . . 12
- 2.12 Archiving your project . . . 12
- 2.13 Summary . . . 12

3 Working on an existing mk-project project **15**
- 3.1 Open an **mk-project** project . . . 15
- 3.2 Reconfiguring an existing project . . . 15
- 3.3 Adding files to your project . . . 16

4 mk-project documentation **17**
- 4.1 Introduction . . . 17
- 4.2 You are a sphinx user: . . . 17
- 4.3 You aren't a sphinx user . . . 17
- 4.4 **mk-project** documentation visualize . . . 18
- 4.5 **mk-project** slots . . . 19
- 4.6 `rst2man` . . . 20

5 mk-project code investigating, debugging and disassembling **21**
- 5.1 Introduction . . . 21
- 5.2 The `make info` target . . . 21

5.3 The `make gdb` target . . . 21
5.4 The `make ldd` target . . . 22
5.5 The `make nm` target . . . 22
5.6 The `make objdump` target . . . 23
5.7 The `make strace` target . . . 24
5.8 The `make ltrace` target . . . 25
5.9 The `make strip` target . . . 25
5.10 Oprofile targets . . . 25
5.11 Valgrind targets . . . 26
5.12 Alternative to *_OPTS . . . 26
5.13 Documentation Source . . . 27

6 mk-project code formatters **29**
6.1 Introduction . . . 29
6.2 Using the `indent` utility . . . 29
6.3 Using the `astyle` utility . . . 30

7 `mk-project` contributing advices **31**
7.1 mk-project zen . . . 31
7.2 How contibute ? . . . 31
7.3 Makefile . . . 33
7.4 code formatters . . . 35

8 *.todo or *.tdo file format specifications: **37**
8.1 Markup syntax . . . 37
8.2 Markup Types . . . 37
8.3 Priority_level . . . 38
8.4 TASK_ORDER . . . 39
8.5 Advices . . . 39
8.6 Syntax of ***.todo** file(s) content . . . 40
8.7 End word of specifications of the *.todo file(s) format . . . 42
8.8 Example of a *.todo file: . . . 42
8.9 License . . . 43

9 About mk-project **45**
9.1 A word from the author . . . 45
9.2 Dependencies . . . 45

10 `mk-project` Gtk3 types **49**
10.1 GtkSmartIconButton . . . 49
10.2 GtkSmartIconButton . . . 51
10.3 GtkTermTab . . . 51
10.4 GtkMkTerm . . . 52

11 Indices and tables **55**

Index **57**

CHAPTER

ONE

MK-PROJECT

author Brüggemann Eddie <mrcyberfighter@gmail.com>

program mk-project

version 2.1

language C

release Nov 15, 2017

1.1 Presentation

mk-project is a **C** and **C++** project builder, with a nice G.U.I (Graphical User Interface), which generate at first a big, big do it all, `Makefile`.

So that you can create a project and keep the tree of your project, which reflect the UNIX file system tree.

> **note** **mk-project** does not claim to replace the auto-tools but it is build on the top in the spirit of development instead of distributing.

In addition **vim** or others T.U.I (Terminal User Interface) editors users,

can use the entire program by editing their source files into the *Terminals* → *Edit terminal* which is a notebook, you can adding as many terminals you want.

Else **mk-project** is a tool done to *ease the development process* of **C** or **C++** programs and a good bridge for the distributing process. Especially with the **autotools**.

mk-project is a T.D.E (Terminal Development Environment) (**T**erminal **D**evelopement **E**nvironment):

> an **utility** used for the **development** of **programs** with many functionalities !

mk-project is based on the **make** tool which on his turn, use severals utilities, for providing many features

and so many useful **make** targets.

Callable through a terminal, in preference (or through the G.U.I from **mk-project**: *targets* → *).

mk-project is an `Environment` in the terms of his wide field of `targets` which aren't statically at all.

But **dynamically** *configurable*, *changeable* and *self-build-able* and so that the grass becomes greener...

The `targets` are short string easy to remember and so you can make work

- your computer, through the terminals
- and mostly your head through **remembering**, **configuring**, **modifying** and **creating** `targets`.

And not become an I.D.E (**I**ntegrated **D**evelopment **E**nvironment) **zombie** thrashing his head and knows !

But a proud well informed programmer which knows exactly how his system and environment works which can easily automate the task using the **make** syntax.

1.2 Hackme

You can **edit** the `Makefile` by hand at your convenience, of course **!**

Note: But I think It's better for some *generic targets to include* them directly

into the template file(s) you will find at **$(pkgdatadir)**: `/usr(/local/share/mk-project/templates/*`

So that you get it every time you generate a new project.

> **warning** If you do this: you must take care of escaping the '%' with a '%' character : "%%".

But think to notify the *developers*,

to inform them about your add-on(s) if you think it's reliable and usable for others.

1.3 What provide `mk-project`

Note: At the time i write this documentation **mk-project** support:

the **C** and **C++** programming language.

"I invite all the community to work together to take in charge more languages..."

> **file** *see this document for participating (You can become from the simple contributor to the entire coauthor).*

- **mk-project** provide at first a solid base for building a work,

  through a big Makefile, which can be edited manually in respect of the following few conventions:

Note:

- Configuration settings are set through the string:
  - **F** For false (disable option).

  - **T** for true (enable option).

- Some few others variables:

  The variable **$(SRC_FILES)** is build from the variable **$(SRC_DIR)** which value is always: `./src`.

  This mean if you want to add files manually (*if you doesn't use the GUI for this task*) to your Makefile,

  do it properly by using the **$(SRC_DIR)** variable:

```
SRC_DIR   = ./src

SRC_FILES = ${SRC_DIR}/my_file${EXT_SRC} \
            ${SRC_DIR}/subfolder/my_file${EXT_SRC}
```

  So you will add file(s) relative to the `./src` directory where source file(s) have to reside.

  note Otherwise simply use the G.U.I for adding file(s) *Project -> Add file(s) to project*.

- A building system for your source files.
- Many tools for machine code investigation:

  From the simple **-g** option setting by a `GNU-Compiler` for debbuging with `gdb`, through **disassemble** the *machine code* files and *executable tracing*, to **profiling** the entire work.

- For the documentation **mk-project** support *the sphinx documentation generator*.

  The **sphinx** documentation `targets` support many output formats:

  - **info** files.
  - **man** (manual pages).
  - **HTML**, single **HTML**, and **texi HTML** documentation.
  - **PDF** and **LATEXPDF** files.
  - **XML** files.
  - **LATEX** files.
  - **EPUB** files.

  And many more through **sphinx** like: **qthelp**, **applehelp**, **xml**, **json** or **devhelp** per example.

mk-project provide a simple G.U.I composing of terminals and a menu-bar.

At first you can use the menu items to perform some actions like:

- Generate a new project: *Project -> New Project*.

  Then you have to configure your project answering some basics questions like:

  - Programming language.
  - Program name
  - Project folder (in which the new project will be generate).

  And some others according to your settings.

  Once the new project is generated you can access to the make targets either through the **mk-project** G.U.I menu-bar (*simple click on the wanted target to execute it !*).

  Or from the terminal of **mk-project** or any else terminal at the condition to be in the `Makefile` current folder.

note Simply type **`$ make help`** to get the list of available targets.

Warning: If you add some user-targets, to the `Makefile(s)`, think at adding them to the **`$ make help`** output.

So that mk-project can auto-detect your target and list it to add it as menu item to the make targets.

If you add a bash comment on the same line it will be displayed as tool-tip by overfly the menu items.

warning Simply think to limit your entry at terminal maximal size: 79 characters.

So **`mk-project`** provide another terminals ordered in tabs which you can add, remove, and configuring.

For purpose of terminals editor users like **`vi`**, **`ed`**, **`emacs`** which can be easily launch an instance their favorite `terminal editor` in every tab all that continuing using the **`mk-project`** *interface*.

Finally you can switch between the single *terminals -> make terminal* (which should stay in the `Makefile` current folder) and the *terminals -> edit terminals* terminals using the menu radio items.

1.4 How `mk-project` works

note The answers is simply `all is make in Makefiles`, which will make you *the development easier*.

`mk-project` doesn't claim to replace an **IDE** or others **building tools** but only give you an alternative

which `you can entirely adapt` to your requirement.

Note: For being **true** the **`make`** tool implementation and the way it make you the life easier

without forgetting your `TTY Knows` has impress me so that

I couldn't develop a good project without it

or in others words:

```
If the make tool have never exist I would invent it...
```

1.5 Author(s)

Developer Brüggemann Eddie

Documenter Brüggemann Eddie

1.6 Contributor(s)

Become one !!!

1.7 The future of `mk-project`

1.7.1 `mk-project: mic-on` !!!

Note: The idea is to sit in front of the interface of **`mk-project`** (`microphone on` **!**),

writing the *source code* from your last creation:

And to say `execute:  make exec`, or the target you want...

The **program** could **react** by *analyzing your voice entry* and **executing the target** !!!

So that the build is **automate** by (*simple*) **voice recognition**.

So you can write your **program** with your hands

and

build it with by emitting a *simple order* so that the *program execute the corresponding target*,

if recognize...

What do you think about it ?

note We could enhance **`mk-project`** in the way of Speech recognition...

CHAPTER

TWO

BUILDING A MK-PROJECT PROJECT

2.1 Starting

At first some basic informations will be required:

- The programming language from the project.
  - **C**

  Or

  - **C++**
- The project name which will become the program name

  the binary name.
- The **program version** which, if empty will be arbitrary set to the value `0.0`.
- The folder where to generate the project.

  warning The folder must be empty (Advice: create it with the folder-chooser at the same time as the project).
- The license of your project.
  - GPL
  - AGPL
  - LGPL
  - FDL
  - Apache 2.0 License
  - Clear BSD
  - Free BSD
  - Other

  note The license files will be copied into the project folder according to your choice of format(s).
- docbook
- epub
- pdf
- latex
- html

- texinfo
- text

2.2 C/C++ Compiler settings

For every entry except the "Compiler entry" you get an aside button which

will permit you to add the *most common settings* **easily**.

- **Compiler:**

  You can choose a compiler to use, which default to `cc` for a **C** project and `c++` for **C++**.

  But you can set clang per example or the compiler you want.

  warning The exactness of your entry will be checked by compiling a minimal program.

- **Warnings:**

  You can set the warnings to use.

  Note: The aside button will permit you to insert as warnings the following most common warnings settings:

  - `-Wall` (*All warning: sea the documentation of your compiler to see which are enabled*).
  - `-Wextra` (*Extra warning: sea the documentation of your compiler to see which are enabled*).
  - `-Wpedantic` (*ISO conform: most extension are permitted. sea the documentation of your compiler to see which are enabled*).
  - `-w` (*No output warnings*).
  - `-Werror` (*A warning is consider as an error*).

  ---

  warning The field is empty per default.

- **CFLAGS:**

  You can set the argument to give to the compiler (like -g, -O2,...).

  Note: The aside button provide few flags adding:

  - `-g`
  - `-O[0123gsf]`
  - `-std=`
  - `-pedantic`

- **CPPFLAGS:**

  Preprocessor instruction to pass onto the compile command line.

  Note: The aside button permit you to define a definition with a value or without.

- **LDFLAGS:**

  Dynamic Linker Flags.

  Note: The aside button will permit you to choose the **`pkg-config`**

  you want to add to your project.

  By listing all the **`pkg-config`** available on your system.

  ---

  warning By hand editing, if you use **`pkg-config`**, use the back-ticks syntax:

  Else this will not work because of the make syntax.

- **LDLIBS:**

  Dynamic Linker library libraries: per example `-lm`.

  Note: The aside button will permit you to add the linker of your choice.

  By listing all linker flags available on your system.

2.3 Files

Here you must set the extension you will use for the source and header files.

Especially for the **C++** language:

- Source files:
  - *.cpp*
  - *.CPP*
  - *.c++*
  - *.C*

    warning **This is very important** because of the compilation automation which will **not work** with the **wrong extension**.
- Header files:
  - *.h*
  - *.hh*
  - *.H*
  - *.hp*
  - *.hxx*
  - *.hpp*
  - *.HPP*
  - *.h++*
  - *.tcc*

  note For the **C** language this default to *.c* and *.h*.

2.4 Disassembling

Here you can give the default options to pass to the debugging tools:

- **nm** options.
- **gdb** options.
- **strace** options.
- **ltrace** options.
- **objdump** options.
- **ldd** options.
- **gprof** options.

  note For further informations sea the *mk-project code investigating, debugging and disassembling page*.

2.5 Profiling

2.5.1 Oprofile

mk-project use **Oprofile** version >= 1.0 for profiling you code.

You can set the following default options:

- **operf** options.
- **ocount** options.
- **opreport** options.
- **opannotate** options.
- **opgprof** options.

2.5.2 Valgrind

mk-project provides 4 **valgrind** targets per default:

```
make valgrind-memcheck    # Launch the valgrind memcheck tool on your binary.

make valgrind-cachegrind # Launch the valgrind cachegrind tool on your binary.

make valgrind-callgrind   # Launch the valgrind callgrind tool on your binary.

make valgrind-helgrind    # Launch the valgrind helgrind tool on your binary.
```

For this **valgrind** targets you can set the options.

note You can define more **valgrind** targets by *editing the template*.

You can give options to apply to valgrind by setting the environment variable `VALGRIND_OPTS`.

Or by passing it like this:

```
$ make valgrind-memcheck VALGRIND_MEMCHECK_OPTS=--option=value

$ make valgrind-cachegrind VALGRIND_CACHEGRIND_OPTS=--option=value

$ make valgrind-callgrind VALGRIND_CALLGRIND_OPTS=--option=value

$ make valgrind-helgrind VALGRIND_HELGRIND_OPTS=--option=value
```

2.6 C/C++ code formatters

Here you can choose the code formatter(s) you want to use.

- You can set the options to give to **indent** and to **astyle** for the `indent-user` and `astyle-user` target if you know this tools.

  note But **mk-project** provides a lot of pre-configurated **astyle**, **indent**, **bcpp** targets.

- You can set the `indentation width` to use and wether to use tabulation or not during the formatting process.

  note For further information sea the page: *mk-project code formatters*

2.7 Documentation

1. Simply choose to use *sphinx or not*.
2. Set the options according to your sphinx version.
3. *Enable/Disable* the wanted sphinx extensions.

   sphinx This will generate a `Makefile` and **sphinx** specific targets.

2.7.1 man-page

The man page generating is separated from the documentation because they normally

does not contains the same,

so **mk-project** provides through the **rst2man** tool an option

to build (using the REST (ReStructured Text) syntax) and view a man page.

2.8 About informations

Here you can set some informations about your program.

- Author(s).
- Mail address.

- Program URL.
- Copyright string.

  note All this informations will generate some constant definition into the `./headers/defines.${EXT_HDR}` file.

2.9 Others Informations

- Make options: the options to pass to **make** at every call.
- The bash location (auto-detect).
- Compression level for the `pkg-\*` targets, with which you can build an archive from your project.

2.10 Licensing

You can edit a source code files header according to the chosen license.

And add it to every source file with the target:

make prepend-license.

2.11 *.desktop file

You can build a desktop file with this boilerplate.

2.12 Archiving your project

mk-project provides many compressed archiving targets:

- **zip** archive.
- **tar** archive compressed with **lzma**, **xz**, **gz**, **bz**.
- **rar** archive.

If the wanted archive program is installed at your site. What is not oblige.

note **mk-project** provides through his G.U.I a: *Project –> extract and load* menu item.

2.13 Summary

Last step to complete the generation of your project.

Enjoy the easiness of working with **mk-project** the T.D.E TERMINAL DEVELOPMENT ENVIRONMENT.

2.13.1 Exporting your settings

You can exporting your settings as a **mk-project** profile.

To load it by the next project because typing all this options can be painful.

You will get the most wanted settings setted like `nm_options` per example,

but not all like the program name.

> **warning** The file extension will arbitrary set to `*.mkpp`.

CHAPTER

THREE

WORKING ON AN EXISTING `mk-project` PROJECT

3.1 Open an `mk-project` project

For opening an existing project you can make use of the `*.mkp` file from your project.

- Either by calling **`mk-project`** with the `*.mkp` file given as argument:

  ```
  $ mk-project /path/to/project_folder/prgname.mkp
  ```

  Or open the project within **`mk-project`**'s G.U.I (*Projects → Load project*).

- By using your file manager:

  Simply click on the `*.mkp` file in the project folder

  or

  Opening the `*.mkp` file with your file manager using the `open with` option.

  To open the **`mk-project`** G.U.I and loading the entire project.

- Using the **`mk-project`** G.U.I:

  Use the menu item *Projects → Load project* and choose the `*.mkp` of interest.

  To load the entire project in the **`mk-project`** G.U.I

All targets will be available according to your settings.

Note: Else you can simply use a terminal to use the **`mk-project`** projects:

simply type `make help` in the project folder to sea the available targets.

3.2 Reconfiguring an existing project

Open the **`mk-project`** G.U.I and use the menu item *Projects -> Reconfigure project*

to open the project reconfiguring project interface.

Here you can:

- **Change** some settings of your project.
- **Enable** or **disable** some features.
- **Edit** the Licensing boilerplate to prepend it to all source and header files if you want to do so.

- **Edit** the desktop file boilerplate.

3.3 Adding files to your project

Open an existing project and then use the menu item *Projects -> Add file(s) to project*.

Then select the file(s) you want to add to your project.

Note: Take care of the checkbutton in the file chooser !

- You can choose to add the corresponding *header* file to your project.

  note If the header file doesn't exist it will be create.

Warning: The file(s) must be in the `./src` folder or subfolders from it !

Take care to organize your project properly so that all source files still in the `./src` folder from your project !

Else you can add the file(s) to your project anyway but this can break your project tree if you rename the project folder.

note It's better to create sub-folders from the `./src` folder to organize your project properly !

CHAPTER

FOUR

MK-PROJECT DOCUMENTATION

4.1 Introduction

mk-project currently support only one single documentation generator: **sphinx**.

sphinx was first design to generate official python documentation but time has past and **sphinx**

has become very popular at first by the python community.

Without any particular extension (like autodoc) **sphinx** is based on the R.e.S.T Re Structured Text language but Mardown can be used through modifying the `conf.py` file.

The Rest language is an easy markup language as like markdown but it is standardized (Markdown not) and can be extented what sphinx does.

> **note** Some R.e.S.T Re Structured Text extension permit to build **C** andf **C++** documentation without using any extension.

4.2 You are a sphinx user:

So everything is are right.

4.3 You aren't a sphinx user

Let me convince you to adopt sphinx by learning the easy markup R.e.S.T Re Structured Text or Markdown language.

For generating documentation in many formats, **mk-project** generate make targets from the make output according which sphinx extension are installed on your system.

Per exemple currently on my system.

```
make sphinx-html # to make standalone HTML files

make sphinx-dirhtml # to make HTML files named index.html in directories

make sphinx-singlehtml # to make a single large HTML file

make sphinx-pickle # to make pickle files

make sphinx-json # to make JSON files

make sphinx-htmlhelp # to make HTML files and an HTML help project
```

```
make sphinx-qthelp # to make HTML files and a qthelp project

make sphinx-applehelp # to make an Apple Help Book

make sphinx-devhelp # to make HTML files and a Devhelp project

make sphinx-epub # to make an epub

make sphinx-epub3 # to make an epub3

make sphinx-latex # to make LaTeX files, you can set PAPER=a4 or PAPER=letter

make sphinx-latexpdf # to make LaTeX files and run them through pdflatex

make sphinx-latexpdfja # to make LaTeX files and run them through platex/dvipdfmx

make sphinx-lualatexpdf # to make LaTeX files and run them through lualatex

make sphinx-xelatexpdf # to make LaTeX files and run them through xelatex

make sphinx-text # to make text files

make sphinx-man # to make manual pages

make sphinx-texinfo # to make Texinfo files

make sphinx-info # to make Texinfo files and run them through makeinfo

make sphinx-gettext # to make PO message catalogs

make sphinx-changes # to make an overview of all changed/added/deprecated items

make sphinx-xml # to make Docutils-native XML files

make sphinx-pseudoxml # to make pseudoxml-XML files for display purposes

make sphinx-linkcheck # to check all external links for integrity

make sphinx-doctest # to run all doctests embedded in the documentation (if enabled)

make sphinx-coverage # to run coverage check of the documentation (if enabled)
```

- Many themes are available.
- Many contrib packages are available for extending sphinx in many ways.
- The RDT (Read The Doc) theme provide a web service, format the output in his theme and provide the documentation downloadable in many format.

4.4 `mk-project` documentation visualize

`mk-project` permit you to visualize all the output files in different manners:

`mk-project` will search severals documentation viewer programs on your installation.

Note:

- The **make** varibale **${BROWSER}** will link to your **browser**.
- The **make** variable **{INFO}** will link to the **info** program.
- The **make** variable **{MAN}** will link to the **man** program.
- The **make** variable **{EPUB}** will link to your **epub** viewer (**fbreader** or **okular**) if available.
- The **make** variable **{PDF}** will link to your **pdf** viewer if available.

  note If **mk-project** doesn't find a binary for viewing a file it will use the **xdg-open** program as fallback.

Warning: The `sphinx-show-\*` targets are set *arbitrary*

as best as I can

because their is either **no** way to know into which `sub-folder` the **documentation** will be **generate**

and **nor** the **filename the documentation will have**...

Simply **trust me** or correct it **yourself** if necessary.

4.5 mk-project slots

Always remember that you can write some **make** targets into the **mk-project** Makefile.

To ease you the documentation generating process and so *extend mk-project*.

Per example by the first version of mk-project, it use a mix of:

- The `pandoc` package.
- The `python(3)-docutils` and the `rst2pdf` packages.
- The `texinfo` and `texlive` packages.

To provide ReST, Markdown and texinfo documentation generation but Only one page per output format.

but I used **sphinx** to write the documentation of the version 1.0 of **mk-project**

with some few self -builded targets like this:

```
##############################################################################

# sphinx slot.

.PHONY: sphinx-singlehtml sphinx-html sphinx-htmlhelp sphinx-epub sphinx-info sphinx-
↪man

# sphinx Makefile singlehtml target link.
sphinx-singlehtml:
      cd sphinx_doc ; ${MAKE} singlehtml ;

# sphinx Makefile html target link.
sphinx-html:
      cd sphinx_doc ; ${MAKE} html ;
```

```
# sphinx Makefile epub target link.
sphinx-epub:
	cd sphinx_doc ; ${MAKE} epub ;

# sphinx Makefile info target link.
sphinx-info:
	cd sphinx_doc ; ${MAKE} info ;

# sphinx Makefile man target link.
sphinx-man:
	cd sphinx_doc ; ${MAKE} man ;

# sphinx Makefile doctest target link.
sphinx-doctest:
	cd sphinx_doc ; ${MAKE} doctest


# sphinx builded files showing targets.
.PHONY: sphinx-show-singlehtml sphinx-show-html sphinx-show-epub sphinx-show-info␣
↪sphinx-show-man

sphinx-show-singlehtml:
	cd ./sphinx_doc/build/singlehtml ; ${BROWSER} index.html ;

sphinx-show-html:
	cd ./sphinx_doc/build/html ; ${BROWSER} index.html ;

sphinx-show-epub:
	cd ./sphinx_doc/build/epub  ; ${EPUB} *.epub ;

sphinx-show-info:
	cd ./sphinx_doc/build/texinfo ; ${INFO} -f *.info ;

sphinx-show-man:
	cd ./sphinx_doc/build/man ; ${MAN} -f ${PRGNAME}.${MAN_LEVEL} ;

# sphinx clean target.
sphinx-clean:
	cd sphinx_doc ; cd build ; rm -R * ;

################################################################################
```

4.6 rst2man

If you get the program **rst2man** installed on your system,

mk-project will create a folder named `rst2man` into the project tree into

which you will find a file `${PRGNAME}.rst` to edit a man-page with **rst2man**.

warning Because the man-page is often apart from the documentation.

CHAPTER

FIVE

MK-PROJECT CODE INVESTIGATING, DEBUGGING AND DISASSEMBLING

5.1 Introduction

You want to investigate in deep your binary cause of it bugs or simply per curiosity.

mk-project provide a lot of targets which will make your investigation easier.

5.2 The `make info` target

The `make info` target display simple informations about your program.

Using the **file**, **size**, **ls -s -h** utilities.

5.2.1 The `file` utility

The **file** utility print out the details about any file-type.

So if an executable file is given, the **file** utility will display informations about it.

5.2.2 The `size` utility

The **size** utility print out the byte length of the main binary components:

`.text`, `.data` and `.bss`.

5.3 The `make gdb` target

At first you can use the famous `GNU/Debugger` **gdb** for investigating your program,

by simply launching the target `make gdb`.

This will launch **gdb** in the **./bin** folder where your binary is located

with your program as argument.

Note: Given options to **gdb**:

For given supplementary options to gdb, which will be passed to **gdb** at every target call, edit the **GDB_OPTS** variable.

Else if you want to change the options for a unique call of **gdb**, by using the target.

Simply set the wanted options into the **GDB_OPTS** variable on the command line:

```
$ make gdb GDB_OPTS="--option value"
```

5.4 The `make ldd` target

The **ldd** utility show the complete list of dynamic libraries which your program will try to load (i.e. *The load time dependencies*).

Note: Given options to **ldd**:

For given supplementary options to **ldd**, which will be passed to **ldd** at every target call, edit the **LDD_OPTS** variable.

Else if you want to change the options for a unique call of **ldd**, by using the target.

Simply set the wanted options into the **LDD_OPTS** variable on the command line:

```
$ make ldd LDD_OPTS="--option value"
```

Warning: Limitations of ldd:

- **ldd** cannot identify the libraries dynamically loaded at runtime using `dlopen()`.

```
Be aware, however, that in some circumstances, some version of ldd may attempt to␣
↪obtain the dependencies informations
by directly executing the program. Thus, you should never employ ldd on untrusted␣
↪executables,
since this may result in the execution of arbitrary code.
```

A safer alternative when dealing with untrusted executables is following:

```
$ objdump -p /path/to/binary | grep NEEDED

# The same result result can be achieve using the readelf utility.

$ readelf -d /path/to/binary | grep NEEDED
```

5.5 The `make nm` target

The **nm** utility is used to list the symbols of a binary or object file(s).

It can also find the indicated symbol type.

Note: Given options to **nm**:

For given supplementary options to **nm**, which will be passed to **nm** at every target call, edit the **NM_OPTS** variable.

Else if you want to change the options for a unique call of **nm**, by using the target.

Simply set the wanted options into the **NM_OPTS** variable on the command line:

```
$ make nm NM_OPTS="--option value"
```

note You can give the **$(OBJECT) make** variable as argument to **nm** instead of the binary.

Note: If the binary contains some **C++** code, the symbols are printed by default in mangled form.

Usage examples:

```
$ nm /path/to/prg

# List all symbols of prg (a binary or object file(s)).

$ nm -D /path/to/prg

# List only the symbols contains into the dynamic section(s) (exported or visible).

$ nm -C /path/to/prg

# List symbols in demangled form.

$ nm -D --no-demangle /path/to/prg

# List symbols in not demangled form.

$ nm -u /path/to/prg

# List undefined symbols.
```

Look at The 20 best nm commands

5.6 The `make objdump` target

objdump is one of the most versatile utility program, so it can support about 50 others binary formats other than the ELF format.

Note: Given options to **objdump**:

For given supplementary options to **objdump**, which will be passed to **objdump** at every target call, edit the **OBJDUMP_OPTS** variable.

Else if you want to change the options for a unique call of **objdump**, by using the target.

Simply set the wanted options into the **OBJDUMP_OPTS** variable on the command line:

```
$ make objdump OBJDUMP_OPTS="--option value"
```

note You can give the **$(OBJECT) make** variable as argument to **objdump** instead of the binary.

Usage examples:

```
$ objdump -f /path/to/prg

# Is used to obtain an insight into the object file(s) header.
```

```
# The header provide plenty of informations like
#
# * binary type
# * entry point (The start of the .text section)
# * etc..

$ objdump -h /path/to/prg

# Is used to list the available sections from the prg.

$ objdump -T /path/to/prg

# List dynamic symbols only.

# Is equivalent to running: $ nm -D /path/to/prg

$ objdump -t /path/to/prg

# Examines the dynamic section(s).

$ objdump -R /path/to/prg

# Examines the relocation section(s).

$ objdump -S -j <section name> /path/to/prg

# Provide the hex-dump of the values carried by the given section.

$ objdump -p /path/to/prg

# Display informations about the ELF binary segments.
```

Usage example for code disassembling using **objdump**:

```
$ objdump -d -M intel /path/to/prg

# Used to disassemble a binary using the Intel syntax.

$ objdump -d -S -M intel /path/to/prg

# Like above but interspercing the original source code.

$ objdump -d -M intel -j <section name> /path/to/prg

# This only works if the binary is compiled with the -g (debugging) option.
```

5.7 The `make strace` target

The **strace** utility tracks down the **system calls** made by the **process** as well as the **signals** received by the **process**.

Note: Given options to **strace**:

For given supplementary options to strace, which will be passed to **strace** at every target call, edit the **STRACE_OPTS** variable.

Else if you want to change the options for a unique call of **strace**, by using the target.

Simply set the wanted options into the **STRACE_OPTS** variable on the command line:

```
$ make strace STRACE_OPTS="--option value"
```

5.8 The make ltrace target

The **ltrace** utility tracks down the **libraries calls** made by the **process**.

Note: Given options to **ltrace**:

For given supplementary options to ltrace, which will be passed to **ltrace** at every target call, edit the **LTRACE_OPTS** variable.

Else if you want to change the options for a unique call of **ltrace**, by using the target.

Simply set the wanted options into the **LTRACE_OPTS** variable on the command line:

```
$ make ltrace LTRACE_OPTS="--option value"
```

5.9 The make strip target

The **strip** utility can be used to eliminated all the symbols not needed in the process.

Note: Given options to **strip**:

For given supplementary options to strip, which will be passed to **strip** at every target call, edit the **STRIP_OPTS** variable.

Else if you want to change the options for a unique call of **strip**, by using the target.

Simply set the wanted options into the **STRIP_OPTS** variable on the command line:

```
$ make strip STRIP_OPTS="--option value"
```

5.10 Oprofile targets

The program collection Oprofile is a profiling system for systems running Linux 2.6.31 and greater.

OProfile makes use of the hardware performance counters provided on Intel, AMD, and other processors.

OProfile can profile a selected program or process or the whole system.

OProfile can also be used to collect cumulative event counts at the application, process, or system level.

Begin to show at:

```
$ man Oprofile

$ ophelp
```

ophelp lists the available performance counter options.

If you give it a symbolic event name, it will return the hardware value (e.g. "ophelp DATA_MEM_REFS").

> **note** **mk-project** use the version >= 1.0 of Oprofile.

And the available Oprofile programs are:

- **operf**
- **ocount**
- **opreport**
- **opannotate**
- **oparchive**
- **opgprof**

mk-project provides wrapper around this programs except **oparchive**.

Simply remember that operf and **ocount** generate a `profile_specification`.

And the other are done to interpret the datas.

> **warning** You must run this programs as root.

5.11 Valgrind targets

If valgrind is present on your system **mk-project** provide you 4 targets for the most common usage of valgrind:

```
make valgrind-memcheck    # Launch the valgrind memcheck tool on your binary.

make valgrind-cachegrind # Launch the valgrind cachegrind tool on your binary.

make valgrind-callgrind   # Launch the valgrind callgrind tool on your binary.

make valgrind-helgrind    # Launch the valgrind helgrind tool on your binary.
```

For every target you can set at creating the project or changing at reconfiguring your project the wanted options.

Note: Fell free *to edit the template to set your prefered options in hard coded*.

Or set the environment variable `$VALGRIND_OPTS`.

5.12 Alternative to *_OPTS

> **note** You can **export *_OPTS** the corresponding variable before launching the **make** target.

5.13 Documentation Source

- `GNU Make manual` (A very good manual from the GNU manuals serie).

  authors Stallman, McGrath, Smith.

- `C/C++ Compiling` (A very good book about libraries and machine code investigation).

  author Milan Stevanovic.

CHAPTER

SIX

MK-PROJECT CODE FORMATTERS

6.1 Introduction

mk-project provide severals utilities with many predefined targets for formatting your source code.

6.1.1 For C or C++ source code:

- **indent**
- **astyle**
- **bcpp**

6.2 Using the `indent` utility

mk-project provide following predefined indent styles:

```
make indent-kr    # Format all source files in the kr style.
make indent-gnu   # Format all source files in the gnu style.
make indent-linux # Format all source files in the linux style.
make indent-orig  # Format all source files in the original style.
make indent-user  # Format all source files in the user defined style.

make indent-clean # Remove all formatted files with suffix.
```

note The `indent-user` target use the given options during the project configuration for formatting your source code.

Note: By launching any code formatting target **mk-project** will output a copy of all your source files suffixed with the corresponding target name:

Per example by using the `indent-kr` target a file named `main.c` will ouput as `main-kr.c`.

For overwriting your source files really you must set the **`make`** variable **`OVERWRITE`** on the value **T**.

```
$ make indent-kr OVERWRITE=T
```

6.3 Using the `astyle` utility

mk-project provide following predefined indent styles:

```
make astyle-ansi          # Format all source files in the ansi style.
make astyle-java          # Format all source files in the java style.
make astyle-kr            # Format all source files in the kr style.
make astyle-stroustrup    # Format all source files in the stroustrup style.
make astyle-whitesmith    # Format all source files in the whitesmith style.
make astyle-banner        # Format all source files in the banner style.
make astyle-gnu           # Format all source files in the gnu style.
make astyle-linux         # Format all source files in the linux style.
make astyle-horstmann     # Format all source files in the horstmann style.
make astyle-lisp          # Format all source files in the lisp style.
make astyle-pico          # Format all source files in the pico style.
make astyle-python        # Format all source files in the python style.
make astyle-user          # Format all source files in the user defined style.

make astyle-clean         # Remove all formatted files with suffix.
```

note The `astyle-user` target use the given options during the project configuration for formatting your source code.

Note: By launching any code formatting target **mk-project** will output a copy of all your source files suffixed with the corresponding target name:

Per example by using the `astyle-kr` target a file named `main.c` will ouput as `main-kr.c`.

For overwriting your source files really you must set the **`make`** variable **`OVERWRITE`** on the value **T**.

```
$ make astyle-kr OVERWRITE=T
```

CHAPTER

SEVEN

MK-PROJECT CONTRIBUTING ADVICES

7.1 mk-project zen

mk-project zen is simple:

```
The minimum work for the user.

The maximum configuration possiblities.

+ Minimum informations asking to the user, maximum deduced.

+ Maximum possibilities, with a minimum informations and binaries.

Bring the maximum with the minimum.

! No package is obligatory except the coreutils
  and them needed by the programming language.
```

7.2 How contibute ?

Write a project for a programming language which isn't done, or enhance one.

Write some useful **make** targets for any purpose you want.

Write **make** targets forn your well know documentation generators, for **mk-project**.

Every help is welcome, thanks.

7.2.1 For writing a new project:

1. Simply fork the project.
2. Make a folder named `lang_mk-project` (per example: perl_mk-project).
3. Put your stuff inside this folder.
4. And ask for merging.

After your submission your project will be a mimimum tested and they is no matter of refusement only enhancement.

7.2.2 Makefile

You can take the included makefiles (`./.SubMakefiles/*.mk`)

To put it into your project, this is highly recommanded, **don't reinvent the wheel**.

We want targets for:

- Executing the source.
- debugging the source.
- profiling the source.
- And what you want else...

7.2.3 Scripting

We script into **bash** or **python** (the script must be compatible with **python2** and **python3**).

Or if your project is about a scripting language you can use this language.

> **warning** Think to modify the script `prepend_license.py` to adapt the comment sign from your language

```
It's easy even if you don't know python or

in the worse case i will do this for you.
```

Scripts are set into the `./.scripts` folder.

7.2.4 Becoming

If you create a **Makefile** project you become a **coauthor** of **`mk-project`**

If you enhance a project you become a **contributor** of **`mk-project`**

So if you submit a project for your well know language(s),

you will take first the benefit to get a **Project done** for **your programming language**.

And the proudness to contribute to **`mk-project`**

> **note** I will ensure the updating of the GUI at every new project adding.

7.2.5 NOTES

If you write the Makefile for your language, think at writing a minimal example project who writes

`hello world welcome to mk-project`

on `stdout`.

You can enhance your project with everything you want like the debugging definition in the C/C++ language,

and write entire module(s) for the project purpose.

7.3 Makefile

7.3.1 BINARIES

- Verify the presence of the binary using the function **BINARY_EXIST**.
- **UPPERCASE** the binary variable name for no confusion.

```
BINARY = ${call BINARY_EXIST, binary}
```

- test if binary installed with:

```
# Compare the ${BINARY} variable with an empty string.
ifneq (${BINARY}, )
# do work...
endif
```

note Binaries test are in file `./.SubMakefiles/binary_checks.mk`

7.3.2 VARIABLES

- Make the same for configuration:
  - use **T** for **TRUE**
  - use **F** for **FALSE**

```
# No comment on following line and remove trailing spaces.
OVERWRITE = T

# Compare the ${OVERWRITE} variable with T (don't insert  a space).
ifeq (${OVERWRITE},T)
# Do work...
else
# Do work...
endif
```

The configuration options set or select by the user must be at the top of the Makefile,

with a default value.

And you must inform me about in the goal to update the GUI properly.

- Use the assigments operators cleverly:

```
# define var     value  # Value definition (used for multiline).
# define var =   value  # indirect. (the value change at the next␣
↪assignment for the final variable value.)
# define var :=  value  # direct.   (the value doesn't change at the␣
↪next assignment for the final variable value.)
# define var ::= value  # retro and inter compatibility with other␣
↪make tools.
# define var +=  value  # increment assignment operator.
# define var ?=  value  # shell expansion operator.
```

- Use the increment operator (**+=**) cleverly so that the user can define the variable on the command-line.

```
USE_TABS += -t
```

Or **not**:

```
   override MY_VAR = value

#
```

note Take care by inserting comments some settings doesn't support comments on the same line as the variable.

7.3.3 Files

You can verify if a file exist or if it's generated by using the function **FILE_EXIST**

```
MY_FILE = ${call FILE_EXIST, /path/to/my_file.ext}
```

It will return **T** (TRUE) or **F** (FALSE) if the file exist or not.

7.3.4 FILES and FILEPATH

- First define all path relativ, included Makefiles are at the same position as the main Makefile.
- Define a variable for the **FILEPATH** and for the **FILE**.

```
MY_FILEPATH = ./filepath/...

MY_FILE = ${MY_FILEPATH}/my_file.txt
```

We construct the filepath relativ to the main Makefile: (./Makefile)

note Filepath are defined in file `./.SubMakefiles/path.mk`

- You can (not obligatory) put the extension in a variable, if this make sens.

```
EXT_TYPE = .type

MY_FILE = ${MY_FILEPATH}/${FILE}${EXT_TYPE}
```

- You can use the make function **FILE_EXIST** to verify the presence of a file.

```
MY_FILE = ${call FILE_EXIST, my_file}
```

note The included Makefiles are correctly named and end in the extension *.mk so that an editor can reconize them.

note The included Makefiles are set in the SubMakefiles folder.

7.3.5 LIBRARIES

Today most of the libraries use the program **`pkg-config`** which you can use to auto-detect the presence of a library.

By using the **PKG_CONFIG_EXIST** function.

```
HAS_LIB_PC =  ${call PKG_CONFIG_EXIST, thelibpc}
```

It will return **T** (TRUE) or **F** (FALSE) in relationship of the presence of a *.pc file for `thelibpc`.

7.3.6 TARGETS

If you need to compose some targets names from more than a word, separate them by:

- A **'-'** (*minus*) if it's a **user-target**.
- A **'_'** (*underscore*) if it's an **intern_target**.

  Which can be put together with others intern targets to form a **user-target**.

  note Don't forget the `.PHONY:` definition if the target has no depdending targets.

7.3.7 ADVICES

IMPORTANT: make doesn't support trailing spaces, so strip them.

You can use the following command

```
$ sed -i 's/[[:space:]]$//' filepath
```

7.4 code formatters

We can make usages of following utilities, for code formatting in severals languages:

7.4.1 C

- **`indent`** (checked).
- **`astyle`** (checked).
- **`bcpp`** (checked).
- **`uncrustify`** (not check, help me !).

7.4.2 C++

- **`indent`** (checked).
- **`astyle`** (checked).
- **`bcpp`** (checked).
- **`uncrustify`** (not check, help me !).

  note Must check if we can use this scripts by the **`universalindentgui`** authors or the tools author(s).

7.4.3 HTML

- **`tidy`** (not checked).

7.4.4 CSS

- **csstidy** (not checked).

7.4.5 Javascipt

- **JsDecoder.js** (not checked).

  note Must check if we can use this scripts by the **universalindentgui** authors or the tools author(s).

7.4.6 Perl

- **perltidy** (not checked).

7.4.7 PHP

- **phpStylist.php** (not checked).

  note Must check if we can use this scripts by the **universalindentgui** authors or the tools author(s).

7.4.8 Ruby

- **rbeautify.rb** (not checked).
- **ruby_formatter.rb** (not checked).

  note Must check if we can use this scripts by the **universalindentgui** authors or the tools author(s).

7.4.9 XML

- **xmlindent** (not checked).

7.4.10 Using a code formatter

The usage of a code formatter must be user defined controlled so that:

1. We ask the user if he wants to use it.
2. We make his usage conditionnaly in the corresponding Makefile: `./.SubMakefiles/code_formatter.mk`.

   By using a variable named **USE_(TOOL NAME UPPERCASE)** given the value:

   - **T** for `true` or
   - **F** for `false`.

   According the user settings.

CHAPTER

EIGHT

*.TODO OR *.TDO FILE FORMAT SPECIFICATIONS:

The *.todo specification give you a advice structure of how structuring an TODO file for efficiently tasks organizing, and so don't forget ideas or things which you may have to do in the future for the development of projects of any professional fields where tasks must be organized and be accomplished in an certain order.

8.1 Markup syntax

Syntax of a mark: `[UPPERCASE :Capitalize:  <digits>]`

Syntax of end mark: `[/UPPERCASE] # The end mark is good for reread his todo note.`

An entire ***.todo** file entry can be represent like this:

```
==================
TITLE OF DOCUMENT
==================

[TYPE :Priority_level: <TASK_ORDER>] Title of todo entry

  Todo main text...

[/TYPE]

[TYPE :Priority_level: <TASK_ORDER>] One line todo entry [/TYPE]
```

8.2 Markup Types

TYPE (can be):

8.2.1 Before complete the task:

- **BUG** (*A bug have to be fixed*).

- **FIXME** (*A problem has to be fixed*).
- **TEST** (*You have to test a feature*).
- **CORRECT** (*Something must be corrected*).
- **REDO** (*Something must be redone*).
- **COMMENT** (*You must make a comment*).
- **TODO** (*Something must be done*).
- **IDEA** (*You get an idea for something*).

8.2.2 After complete the task:

- **BUGFIX** (*The bug is fixed*).
- **FIXED** (*The problem is fixed*).
- **TESTED** (*The test is done*).
- **CORRECTED** (*The correction is done*).
- **REDONE** (*The task is rewritten*).
- **COMMENTED** (*The commenting is done*).
- **DONE** (*The task TODO is DONE*).
- **IDIE** (*The idea is complete (become true)*).

8.2.3 Summary of TYPE

- **BUG** or **BUGFIX**
- **FIXME** or **FIXED**
- **TEST** or **TESTED**
- **CORRECT** or **CORRECTED**
- **REDO** or **REDONE**
- **COMMENT** or **COMMENTED**
- **TODO** or **DONE**
- **IDEA** or **IDIE**

8.3 Priority_level

Priority_level (can take following values:)

- **High**
- **Medium**
- **Low**

8.4 TASK_ORDER

TASK_ORDER (can be)

1. a **2** (Maybe *3* or *4*) digits sequences for organizing.
2. a digit and a **UPPERCASE** letter with meaning:
   - By **TODO**, **TEST**, **IDEA** entries
     - **F** -> *Free time*
     - **N** -> *Normal* (When possible)
     - **U** -> *Urgent*
   - By **CORRECT**, **BUG** and **FIXME**
     - **I** -> *Info*
     - **W** -> *Warning*
     - **F** -> *Fatal*

     note the digit(s) are zero per default but it can take a value between 0-9 for very organized structures.

   note This can be omit. Only **TYPE** and **Priority_level** are mandatory.

   Summary The **TASK_ORDER** are written between < and >.

   Can be composed either of:

   - **2** digits representing the *task priority*.
   - A digit and a special mean **UPPERCASE** letter.

     syntax <[0-9][0-9|[F|N|U]|[I|W|F]]>

     example [TODO :Medium: <0F>] Make a new icon because actual is ugly !!! [/TODO]

8.5 Advices

***.todo** files extensibility:

1. Every entry **TYPE** can be invented but must be written in **UPPERCASE**.

   advice use only one word. (Else use **'_'**).

   DFY (*Don't Forget Yourself: this make sens*),

   DRY (*Don't Repeat Yourself: don't be stupid*),

   KISS (*Keep It Simple Stupid: be concise*).
2. Priority level can be added as long as they are one Capitalize word.
3. **DIY** (*Do It Yourself*) for the **TASK_ORDER** or in order to maintain them ordered.

   Advice Keep terminal width max 79 chars a line.

```
The Best for the End: Think at things like timestamps,

doing order, prerequisite for task, and so soon !!!
```

8.6 Syntax of *.todo file(s) content

You can use the `ReST` or `Markdown` syntax for the content between or inside the marks.

8.6.1 For Titles

```
==============
My first Title
==============

***************
My second title
***************

###############
My third title
###############

+++++++++++++++
My fourth title
+++++++++++++++

:::::::::::::::
My fifth title
:::::::::::::::

--------------
My sixth title
--------------

~~~~~~~~~~~~~~~
My seventh title
~~~~~~~~~~~~~~~
```

8.6.2 For text decorations:

```
**bold**

*italic*

_underline_

``inline literals``

--strike-trough--

^^over-line^^
```

8.6.3 For Lists:

```
+ List item 1
```

```
  - Sub list item 1

  - Sub list item 2

+ List item 2

  1. First numbered list item.

  2. Second numbered list item.

  3. Third numbered list item.

+ List item 3

  Definition list title

    Definition list text
```

8.6.4 For keywords values pairing:

```
:author: foo bar

:license: fdl

:version: 1.0.0
```

8.6.5 For links:

```
`Link text <http://www.domain.com/folder/file.html>`
```

8.6.6 For footnotes:

```
[*] my footnote text
```

8.6.7 For comments:

```
# My comment line
```

8.6.8 For code text:

```
[:LANGUAGE:]

  Indented text is code !
```

Per example for C code:

```
[:C:]

  const char *var = "value" ;
```

8.7 End word of specifications of the *.todo file(s) format

Do what you must with this specifications and take it like an TODO file
structuring advice, but this document was establish to define the
specifications of a clean TODO file.

8.8 Example of a *.todo file:

An example from a real *.todo file from one of my projects.

```
IT-EDIT TODO:
++++++++++++

[IDEA :Low:] Advertisement for it-edit:

  it-edit provide so many schemes (more than the underlying library per default)␣
↪because per example the

  emacs scheme support italic for the ReST or Markdown language which the kate scheme␣
↪doesn't support.

  But I think the kate scheme is more adapt, with his settings, for program source␣
↪code writing in terms of syntax coloration.

  And the emacs theme is better to use for ReST per example, because of better syntax␣
↪coloration of italic.

  So better get 2 schemes, which you can easily switch, than missing a feature.

[/IDEA]

Editor
======

[IDEA :High:] Make the text-completion configurable.

  1. The text completion is one per file.

  2. The text-completion is one for all files (**not easy**).

[/IDEA]

[TODO :High:] Make the regex replacement become true. (See GLib regex) [/TODO]

[IDEA :Medium:] What about enable/disable spell-check ? [/IDEA]

Schemes
-------
```

```
[BUG :High:]

They is a highlight problem with the search all highlight with emacs schemes.

[/BUG]


[IDEA :low: <0F>]

Think of schemes pairs like:

  + kate && emacs (bg white)

  + cobalt && turbo (bg blue)

  + tango && classic (Are settings defendant).

  + vsdark && oblivion (bg maroon).

  + slate && solarized-dark. (bg turquoise)

  + build && solarized-light (bg light yellow).

  + matrix (standalone).

[/IDEA]

Terminals
=========

[IDEA :Medium:] Maybe a (main  start) settings individually for every different␣
↪terminals and/or a main (main  start) settings configuration. [/IDEA]

[TODO :Medium:] Open a file into an editor tab with a terminal pattern [/TODO]

Files
=====

[TODO :Medium:] Add a ChangeLog entry !

  The clipboard from the terminals has been upgraded from severals functionalities.

[/TODO]
```

8.9 License

```
Copyright (c)  2016,2017  Brüggemann Eddie.

Permission is granted to copy, distribute and/or modify this document

under the terms of the GNU Free Documentation License, Version 1.3

or any later version published by the Free Software Foundation;
```

```
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU

Free Documentation License".
```

CHAPTER

NINE

ABOUT MK-PROJECT

author Eddie Brüggemann <mrcyberfighter@gmail.com>

documenter Eddie Brüggemann <mrcyberfighter@gmail.com>

9.1 A word from the author

I must recognize to write a program which **generates** and **parse** *severals files* is **painfull** in the **C** *programming language*.

But I hope *that the community will adopt this usefull tool...*

I dislike I.D.E (Integrated Development Environment)'s because their advantages is their weak point:

They let you make forget everything once you have configurate their interface.

Even how to build your program (i.e. The command line to build your program, you know it ?).

I use the commandline everyday and by doing a **good compromise** between **automating task** and doesn't forget **how the command works**.

Is issue **mk-project**...

The adding of the `Edit terminals` is suppose for **`ed`**, **`vi`**, **`emacs`**, etc users.

And the G.U.I **`make`** targets launching can be extend *like explain in the presentation*.

Finally: I hope you will join us to make **mk-project** support more and more programming languages.

note I have put all my *savoir-faire* in this project for you and the entire community.

9.2 Dependencies

9.2.1 Libraries

- `libgtk-3-dev`
- `libvte-2.91-dev`
- `libxml2-dev`

9.2.2 Main program

- The **make** program.
- coreutils

9.2.3 Documentation

- python(3)-sphinx
- python(3)-docutils

9.2.4 Debugging

- binutils
- libc-bin
- findutils
- file
- size
- hexdump

  note Only required if you make usage of them, else the corresponding target won't be available.

9.2.5 Code formatters

- indent
- astyle
- bcpp

  note Only required if you make usage of them, else the corresponding target won't be available.

9.2.6 Internationalisation

- gettext

  note Only required if you make usage of them, else the corresponding target won't be available.

9.2.7 Documentation Source

- GNU Make manual (A very good manual from the GNU manuals serie).

  authors Stallman, McGrath, Smith.
- C/C++ Compiling (A very good book about libraries and machine code investigation).

  author Milan Stevanovic.
- Writing efficient C code.

  author Jonas Skeppstedt (author of the compiler *ISO Certicated and Validated* **lmpcc**).

ISO (ISO/IEC 9899:19999, C language) conform compiler list

- EDG C/C++ 3.0.1, december 2002.
- lmpcc C99 Compiler for Linux / PowerPC 1.3, july 2003.
- Sun studio 9, May 2004.
- IBM VAC 6.0.0.8, October 2004.

  note No **`gcc`** neither **`clang`** are certified to be fully compliant with it.

9.2.8 THANKS

- Dennis M Ritchie, for UNIX and C.
- Richard Stallman, for gcc and the F.S.F movement.
- Ken Tompson, for UNIX.
- Linus Tornvalds, for Linux and git.
- And to every worker for a better world...

9.2.9 Author final word:

I use **`mk-project`** since the version **1.0** (spring *2016*) for my programs.

Accompanier with my terminals integrated editor it-edit,

where I type my targets instead of using vim or any other T.U.I TERMINAL USER INTERFACE.

I must confess that I do not use all the targets provided by mk-project.

My most used targets are:

```
$ make

$ make -B

$ make exec

$ make fdebug

$ make gdb

$ make search-pattern argv="pattern"

...
```

CHAPTER

TEN

MK-PROJECT GTK3 TYPES

mk-project implement some few derivate Widgets, which I will present here.

You can take a look at the source located in the sub-folders from `/usr(/local)/share/mk-project/src`.

To learn how to implement this kind of Widgets:

note Here you can sea how a sphinx documentation looks like, with this theme, for C code (c++ code can be documented too) with sphinx.

10.1 GtkSmartIconButton

A simple button with an icon without label and tool-tip which embed an universal short-cut text.

GtkWidget* gtk_smart_menu_item_new_all(const gchar *label, const gchar *icon_filepath, Gtk

Parameters

- **label** (`const gchar *`) – The label to display into the menu item.
- **icon_filepath** (`const gchar *`) – The menu item icon file-path.
- **accel_group** (`GtkAccelGroup *`) – The shortcut accelerator group.
- **accel_modifier** (`const GdkModifierType`) – The shortcut modifier.
- **accel_key** (`const guint`) – The shortcut accelerator key.

Return type `GtkWidget*`

Returns A pointer to the `GtkSmartMenuItem`.

GtkWidget* gtk_smart_check_menu_item_new_all(const gchar *label, const gboolean draw_as_ra

Parameters

- **label** (`const gchar *`) – The label to display into the menu item.
- **draw_as_radio** (`const gboolean`) – draw_as_radio
- **icon_filepath** (`const gchar *`) – The menu item icon file-path.
- **accel_group** (`GtkAccelGroup *`) – The shortcut accelerator group.
- **accel_modifier** (`const GdkModifierType`) – The shortcut modifier.
- **accel_key** (`const guint`) – The shortcut accelerator key.

Return type `GtkWidget*`

Returns A pointer to the `GtkSmartMenuItem` check button.

GtkWidget* gtk_smart_radio_menu_item_new_all(const gchar *label, const gchar *icon_filepath

Parameters

- **label** (const gchar *) – The label to display into the menu item.
- **draw_as_radio** (const gboolean) – draw_as_radio
- **icon_filepath** (const gchar *) – The menu item icon file-path.
- **accel_group** (GtkAccelGroup *) – The shortcut accelerator group.
- **accel_modifier** (const GdkModifierType) – The shortcut modifier.
- **accel_key** (const guint) – The shortcut accelerator key.
- **widget** (NULL or GtkWidget*) – A member from the radio button group.

Return type GtkWidget*

Returns A pointer to the GtkSmartMenuItem radio button.

Note: You can pass a NULL pointer or 0 to the parameters :

- icon_filepath
- accel_group
- accel_modifier
- accel_key.
- widgets.

---

note You can build others constructors if you have understand How-To build this kind of widgets.

10.1.1 Getters

GtkWidget* gtk_smart_menu_item_get_image(GtkWidget* smart_menu_item) ;

Parameters

- **smart_menu_item** (GtkWidget*) – The return value from the constructors.

Return type GtkWidget*

Returns A pointer to the GtkImage widget.

GtkWidget* gtk_smart_menu_item_get_menuitem(GtkWidget* smart_menu_item) ;

Parameters

- **smart_menu_item** (GtkWidget*) – The return value from the constructors.

Return type GtkWidget*

Returns A pointer to the GtkMenuItem widget.

GtkWidget* gtk_smart_menu_item_get_label(GtkWidget* smart_menu_item) ;

Parameters

- **smart_menu_item** (GtkWidget*) – The return value from the constructors.

Return type GtkWidget*

Returns A pointer to the GtkLabel widget.

GtkWidget* gtk_smart_menu_item_get_accel_label(GtkWidget* smart_menu_item) ;

Parameters

- **smart_menu_item** (GtkWidget*) – The return value from the constructors.

Return type GtkWidget*

Returns A pointer to the GtkAccelLabel widget.

10.2 GtkSmartIconButton

A simple button with an icon without label and tool-tip which embed an universal short-cut text.

10.2.1 Constructors

GtkWidget* gtk_smart_icon_button_new_all(const gchar *filepath, const gchar *tooltip_text,

Parameters

- **filepath** (const gchar *) – The filepath to the image to use as icon.
- **tooltip_text** (const gchar *) – The tool-tip text without the accelerator label.
- **accel_key** (const guint) – The shortcut accelerator key.
- **accel_modifier** (const GdkModifierType) – The shortcut modifier.

Return type GtkWidget*

Returns A pointer to the GtkSmartIconButton widget.

GtkWidget* gtk_smart_icon_toggle_button_new_all(const gchar *filepath, const gchar *tooltip

Parameters

- **filepath** (const gchar *) – The filepath to the image to use as icon.
- **tooltip_text** (const gchar *) – The tool-tip text without the accelerator label.
- **accel_key** (const guint) – The shortcut accelerator key.
- **accel_modifier** (const GdkModifierType) – The shortcut modifier.

Return type GtkWidget*

Returns A pointer to the GtkSmartIconButton toggle button widget.

note This widget is not used into **mk-project** but provided in the hope to be useful.

10.3 GtkTermTab

A GtkNoteBook tab with an decorative icon, a label, and close icon button.

10.3.1 Constructor

GtkWidget* gtk_mk_term_tab_new(const gchar *icon_filepath, const gchar *label, const gchar

Parameters

- **icon_filepath** (const gchar *) – Image filepath to display as decoration on the right of the tab-label.
- **label** (const gchar *) – The label to display in the GtkMkTermTab.
- **close_filepath** (const gchar *) – Image filepath to display in the GtkMkTermTab as close button icon.

Return type GtkWidget*

Returns A pointer to the Widget GtkMkTerm.

10.3.2 Getters

GtkWidget* gtk_mk_term_tab_get_close_button(GtkMkTermTab *tab);

Parameters

- **tab** (const gchar *) – An instance of the GtkMkTermTab.

Return type GtkWidget*

Returns A pointer to the Widget GtkButton at the right of the label.

10.4 GtkMkTerm

warning This widget implementation is not reusable as is, because of VteTerminal configuration variables.

10.4.1 Constructor

GtkWidget* gtk_mkterm_new(gboolean initialize, gboolean is_edit_term) ;

Parameters

- **initialize** (gboolean) – Initializing or reconfiguring the GtkMkTerm.
- **is_edit_term** (gboolean) – Whether or not the GtkMkTerm is a editor widget.

Return type GtkWidget*

Returns A pointer to the Widget GtkMkTerm.

10.4.2 Getters

GtkWidget* gtk_mkterm_get_vte_terminal(GtkWidget* mkterm) ;

Parameters

- **mkterm** (GtkWidget*) – An instance of the GtkMkTerm Widget.

Return type GtkWidget*

Returns A pointer to the `VteTerminal`.

`GtkWidget* gtk_mkterm_get_clipboard_menu(GtkWidget* mkterm) ;`

Parameters

- **`mkterm`** (`GtkWidget*`) – Initializing or reconfiguring the `GtkMkTerm`.

Return type `GtkWidget*`

Returns A pointer to the `GtkMenu` associated to the `GtkMkTerm`.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

$VALGRIND_OPTS, 26

E

environment variable
    $VALGRIND_OPTS, 26
    VALGRIND_OPTS, 10

V

VALGRIND_OPTS, 10