NAME

CLIFM − The Command Line File Manager

SYNOPSIS

clifm [OPTIONS] [PATH]

INDEX

1. GETTING HELP

There are several ways of getting help in CliFM. Once in the program, enter '**?**' to see some basic usage examples, or press **F1** to access this manpage, **F2** to go to the **COMMANDS** section of this very manpage, or **F3** to go to the **KEYBOARD SHORTCUTS** section.

Help for all internal commands can be accessed via the *-h* or *--help* options. For example, *s -h* or *s --help*.

A convenient way of getting full information about CliFM commands is via the '**ih**' action, bound by default to the interactive help plugin (*ihelp.sh*). Just type 'ih' to run the plugin (it depends on *FZF*) and select the command you want to obtain information about.

For a quick introduction jump to the **EXAMPLES** section at the bottom of this document.

2. DESCRIPTION

CliFM is a **C**ommand **L**ine **I**nterface **F**ile **M**anager. This is its main feature and strength: all input and interacion is performed via commands typed in a prompt. In other words, CliFM is a REPL, since it's basic structure is simply this: **R**ead (user input via a command line), **E**valuate/Execute the command, **P**rint the results, **L**oop (start all over again).

Unlike most terminal file managers out there, indeed, CliFM replaces the traditional TUI interface (also known as curses or text-menu based interface) by a simple command-line interface (REPL). In this sense, it is a file manager, but also **a shell extension**: search for files, copy, rename, and trash some of them, but, at the same time, update/upgrade your system, add some cronjob, stop a service, and run nano (or vi, or emacs, if you like).

Simply put, with CliFM the command-line is still there, never hidden, but enriched with file management oriented functionalities.

3. FEATURES

CliFM is a completely text−based, **shell−like** file manager for the terminal able to perform all the basic operations you may expect from any other FM. However, its most distinguishing characteristics are:

a) An extensive and customizable list of **color codes** to easily identify file types and extensions.

b) The use of **short (and even one−character) commands**, and **entry list numbers (ELN's)** instead of file names. Enter 'o 12', for instance, to open a file with your default text editor or to change to the desired directory. If the **autocd** and **auto-open** functions are enabled, which is the default (see below *ac* and *ad* commands), you only need to enter the ELN to open the file or change to directory: instead of 'o 12', just enter '12'.

With the **automatic ELN expansion** feature you can use ELN's with external commands as well. 'diff 12 5', for example, will run 'diff' over the files corresponding to ELN's 12 and 5. **Ranges** are also accepted, for example: 'rm 1−12' or 'r 1−12'. Since numbers could be a bit tricky when it comes to listed files, **TAB completion** is available for ELN's and ELN ranges, just as for many other things (see the **BUILT-IN EXPANSIONS** section below). *Gemini*, a powerful **auto-suggestions** system (heavily inspired by the Fish shell), was developed with this very purpose in mind.

c) Automatic files listing: Unlike the shell 'cd' command, CliFM's built−in *cd* function automatically lists files in the new directory.

d) Several shell capabilities: Fish−like auto−suggestions that, together with TAB completion, syntax highlighting, and history (for both commands and visited directories) assists the user when it comes to typing commands. This feature includes a warning prompt to warn the user when typing an invalid command name.

e) Bookmarks: keep a list of your favorite directories, and even files, to get easy access to them. Example: enter 'bm' (or press Alt−b) to open the bookmarks screen and then simply type a number or a shortcut name to access the desired bookmark.

f) The **Selection Box** allows you to drop files and directories from different parts of the file system (even from different instances of CliFM) and then operate on them with just one command. Example: select a few files in one instance of the program (say 's 12; cd /media; s 2 5−6') and then paste them somewhere else using a second instance ('v sel', 'paste sel' or Ctrl-Alt-v). Both wildcards (globbing) and regular expressions are supported. Inverse matching is also allowed for patterns.

g) The **directory history** function (*back* and *forth*) keeps track of all visited directories, so that you can go back and forth to any of them by just entering 'b' or 'f' (or just pressing 'Alt−j' and 'Alt−k', or Shift−Left and Shift−Right, respectively). Another way of quickly navigating through visited directories is using *Kangaroo*, a built-in **directory jumper** function. See the *j* command below.

h) An extensive list of **keyboard shortcuts** make it even easier and faster to navigate and operate on your files. For example, instead of typing 'cd ..' to go back to the parent directory, or 's *' to select all files in the current directory, you can simply press Alt−u (or Shift−Up) and Alt−a respectively.

i) The **quick search** function makes it really easy to quickly find the files you are looking for: just enter a slash followed by the string or regular expression you want (ex: '/myfile' or '/.*.png'), that's all. Inverse search is also allowed by prepending an exclamation mark (!) to the search pattern.

j) Plugins: Build your own custom commands using executable files (shell scripts or binaries). Just drop an executable file (all languages are supported) in the plugins directory, tell CliFM to use this executable file via a custom action name, and use it as any other command.

k) Files preview: Via the *fzfnav* plugin (see the **PLUGINS** section below), CliFM is able to preview several kinds of files, including not only text files, but also GIF's, images, office documents, PDF's, and more. This plugin requires *FZF* to be installed. For basic images preview *ueberzug* is required. A list of optional dependencies to preview different kinds of files is available in the **PLUGINS** section.

l) It is blazing **fast** and incredibly **lightweight**. With a memory footprint below 5 MiB and a disk usage of 0.5 MiB, it can run on really ancient hardware. It is so simple that it doesn't require an X session nor any graphical environment at all, being able thus to perfectly run on the console (TTY) and even on a headless machine via a SSH or any other remote session. And if this is not enough, you can still try the **light mode** to make it even faster.

m) When running in **stealth mode**, it will leave not trace at all on the host system.

Because inspired by the **KISS** principle, CliFM is fundamentally aimed to be lightweight, fast, simple, and efficient. On Archlinux's notion of simplicity see:
https://wiki.archlinux.org/index.php/Arch_Linux#Simplicity

List of CliFM features:

1) FreeDesktop compliant Trash system
2) Automatic files listing
3) Commands and directory history function
4) TAB completion/expansion for ELN's, ELN ranges, commands, paths, bookmarks, profiles, color schemes, command history, directory history (via the *jump* command), remote resources, sort methods, selected files, the 'sel' keyword, trashed files, plus the deselect and the open-with commands (*ow*). There are two available modes: standard and FZF (see the **BUILT−IN EXPANSIONS** section below)
5) More than 40 customizable keyboard shortcuts

6) Wildcards and regex auto−expansion
7) Braces auto−expansion
8) ELN's auto−expansion
9) ELN ranges auto−expansion
10) 'sel' keyword auto−expansion
11) Bookmarks auto−expansion
12) Bash−like quoting system
13) Custom aliases and variables
14) Full theming support via a single configuration file
15) Shell commands execution
16) Startup and prompt commands
17) User profiles
18) Commands sequential and conditional execution
19) *Lira*, a built-in resource opener
20) Eleven sorting methods and reverse sorting
21) Bulk rename and bulk remove
22) Archiving and compression support
23) Auto−cd and auto−open
24) Plugins support via custom actions linked to executable files
25) Symbolic links editor
26) Regular expressions, file type filter, and inverse matching support for both search and selection functions
27) Command substitution and regular expressions for all internal commands
28) Privacy oriented
29) *Kangaroo*, a built−in directory jumper function
30) Icons support (depends on icons−in−terminal: *https://github.com/sebastiencs/icons−in−terminal*)
31) Batch symbolic links
32) Files filter via regular expressions for the files list
33) Up to eight workspaces
34) Fused parameters support for ELN's
35) Advcpmv support (cp and mv with a nice progress bar). Depends on advcpmv (See *https://github.com/jarun/advcpmv*)
36) Fastback function
37) Remote file systems management via the *net* command
38) *Gemini*, a Fish-like auto-suggestions system (including a warning prompt to highlight invalid command names)
39) Syntax highlighting
40) Interactive rename functionality for the *m* command
41) Easily mount/unmount storage devices via the *media* command
42) Control settings on a per directory basis via the *autocommands* function
43) *Bleach*, a built-in file names cleaner
44) Backdir: quickly change to a parent directory
45) Secure environment
46) Secure commands
47) URI file scheme support (file://)
48) URL support when running as standalone resource opener (see the *--open* option)
49) Disk usage analyzer
50) *Etiqueta*, a files tagging system

4. POSITIONAL PARAMETERS

If the first non-option parameter is a directory, CliFM will start in this directory. Otherwise, if not a directory, it will open the file via the default associated application and exit (working thus as a stand-alone resource opener). URL's and the URI file scheme (for local file systems) is supported.

For example, by running 'clifm /etc/hosts', the *hosts* file will be opened and CliFM will immediately exit. In the same way 'clifm https://some_domain' will open this web resource with the application associated to the *text/html* MIME type.

On the other side, the command 'clifm /etc' instructs CliFM to start in the directory */etc*. If not specified, the first workspace will be used. To start in a different workspace use the *−w* option. For instance: 'clifm −w4 /etc'.

5. OPTIONS

−a, −−no−hidden
do not show hidden files (default)

−A, −−show−hidden
show hidden files

−b −−bookmarks−file=FILE
set an alternative bookmarks file

−c −−config−file=FILE
set an alternative configuration file

−D −−config−dir=DIR
use *DIR* as an alternative configuration directory. If configuration files do not exist already, they will be created anew here

−e, −−no−eln
do not print ELN's (entry list number) at the left of file names. Bear in mind, however, that though ELN's are not printed, they are still there and can be used as always

−f, −−no−folders−first
do not list folders first

−F, −−folders−first
list folders first (default)

−g, −−pager
enable *Mas*, the built-in pager for files listing

−G, −−no−pager
disable the pager (default)

−h, −−help
show this help and exit

−H, −−horizontal-list
list files horizontally instead of vertically ordered

−i, −−no−case−sensitive
no case−sensitive files listing (default)

−I, −−case−sensitive
case−sensitive files listing

−k −−keybindings−file=FILE
set an alternative keybindings file

−l, −−no−long−view
disable long view mode (default)

−L, −−long−view
enable long view mode to list files and their properties. For each file, the following information is provided: file name, file properties (symbolic notation), owner, group, last modification time, and size. To get more detailed information use the *p* command (see below).

−m, −−dirhist−map
enable the directory history map to keep in view previous, current and next entries in the directory history list

−o, −−no−autols
´cd´ works like the shell 'cd' command: change directory but do not list files automatically

−O, −−autols
´cd´ changes directory and lists files automatically (default)

−p, −−path=PATH
use *PATH* as CliFM starting path. Default starting path is the current working directory. If no workspace is specified via the **--workspace** option (see below), the first workspace (1) is used. This option is deprecated: use positional parameters instead.

−P, −−profile=PROFILE
use *PROFILE* as profile. If *PROFILE* does not exist, it will be created. Default profile is 'default'

−s, −−splash
enable the splash screen

−S −−stealth-mode
leave no trace on the host system. Nothing is read from files nor any file is created: all settings are set to the default value. However, most settings can still be controlled via command line options. Listing colors could be customized via dedicated environment variables (see the **ENVIRONMENT** section below). Take a look as well to the *history* command and the *--no-history* command line switch.

−t, −−disk−usage−analyzer
run in disk usage analyzer mode. Equivalent to *--sort=size --long-view --sort-reverse --full-dir-size --no-folders-first*. The total size of the current directory, plus the name and size of the largest file will be printed after the list of files. Sizes are calculated using actual device usage (rather than apparent size) in powers of 1024 (KiB, MiB, etc). To use apparent sizes instead add the *--apparent-size* switch.

−u, −−no−unicode
disable unicode

−U, −−unicode
enable unicode to correctly list file names containing accents, tildes, umlauts, non−latin letters, etc. This option is enabled by default.

−v, −−version
show version details and exit

−w, −−workspace=NUM
start in workspace *NUM*. By default, CliFM will recover the last visited directory for each workspace. However, you can override this behaviour using positional parameters, as described above, to start in workspace *NUM* and in path *PATH*.

−x, −−no−ext−cmds
disallow the use of external, shell commands

−y, −−light−mode
enable the light mode to speed up CliFM. See the **NOTE ON SPEED** section below.

−z, −−sort=METHOD
sort files by *METHOD*, **where** *METHOD* could be one of: 0 = none, 1 = name, 2 = size, 3 = atime, 4 = btime (ctime if not available), 5 = ctime, 6 = mtime, 7 = version (name if not available), 8 = extension, 9 = inode, 10 = owner, 11 = group. Both numbers and strings are allowed. E.g: --sort=9, --sort=inode.

−−apparent−size
use apparent sizes instead of actual device usage

−−case−sens−dirjump
do not ignore case when consulting the jump database (via the *j*command)

−−case−sens−path−comp
enable case sensitive path completion

−−cd−on−quit
write last visited directory to *$XDG_CONFIG_HOME/clifm/.last* to be later accessed by the corresponding shell function at program exit. See the **SHELL FUNCTIONS** section below.

−−color−scheme=NAME
set *NAME* as color scheme

−−no-control−d−exit=NAME
do not allow exit on EOF (Control-d)

−−cwd−in−title
print current working directory in terminal window title. Otherwise, only the program name is printed

−−disk−usage
show disk usage (free/total) for the file system the current directory belongs to

−−enable−logs
enable program logs. See the *log* command for more information

−−expand−bookmarks
expand bookmark names into the corresponding bookmark paths and enable TAB completion for bookmark names as well. If the bookmark is, .e.g. mybookmark=/my/path, "mybo" will be completed as "mybookmark", which will be interpreted then as "/my/path"

−−full−dir−size
if running in long view, print directories size and their contents

−−fzftab
enable the FZF mode for TAB completion (See the **BUILT−IN EXPANSIONS** section below)

−−icons
enable icons. Depends on the **icons-in-terminal** project.

−−icons−use−file−color
instead of an specific color, icons take the color of the corresponding file name (specified either via file type or via file extension). Useful when building custom color schemes. This option implies **--icons**.

−−list−and−quit
list files and quit. It may be used in conjunction parameter substitution. Ex: 'clifm --list-and-quit /etc'

−−mnt−udisks2
use *udisks2* instead of *udevil* (default), for the *media* command

−−max−dirhist
maximum number of visited directories to remember

−−max−files=NUM
list only up to *NUM* files. Use -1 to remove the files limit (default). See the *mf* command for a more detailed description

−−max−path=NUM
set the maximum number of characters after which the current directory in the prompt line will be abbreviated to the directory base name (if \z is used in the prompt)

−−no−cd−auto
by default, CliFM changes to directories by just specifying the corresponding ELN (e.g. '12' instead of 'cd 12'). This option forces the use of *cd*

−−no−dir−jumper
disable the directory jumper function

−−no-classify
by default, CliFM appends a file type indicator character to file names when running with no colors (see the **--no-color** option below) and both a directory indicator and a files counter for directories when running with colors. Classification characters are:
/n: directories (n = files counter)
@: symbolic links
|: fifo/pipes
=: sockets
*: executable files
?: unknown file type Use this option to disable this file type classification.

−−no−clear−screen
do not clear the screen before listing files

−−no−color
disable colors

−−no−columns
disable columns for files listing

−−no−file−cap
do not check files capabilities when listing files

−−no−file−ext
do not check files extension when listing files

−−no−files−counter
disable the files counter for directories. This option is especially useful to speed up the listing process; counting files in directories is particularly expensive

−−no−follow−symlink
do not follow symbolic links when listing files

−−no−highlight
disable syntax highlighting. To customize highlighting colors see the **COLOR CODES** section below

−−no−history
do not write commands into the history file

−−no−open−auto
same as no-cd-auto, but for files instead of directories

−−no-restore−last−path
by default, CliFM saves the last visited directory for each workspace to be restored in the next session. Use this option to disable this behavior.

−−no-suggestions
disable the auto-suggestions system

−−no−tips
disable startup tips

−−no−warning−prompt
disable the warning prompt (used to highlight invalid command names)

−−no−welcome−message
disable the welcome message

−−only−dirs
list only directories

−−open=FILE
run as a stand-alone resource opener: open *FILE* and exit, where FILE could be a regular file or a directory, using either standard notation (/dir/file) or the URI file scheme (file:///dir/file]), or a URL (www.domain or https://domain).

−−opener=APPLICATION
specify a resource opener to use. If *opener* is not set, *Lira* will be used instead

−−print−sel
always print the list of selected files. Since this list could be quite extensive, the maximum number of selected files to print could be specified via the **MaxPrintSelfiles** option in the configuration file. Defaults to 0 (auto, i.e. never take more than half terminal height). Use -1 to remove the limit or any other positive value.

−−rl−vi−mode
set readline to vi editing mode (defaults to emacs editing mode)

−−secure−cmds
Filter commands passed to the OS to mitigate command injection attacks. Consult the **SECURITY** section below

−−secure−env
run CliFM in a secure environment (regular mode). Consult the **SECURITY** section below

−−secure−env−full
run CliFM in a secure environment (full mode). Consult the **SECURITY** section below

−−share−selbox
make the Selection Box common (that is, accessible) to different profiles. By default, each profile has a private Selection Box.

−−si print sizes in powers of 1000, as defined in the International System of Units (SI), instead of 1024 (Linux only)

−−sort−reverse
sort in reverse order, for example: z-a instead of a-z, which is the default order

−−trash−as−rm
the *r* command executes the built-in 'trash' (see the *t* command) instead of **rm**(1) to prevent accidental deletions

Options precedence order: 1) command line; 2) configuration file; 3) default values.

6. COMMANDS

Help for all commands listed here can be accessed via the *-h* or *--help* options.

NOTE: ELN = Entry List Number. Example: in the line "12 openbox" (when listing files), 12 is the ELN corresponding to the file named "openbox". The slash followed by a number (/xx) after directories and symbolic links to directories (the files counter) indicates the amount of files contained by the corresponding directory, excluding self and parent directories ("." and ".." Respectively).

NOTE 2: In case of ELN-filename conflict the backslash can be used to prevent ELN expansion. For example, if we have at least two files and one of them in named "2", then CliFM cannot know in advance if the command refers to the ELN 2 or to the file name "2". In we want the ELN, we just write the ELN number, for example: 's 2'. But if we want the file name, we need to escape the file name using the

backlash character: 's \2'.

NOTE 3: CliFM supports **fused parameters** for internal commands taking an ELN or range of ELN's as parameters. Much like short options for command line programs, you can drop or omit the space between internal commands and the corresponding ELN passed as argument. In general, you can write 'CMDELN' instead of 'CMD ELN'. For example: 'o12' or 's1-5' instead of 'o 12' and 's 1-5' respectively. Bear in mind, however, that in thus omitting the space char TAB completion for ELN's won't be available. If there is a file named 'o12' (more generally, CMDELN), and if you want to refer to this file instead of a CliFM command, escape the file name to prevent the split; for example: 's \o12'.

NOTE 4: CliFM implements a **fastback** function, that is to say, the conversion of "... n" or "cmd ... n" into "../.. n" or "cmd ../.. n". In other words, subsequent dots after ".." will be converted each into "/..". For example, if you are in your home directory and type "..." or "cd ...", and since "..." amounts to "../..", you will be taken to the root directory. TAB completion is available for the fastback function: "....bin" -> TAB -> "../../../bin".

FILE/DIR
: if the *autocd* and *auto-open* functions are enabled, which is the default value, open FILE or change directory to DIR. In other words, 'FILE' amounts to 'open FILE' or 'o FILE', and 'DIR' to 'cd DIR'. ELN's, of course, are allowed. Example: '12'.

/PATTERN [−filetype] [-x] [DIR]
: this is the quick search function. Just type '/' followed by a glob or regular (or extended regular) expression, and CliFM will list all matches in the current working directory. For example, both '/*.pdf' and '/.pdf$' expressions will list all PDF files in the current working directory, the former using wildcards, and the second a regular expression.

  Bear in mind that search patterns (if contain no metacharacter) are evaluated first as glob expressions, and then, if no matches are found, as regular expressions. Expressions containing no pattern metacharacter are automatically transformed into a regular expression: '/.*EXP.*'. For example '/test' becomes '/.*test.*'.

  By default, regular expressions (and thereby expressions containing no metacharacter) are case insensitive (glob expressions, by contrast, are always case sensitive). However, you can enable case sensitive search by setting the *CaseSensitiveSearch* option to true in the configuration file.

  To search for files in any directory other than the current one, specify the directory name as a further argument. This argument (DIR) could be an absolute path, a relative path, or an ELN. For example, enter '/^A 7' to search for all files starting with 'A' in the directory corresponding to the ELN 7.

  The result of the search could be further filtered by specifying a filter type: −d, −r, −l, −s, −f, −b, and −c (directory, regular file, symbolic link, socket, FIFO/pipe, block device, and character device respectively. For example, '/[.-].*d$ −d Documents/' will list all directories containing a dot or a dash and ending with 'd' in the directory named "Documents".

  The quick search function also supports invert search: prepend the exclamation mark (!) to negate a given search pattern. For example: "!.*s$ -d /etc" will match all directories in /etc NOT ending with 's', just as "!D*" will match all files in the current directory NOT starting with 'D'.

  To perform a recursive search use the -x parameter, and, optionally, a search path (DIR). The search will be performed using *find* as follows: 'find DIR -name PATTERN'. If no search path is provided, the search is executed starting in the current directory. Otherwise, the search starts in DIR.

;[CMD], **:**[CMD]
If no CMD, run the system shell in the current working directory. If CMD is specified, skip all CliFM expansions (see the **BUILT−IN EXPANSIONS** section below) and run the input string (CMD) as is via the default system shell.

ac, ad ELN/FILE ... n
archive/compress and dearchive/decompress one or multiple files and/or directories. The archiver function brings two modes: *ac*, to generate archives or compressed files, and *ad*, to decompress or dearchive files, either just listing, extracting, recompressing, or mounting their content. In this latter case, the mountpoint used automatically is *$HOME/.config/clifm/PROFILE/mounts/ARCHIVE_NAME*.

The function accepts single and multiple file names, wildcards, ELN ranges, and the 'sel' keyword. For example: 'ac sel', 'ac 4−25 myfile', or 'ad *.tar.gz'. Multiple archive/compression formats are supported, including Zstandard. When it comes to ISO 9660 files only single files are supported.

The archive mount function for non ISO files depends on **archivemount**, while the remaining functions depend on **atool** and other third−party utilities for achive formats support, for example, **p7zip**. **p7zip** is also used to manage most decompressing options for ISO 9660 files, except for mount, in which case **mount(8)** is used. Creation of ISO files is done via **genisoimage(1)**. For more information consult **atool(1)**, **archivemount(1)**, **zstd(1)**, and **7z(1)**.

acd, autocd [on, off, status]
toggle the autocd function on/off. If set to on, 'DIR' amounts to 'cd DIR'.

actions [edit [APP]]
with no argument, lists available custom actions (or plugins). Use the 'edit' option to add, remove or modify custom actions (using APP if specified or the default associated application otherwise). The aim of this function is to allow the user to easily add custom commands and functions to CliFM. In other words, the actions function is a plugins capability.

The general procedure is quite simple: a) bind a custom action name to an executable file written in any language you want, be it a shell or Python script, a C program or whatever you like (using the *actions.cfm* file located in the configuration directory). Example: "myaction=myscript.sh". b) Now, drop the corresponding script (in our example, myscript.sh) into the plugins directory (see the **FILES** section below). 3) Once this is done, you can call the script using the custom action name defined before as if it were any other command: run 'myaction', and myscript.sh will be executed.

All arguments passed to the action command are passed to the script or program as well (which is run via the system shell).

The plugins provided with CliFM (take a look at the plugins directory) could be used as a starting point to create custom plugins.

alias [import FILE] [ls,list] [NAME]
with no argument (or with *ls,list* parameters), it prints the list of available aliases, if any. To get the description of a specific alias just enter *alias* followed by the alias name. To write a new alias simply enter *edit* (or press F10) to open the configuration file and add a line like this: "alias name='command args...'" or "alias name='directory'".

To import aliases from a file, provided it contains aliases in the specified form, use the *import* parameter. Aliases conflicting with some of the internal commands won't be imported.

However, a neat usage for the alias function is not so much to bind short keys to commands, but to files and directories visited regularly. In this way, it is possible to bind as many files or directories,

no matter how deep they are in the file system, to very short strings, even single characters. For example, "alias w='/some/file/deep/in/the/filesystem'. Now, no matter where we are, we can just enter 'w', provided *autocd* and/or *auto-open* function is enabled, to access the file or directory we want. Theoretically at least, this procedure could be repeated until the system memory is exhausted.

To create multiple aliases for files at once, this is the recommended procedure: 1) Select all files you want to alias with the *sel* function: 's file1 file2 file3 ...'. 2) Export the selected files into a temporary file running 'exp sel'; 3) Edit this file to contain only valid alias lines:

```
alias a1='file1'
alias b1='file2'
alias c1='file3'
```

NOTE: Make sure alias names do not conflict with other commands, either internal or external. To bypass the conflicts check, performed automatically by the 'alias import' command, you can just edit the aliases file manually (F10).

4) Finally, import this file with the *alias* function: 'alias import tmp_file'. Now, you can access any of these files by entering just a few characters: a1, b1, and c1.

ao, auto-open [on, off, status]
toggle the auto-open function on/off. If set to on, 'FILE' amounts to 'open FILE'.

b, back [h, hist] [clear] [!ELN]
unlike 'cd ..', which sends you to the parent directory of the current directory, this command (with no argument) sends you back to the previously visited directory.

CliFM keeps a record of all visited directories. You can see this list by typing 'b hist', 'b h' or 'bh', and you can access any element in this list by simply passing the corresponding ELN in this list to the *back* command. Example:

```
:) > ~ $ bh
1 /home/user
2 /etc
3 /proc
:) > ~ $ b !3
:) > /proc $
```

NOTE: the line printed in green indicates the current position of the *back* function in the directory history list.

Finally, you can also clear this history list by typing 'b clear'.

The best way of navigating the directory history list, however, is via the *directory jumper* function. See the *j* command below.

bb, bleach ELN/FILE ... n
Bleach is a built-in file names cleaner (based on detox [*https://github.com/dharple/detox*]), whose main aim is to rename file names using only safe characters. Bleach cleans file names up either by removing unsafe (extended-ASCII/Unicode) characters without an ASCII alternative/similar character, or by translating these unsafe characters into an alternative ASCII character based on familiarity/similarity.

These following simple rules are used to compose clean/safe file names:
- NUL (\0) and slash (/) characters are completely disallowed
- Only characters from the **Portable Filename Characters Set** (a-zA-Z0-9._-) are allowed

- { [( )] } are replaced by a dash (-). Everything else is replaced by an underscore (_)
- Unicode characters are translated, whenever possible, into an ASCII replacement. Otherwise, they are just ignored. For example, an upper case A with diacritic (accent, umlaut, diaresis, and so on) will be replaced by an ASCII A, but the smiley face emoji will be simply ignored. A few special signs will be translated into text, for instance, the pound sign will be replaced by "_pound_" and the Euro symbol by "EUR". Translations are made via a translation table (cleaner_table.h)
- File names never start with a dash (-)
- Files named . and .. are not allowed
- Append .bleach to one character long file names
- Do not let a replacement file name start with a dot (hidden) if the original does not
- Max file name length is NAME_MAX (usually 255)

Modified file names will be listed on screen asking the user for confirmation, allowing besides to edit (by pressing 'e') the list of modified file names via a text editor.

If the replacement file name already exists, a dash and a number (starting from 1) will be appended. Ex: file-3.

bd [NAME]

bd is the **backdir** function: it takes you back to the parent directory matching NAME.

With no arguments, *bd* prints a menu with all parent directories relative to the current directory, allowing the user to select an entry. Otherwise, it checks the absolute current directory against the provided query string (NAME): if only one match is found, it automatically changes to that directory; if multiple matches are found, the list of matches is presented to the user in a selection menu. If NAME is a directory name, *bd* just changes to that directory, be it a parent of the current directory or not.

TAB completion and suggestions are available for this function.

Example:

Provided that the current directory is */home/user/git/repositories/lambda*, entering *bd git* will take you immediatelly to */home/user/git*.

Note that there is no need to type the entire directory name; if the query is unambiguous, only a few characters, and even just one, suffices to match the appropriate directory. In our example, *bd g* is enough to take you to */home/user/git*, just as *bd h* will take you to */home*.

The query string could match any part of a directory name: *bd er*, for instance, will take you to */home/user*, since it is an unambiguous query.

bl ELN/FILE ... n

Create symbolic links (in the current directory) for each specified file. The user will be asked to enter a specific suffix for the symlinks. If none is specified, the basename of the corresponding file is used.

bm, bookmarks [a, add PATH] [d, del] [edit] [SHORTCUT, NAME]

with no argument, open the bookmarks menu. Here you can cd into the desired bookmark by entering either its ELN, its shortcut or its name. In this screen you can also add, remove or edit your bookmarks by simply typing 'e' to edit the bookmarks file (which is simply a list of lines with this format: [shortcut]name:path. Ex: [d]documents:/home/user/documents).

If you want to add or remove a bookmark directly from the command line, use the 'a' and 'd' arguments respectively. Example: 'bm a /media/misc' or 'bm d'. You can also open a bookmark by

typing 'bm shortcut' or 'bm name' (in which latter case TAB completion is available).

A handy use for the bookmarks function, provided the expand-bookmarks option is enabled, is to create bookmarks using short names, which will be later easily accessible via TAB completion.

br, bulk ELN/FILE ... n
rename at once all files passed as arguments to the function. It accepts single and multiple file names, wildcards, ELN ranges, and the 'sel' keyword. Example: 'br myfile 4–10 sel'.

Each file name will be copied into a temporary file, which will be opened with the default text editor (via the *mime* function), letting the user modify it. Once the file has been modified and saved, the modifications are printed on the screen and the user is asked whether to proceed with the actual bulk renaming or not.

This built-in bulk rename function won't deal with deletions, replacements, file name conflicts and the like. For a smarter alternative use **qmv(1).**

c, l [e, edit], m, md, r
short for the following commands respectively: 'cp -iRp', 'ln -sn', 'mv -i', 'mkdir -p', and 'rm -I' (for files) or 'rm -dIr' (for directories). To use these commands without any of these arguments, or with any other argument you need, just use the non-abbreviated command, for instance, 'cp' instead of 'c'.

The *ln* command allows the use of the 'e, edit' option to modify the destination of a symbolic link. Example: 'l e 12', or 'le 12', to relink the symbolic link corresponding to the ELN 12.

The *m* command is just like 'mv -i', but with the following differences: When using the *sel* keyword and no destiny is provided, *m* will move selected files into the current directory. Otherwise, whenever *sel* is not used, but just a source file name (and no destiny is provided), the *m* command behaves much like the **imv(1)** shell command (from the 'renameutils' package), providing an interactive renaming function: it prompts the user to enter a new name using the source file name as base, so that it does not need to be typed twice. For this alternative prompt, only TAB completion for file names is available.

NOTE: On POSIX compliant systems, like NetBSD, the *r* command amounts to *rm -r* for directories and *rm -f* for files, since the *-I* option is not available and *-i* is too instrusive.

CliFM supports *advcp*, *wcp*, and *rsync* to copy files (they include a progress bar). To use them instead of *cp* set the corresponding option (*cpCmd*) in the configuration file. If *advcp* is selected, the command used is 'advcp -giRp'. If *rsync*, the command is 'rsync -avP'. *wcp* takes no argument.

advmv is also supported to move files (to add a progress bar to *mv*). Use the 'mvCmd' option in the configuration file to choose this alternative implementation of *mv*. In this case, the command used is 'advmv -gi'.

cc, colors
print the color codes list currently used for files listing.

cd [ELN/DIR]
Change the current working directory to ELN/DIR.

Directories check order:
1. If no argument, change to the home directory (**HOME**, or, if **HOME** is not set, the sixth field of the entry corresponding to the current user in */etc/passwd*)
2. If argument is an absolute path (begins with a slash character), or the first component is dot (.)

or dot-dot (..), convert to canonical form (via **realpath**(3)) and, if a valid directory, change into that directory.
3. Check **CDPATH** environment variable and append /DIR to each of the paths specified here. If the result of the concatenation is a valid directory, change into it.
4. Check directories in the current working directory. If a matching directory is found, change to it.

You can use either ELN's or a string to indicate the directory you want. Ex: 'cd 12' or 'cd ~/media'. If *autocd* is enabled (default), 'cd 12' and 'cd ~/media' could be written as '12' and '~/media' respectively as well.

Unlike the shell **cd**(1) command, CliFM's built−in *cd* function not only changes the current directory, but also lists its content (provided the option *CdListsAutomatically* is enabled, which is the default) according to a comprehensive list of color codes. By default, the output of *cd* is much like this shell command: cd DIR && ls −−color=auto −−group−directories−first.

Automatic files listing can be disabled either setting *CdListsAutomatically* to "false" in the configuration file or running CliFM with the *-o* or *--no-list-on-the-fly* option.

cl, columns [on, off]
toggle columns on/off.

cmd, commands
show this list of commands. A more convenient way of getting information about CliFM commands is via the interactive help plugin (depends on *fzf*), by default bound to the "ihelp" action name.

cs, colorschemes [edit [APP]] [NAME]
with no arguments, list available color schemes. Use the 'edit' option to open/edit the configuration file of the current color scheme (open with APP if specified or via the default associated application). Otherwise, just switch to the color scheme NAME. TAB completion is available.

d, dup FILE(s)
Create a duplicate of FILE, where FILE could be either a directory or a regular file. The duplicated file name is generated by appending ".copy" to the basename of FILE. For example: 'd /my/file' will copy '/my/file' into the current directory as 'file.copy'. If 'file.copy' already exists, an extra suffix will be added as follows: 'file.copy-N', where N is an integer value greater than zero.

If **rsync**(1) is found, it will be used as follows: 'rsync -aczvAXHS --progress'. Else, **cp**(1) will be used: 'cp -a'.

dc [on, off, status]
toggle the files counter function on/off.

ds, desel [*, a, all] [FILE ...]
deselect one or more selected files.

If no parameter is passed, the user is prompted to either mark selected files to be deselected or to edit the selections file (entering 'e') via a text editor to manually deselect files.

Use *, *a* or *all* to deselect all selected entries at once. Ex: 'ds *'.

You can also pass the file name(s) (or ELN's) to be deselected as a parameter. For example: 'ds myfile 24'.

TAB completion is available for this command.

edit [reset] [APPLICATION]
edit the main configuration file (F10 key is also available). If an application is specified, it will be used to open the configuration file. Use the 'reset' option to generate a fresh configuration file and create a backup copy of the old one (named clifmrc.YYYYMMDD@HH:MM:SS).

exp [FILE ...]
with no argument, export the list of files in the current working directory to a temporary file. Otherwise, export only those specified as further arguments: they could be directories, file names, ELN's or some search expression like "*.c".

ext [on, off, status]
toggle external commands on/off.

f, forth [h, hist] [clear] [!ELN]
it works just like the *back* function, but it goes forward in the history record. Of course, you can use 'f hist', 'f h', 'fh', and 'f !ELN'.

fc, filescounter [on, off, status]
By default, CliFM prints the amount of files contained by listed directories next to directories name. However, since this is an expensive feature, It might be desirable, for example, when listing files in a remote machine, to disable this feature. Use the 'off' option to disable it. To permanently disable it, use the FilesCounter option in the configuration file.

ff, folders-first [on, off, status]
toggle list folders first on/off.

fs Print an extract from 'What is Free Software?', written by Richard Stallman.

ft, filter [unset] [[!]REGEX]
with no argument, print the current filter. To remove the current filter use the 'unset' option. To set a new filter enter 'ft [!]REGEX'. Use the exclamation mark to reverse the behavior or a filter. For example: 'ft !^.' will prevent hidden files from being listed, just as 'ft ^D' will list only files starting with 'D'.

The filter will be lost at program exit. To permanently set a files filter use the *Filter* option (in the configuration file). You can also use the **CLIFM_FILTER** environment variable (see below), though the value of this variable will be lost at system shutdown or reboot.

fz [on, off]
Toggle full directory size on/off (only for long view mode).

hf, hidden [on, off, status]
toggle hidden files on/off.

history [edit [APP]] [clear] [−n] [on, off, status]
with no arguments, it shows the history list. If 'clear' is passed as argument, it will delete all entries in the history file. Use 'edit' to open the history file and modify it if needed (the file will be opened with APP, if specified, or with the default associated application otherwise). '−n' tells the *history* command to list only the last 'n' commands in the history list. Finally, you can disable history (subsequent entries won't be written to the history file) via 'history off'.

You can use the exclamation mark (!) to perform some history commands:
!!: Execute the last command.
!n: Execute the command number 'n' in the history list.
!−n: Execute the last−n command in the history list.
![STRING]: Execute command starting with STRING. TAB completion is available in this case.

TAB completion is available: just type '!' and then TAB to see the complete list of history commands, or '!str' and then TAB to see the list of entries matching 'STR'.

icons [on, off]
toggle icons on/off

j, jc, jl, jp [STR ...], **jo** [NUM], **je**
j is the fastest way of using **Kangaroo**, a **directory jumper** function to quickly navigate through the jump database (i.e. a database of visited directories).

With no argument, *j* just lists the entries in the jump database, printing the order number of the corresponding entry, the number of visits, the days since the first visit, the rank value, and the directory name itself (an asterisk next to the rank value means that the corresponding directory is bookmarked, the current directory in some workspace or pinned). Otherwise, it searches for STR in the database and cd into the best ranked matching entry. Example: 'j D' will probably take you to */home/user/Downloads*, provided this directory has been already visited and is the best ranked match in the database. For a more detailed description of the matching algorithm see the **KANGAROO FREQUENCY ALGORITHM** section below.

Multiple query strings could be passed to the function. For example, 'j et mo' will first check for 'et' in the jump database and then will further filter the search using the second parameter: 'mo'. It will most probably take you (again, provided the directory has been already visited and is the best ranked match) to /etc/modprobe.d directory. Bear in mind that if STR is an actual directory, *jump* will just cd into it without performing any query.

The backslash (\) and the slash (/) could be used to instruct **Kangaroo** to search for the string query only in the first or last path segment of each entry in the database respectivelly. Let's suppose we have two entries matching **src** in the database: */media/src/images* and */home/user/Downloads/clifm/src*. If the first entry is better ranked than the second, *j src* will match this first entry. However, if what we really want is the second entry, appending a slash to the query string instructs **Kangaroo** to only match entries having src in the last path segment, here */home/user/Downloads/clifm/src*.

Since it is not always obvious or easy to know where exactly a query string will take you, CliFM (if the suggestions system is enabled) will print, at the right of the cursor, the path matched by **Kangaroo**. If that is the actually intended path, just press the Right arrow key to accept the suggestion. Otherwise, it will be ignored. You can also use TAB completion to print the list of matches for the current query string. For example: *j - c<TAB>* to list all entries in the directory history list containing a dash (-) and a 'c'.

j accepts five modifiers: 'e', 'p' 'c', 'o', and 'l', the first standing for "edit", the second for "parent", the third for "child", the fourth for "order", and the last one for 'list'. Thus, 'je' will open the jump database to be edited if needed; 'jc' will search for files querying only child directories relative to the current working directory, while 'jp' will do the same but for parent directories. 'jo' allows to specify an order number (the left most value in the jump list) instead of a string or a file name, in which case no matching process is performed. Finally, 'jl' just prints the matches for the given query string(s), but without changing the current directory. Examples:

´jp foo´ will take you to the most visited **parent** directory containing the string "foo".

´jc bar test´ will take you to the most visited **child** directory containing the strings "bar" and "test".

´jo 13´ will cd into the path corresponding to the order number 13. TAB completion is available to expand order numbers into the corresponding paths.

´jl foo´ will print all entries in the database matching the word "foo".

To reset or modify the jump database as you wish, simply open the jump file using the 'je'

command, edit whatever needs to be edited, save changes, and close the editor.

An alternative way of navigating the jump database is using the jumper plugin (located in the plugins directory and bound by default to the "++" action name), which uses *fzf* to enable fuzzy searches. Just enter ++ to perform a fuzzy search over the jump database.

kb, keybinds [edit [APP]] [reset] [readline]
with no argument, prints the current keyboard codes and their associated functions. To edit the keybindings file, use the 'edit' option (the file will be opened with APP, if specified, or with the default associated application otherwise). If you somehow messed up your keybindings, use the 'reset' option to create a fresh keybindings file with the default values. To list readline keybindings, use the 'readline' option. Bear in mind that these keybindings are not provided by CliFM, but by readline itself, and as such depend on the system settings (they can be customized however via the '~/.inputrc' file).

lm [on, off]
Toggle the light mode on/off. This option, aimed to make files listing faster than the default mode, is especially useful for really old hardware or when working on remote machines. For more details see the **NOTE ON SPEED** section below.

log [clear] [on, off, status]
with no arguments, it prints the contents of the log file. If 'clear' is passed as argument, all the logs will be deleted. 'on', 'off', and 'status' enable, disable, and check the status of the *log* function for the current session.

media

NOTE: This command is Linux-specific

List available storage devices and mount/unmount the selected one using either *udevil* or *udisks2* (at least one of these must be installed. *udevil* will be preferred over *udisks2*). If the device is unmounted, it will be automatically mounted, and if mounted, it will be automatically unmounted.

Though mountpoints are determined by the mounting application itself (*udevil* or *udisks2*), CliFM will automatically cd into the corresponding mountpoint whenever the mount operation was succesfull.

When unmounting, and if the current directory is inside the mountpoint, CliFM will attempt to cd into the previous visited directory, and, if none, into the home directory, before unmounting the device.

To get information about a device, enter *iELN*, for example, 'i12', provided '12' is the ELN of the device you want.

mf [NUM, unset]
List only up to NUM files (valid range: >= 0). Use 'unset' to list all files (default). An indicator (listed_files/total_files) will be printed below the list of files whenever some file is excluded from the current list (e.g. 20/310). Note however that though some files are excluded, all of them are loaded anyway, so that you can still perform any valid operation on them. For example, even if only 10 files are listed, you can still search for ALL symbolic links in the corresponding directory using the appropriate command: '/* -l'.

mm, mime [info ELN/FILENAME] [edit] [import]
This is *Lira*, CliFM's resource opener. The 'info' option prints the MIME information about ELN/FILENAME: its MIME type, and, if any, the application associated to this file name or to the file's MIME type.

The 'edit' option allows you to edit and customize the MIME list file. So, if a file has no default

associated application, first get its MIME info or its file extension (running 'mm info FILE'), and then add a value for it to the MIME list file using the 'edit' option ('mm edit' or F6). Check the **RESOURCE OPENER** section below for information about the mimelist file syntax.

Finally, via the 'import' option CliFM will try to import MIME definitions from the system looking for *mimeapps.list* files in those paths specified by the Freedesktop specification (see *https://specifications.freedesktop.org/mime-apps-spec/mime-apps-spec-latest.html*). If at least one MIME definition is successfuly imported, a backup of the old *mimelist.cfm* file will be stored as *mimelist.cfm.YYYYMMDDHHMMSS*. Otherwise, no change will be made.

mp, mountpoints
list available mountpoints and change the current working directory to the selected mountpoint.

msg, messages [clear]
with no arguments, prints the list of messages in the current session. The 'clear' option tells CliFM to empty the messages list.

n, new [FILE DIR/ ...n]
create new empty files and/or directories. If a file name ends with a slash (/), it will be taken as a directory name and created via the shell command 'mkdir -p'. Else, it will be created via the 'touch' shell command. Ex: 'n myfile mydir/', to create a file named "myfile" and a directory named "mydir". If no file name is specified, the user will be asked for one. If one or more of the specified file names already exist, ".new" will be appended to the file name.

net [NAME] [edit] [m, mount NAME] [u, unmount NAME]
1. The configuration file

The *net* command manages connections to remote systems via a simple samba-like configuration file (*$HOME/.config/clifm/profiles/PROFILE/nets.cfm*). Here you can specify multiple remotes and options for each of these remotes. Syntax example for this file:

```
[remote_name]
Comment=A nice descriptive comment
Mountpoint=/path/to/mountpoint
MountCmd=sudo mount.cifs //192.168.0.12/share %m -o OPTIONS
UnmountCmd=sudo umount %m
AutoUnmount=true (Auto-unmount this remote at exit)
AutoMount=false (Auto-mount this remote at startup)
```

Note: *%m* could be used as a placeholder for *Mountpoint*. *%m* will be replaced by the value of *Mountpoint*.

1.a. Mounting remote file systems

A Samba share:

```
[samba_share]
Comment=My samba share
Mountpoint="~/.config/clifm/mounts/smb_share"
MountCmd=sudo mount.cifs //192.168.0.26/samba_share %m -o
mapchars,credentials=/etc/samba/credentials/samba_share
UnmountCmd=sudo umount %m
AutoUnmount=false
AutoMount=false
```

A SSH file system (sshfs):

```
[ssh_share]
Comment=My ssh share
```

```
Mountpoint="/media/ssh"
MountCmd=sshfs user@192.168.0.26: %m -C -p 22
UnmountCmd=fusermount3 -u %m
AutoUnmount=true
AutoMount=false
```

1.b. Mounting local file systems

Though originally intended to manage remote file systems, *net* can also manage **local file systems**. Just provide the appropriate mount and unmount commands. Since the device name assigned by the kernel might change accross reboots (specially when it comes to removable drives), it is recommended to mount using the device's UUID (Universal Unique Identifier) instead of the drive name. For example:

```
MountCmd=sudo mount -U c98d91g4-6781... %m
```

Here's an example of how to set up *net* to mount USB devices, one with a FAT file system, and another with an ISO9660 file system:

```
[Sandisk USB]
Comment=Sandisk USB drive
Mountpoint="/media/usb"
MountCmd=sudo mount -o gid=1000,fmask=113,dmask=002 -U 5847-xxxx %m
UnmountCmd=sudo umount %m
AutoUnmount=false
AutoMount=false

[Kingston USB]
Comment=Kingston USB drive
Mountpoint="/media/usb2"
MountCmd=sudo mount -t iso9660 -U 2020-10-01-15-xx-yy-zz %m
UnmountCmd=sudo umount %m
AutoUnmount=false
AutoMount=false
```

NOTE: The *gid*, *fmask*, and *dmask* options are used to allow the user to access the mountpoint without elevated privileges.

If the device data is unknown, as it often happens when it comes to removable devices, you should use the *media* command instead.

2. Command syntax

Without arguments, *net* lists the configuration for each remote available in the configuration file.

Use the *edit* option to edit the remotes configuration file. If no further argument is specified, the file will be opened with the current resource opener. However, you can pass an application as second parameter to open to configuration file. Example: 'net edit nano'.

If not already mounted, the *m*, *mount* option mounts the specified remote using the mount command and the mounpoint specified in the confifuration file and automatically cd into the corresponding mountpoint. Example: *net m smb_work*. *m*, *mount* could be omitted, so that *net smb_work* amounts to *net m smb_work*. TAB completion is available for this function.

The *u*, *unmount* option unmounts the specified remote using the unmount command specified in the configuration file. For example: *net u smb_work*. TAB completion is also available for this function.

NOTE: If you only need to copy some files to a remote location (including mobile phones) without the need to mount the resource, you can make use of the cprm.sh plugin, bound by default to the *cr* action. Just set up your remotes (*cr edit*) and then simply send the file you want (*cr FILE*). That's all.

o, open ELN/FILE [APPLICATION]
open FILE, which can be either a directory, in which case it works just like the *cd* command (see above), a regular file, or a symbolic link to either of the two. For example: 'o 12', 'o filename', 'o /path/to/filename'.

By default, the *open* function will open files with the default application associated to them via *Lira*, the built−in resource opener (see the *mime* command above). However, if you want to open a file with a different application, just add the application name as second argument, e.g. 'o 12 leafpad' or 'o12 leafpad'.

If you want to run the program in the background, simply add the ampersand character, as usual: 'o 12 &', 'o 12&', 'o12&' or (if auto-open is enabled) just '12&'.

If the file to be opened is an archive/compressed file, the archive function (see the *ad* command above) will be executed instead.

ow ELN/FILE [APPLICATION]
Print a list of available applications associated to _ELN/FILENAME (either via its MIME type or its file extension), allowing the user to choose one of these applications, and then open the file with the selected application. In simple words, this is what in most GUI file managers is called **Open with...** This command supports TAB completion: just type "*ow filename* ", then press TAB, and those applications able to open FILENAME will be listed.

opener [default] [APPLICATION]
with no argument, prints the currently used resource opener (by default, *Lira*, CliFM's built−in opener). Otherwise, set APPLICATION as opener or, if 'default' is passed instead, use *Lira*.

p, pr, prop ELN/FILE ... n
print file properties for ELN/FILE. The output of this function is much like the combined output of 'ls -l' and 'stat'. By default, directories size is not shown. Use *pp* instead of just *p* to print directories size as well (it could take longer depending on the directory's content).

path, cwd
print the current working directory.

pf, prof, profile [ls, list] [set, add, del PROFILE]
with no arguments, prints the name of the currently used profile. Use the 'ls' or 'list' option to list available profiles. To switch, add or delete a profile, use the 'set', 'add', and 'del' options respectively followed by the corresponding profile name. Bear in mind that, when switching profiles, command line arguments will be ignored.

pg, pager [on, off, status]
toggle **Mas**, the built-in pager, on/off. Useful to list directories with hundreds or thousands of files, the pager will start working, if set to on, whenever the screen is not enough to list all files.

Once in the pager, press the Down arrow key, Space or Enter to move downwards one line, or PageDown to move downwards an entire page. To go upwards, use the shortcuts provided by your terminal emulator, for example, Alt−PageUp or Alt−Up. Press 'c', 'p', or 'q' keys to stop the pager, and 'h' or '?' for help.

pin [FILE/DIR]
pin a file or a directory to be accessed later via the comma (,) keyword. For example, run 'pin mydir' and then access "mydir" as follows: 'cd ,' where the comma is automatically expanded to the pinned file, in this case "mydir". The comma keyword could be used with any command, either internal or external, e.g, 'ls ,'.

With no arguments, the *pin* command prints the current pinned file, if any. If an argument is given, it will be taken as a file name to be pinned. Running this command again, frees the previous pinned file and sets a new one. In other words, only one pin is supported at a time.

An easy alternative to create as many pins or shortcuts as you want, and how you want, is to use the *alias* function. Bookmarks could also be used to achieve a very similar result.

At program exit, the pinned file is written to a file in the configuration directory (as .pin) to be loaded in the next session.

q, quit, exit, Q
Gracefully quit CliFM. Use 'Q' to gracefully quit and enable the CD on quit functionality (write last visited directory to *$XDG_CONFIG/clifm/.last* to be later read by a shell function. See the **SHELL FUNCTIONS** section below).

rf, refresh
refresh the screen, that is, reprint files in the current directory and update the prompt. If the current directory is not accessible for any reason, *rf* will go up until it finds an accessible one and then will change to that directory.

rl, reload
Reload all settings, except those passed as command line arguments, from the configuration file.

rr [DIR] [EDITOR]
Remove files and/or directories in bulk using a text editor.

rr sends all files in *DIR* (or in the current directory if *DIR* is omitted) to a temporary file and opens it using *EDITOR* (or the default associated application for *text/plain* MIME type, if *EDITOR* is omitted).

Once in the editor, remove the lines corresponding to the files you want to delete. Save changes and close the editor. Removed files will be listed and the user asked for confirmation.

s, sel ELN/FILE ... n [[!]PATTERN] [-filetype] [:PATH]
send one or multiple elements (either files or directories) to the Selection Box. *sel* accepts individual elements, range of elements, say 1-6, file names and paths, just as wildcards (globbing) and regular expressions. Example: 's 1 4-10 ^r file* file name /path/to/filename'. To deselect selected files, use the *ds* command (see above), or the M-d keyboard shortcut.

If not in light mode, once a file is selected, and if the file is in the current working directory, the corresponding file name will be marked with an asterisk (colored according to the value of *li* in the color scheme file (by default bold green)), at the left of the file name (and at the right of its ELN).

Just as in the *search* function, it is also possible to further filter the list of matches indicating the desired file type. For instance, 's ^ -d' will select all directories in the current working directory. For available file type filters see the *search* function above.

By default, the selection function operates on the current working directory. To select files in any other directory use the ":PATH" expression. For example, to select all regular files with a .conf extension in the /etc directory, the command would be: 's .*\.conf$ -r :/etc', or using wildcards: 's *.conf -r :/etc'.

Just as in the case of the *search* function, inverse matching is supported for patterns, either wildcards or regular expressions. To invert or reverse the meaning and action of a pattern, just prepend an exclamation mark (!). E.g., to select all non-hidden regular files in the Documents directory, issue this command: 's !^. -r :Documents', or, to select all directories in /etc, except those ending with ".d": 's !*.d -d :/etc'.

Glob and regular expressions could be used together. For example: 's ^[r|R].*d$ /etc/*.conf' will select all files starting with either 'r' or 'R' and ending with 'd' in the current working directory, plus all .conf files in the /etc directory. However, this use is discouraged if both patterns refer to the same directory, since the second one will probably override the result of the first one.

It is important to note that glob expressions are evaluated before regular expressions, in such a way that any pattern that could be understood by both kinds of pattern matching mechanisms will be evaluated first according to the former, that is, as a glob expression. For example, '.*', as regular expression, should match all files. However, since glob expressions are evaluated first, it will only match hidden files. To select all files using a glob expression, try '.* *', or, with a regular expression: '^' or '(.*?)'. The keyboard shortcut M-a is also available to perform the same operation.

The Selection Box is accessible to different instances of the program, provided they use the same profile (see the *profile* command below). By default, indeed, each profile keeps a private Selection Box, being thus not accessible to other profiles. You can nonetheless modify this behavior via the ShareSelbox option in the configuration file. If the ShareSelbox option is enabled (see the configuration file), selected files are stored in */tmp/clifm/username/.selbox.cfm*. Otherwise, */tmp/clifm/username/.selbox_profilename.cfm* is used (this is the default).

To operate on selected files, use the *sel* keyword. Consult the **BUILT-IN EXPANSIONS** section below.

sb, selbox
show the elements currently contained in the Selection Box.

splash show the splash screen.

st, sort [METHOD] [rev]
with no argument, print the current sorting method. Else, set sorting method to METHOD, where METHOD could be one of: 0 = none, 1 = name, 2 = size, 3 = atime, 4 = btime (ctime, if btime is not available), 5 = ctime, 6 = mtime, 7 = version (name, if ctime is not available), 8 = extension, and 9 = inode, 10 = owner, and 11 = group. Both numbers and names are allowed. Bear in mind that methods 10 and 11 sort by owner and group ID number, not by owner and group names.

By default, files are sorted from less to more (ex: from 'a' to 'z' if using the "name" method). Use the 'rev' option to invert this order. Ex: 'st rev' or 'st 3 rev'. Switch back to the previous ordering running 'st rev' again.

stats print statistics about files in the current directory (not available in light mode).

t, tr, trash [ELN/FILE ... n] [ls, list] [clear] [del [FILE(s)]]
with no argument (or by passing the *ls* option), it prints the list of currently trashed files. The *clear* parameter removes all files from the trash can, while the *del* parameter lists trashed files allowing the user to remove one or more of them. If using *del*, TAB completion to list/select currently trashed files is available.

The trash directory is *$XDG_DATA_HOME/Trash*, usually *~/.local/share/Trash*. Since this trash system follows the Freedesktop specification, it is able to handle files trashed by different Trash implementations.

To undelete/untrash trashed files see the *undel* command below.

tag [ls, list] [new] [rm, remove] [mv, rename] [untag] [merge] [FILE(s)] [[:]TAG]
tag is the main *Etiqueta* command, CliFM's built-in files tagging system. See the **FILE TAGS** section for a complete description of this command.

te FILE(s)
toggle executable bit (on user, group, and others) on FILE(s). It is equivalent to the **-x** and **+x** options for the **chmod**(1) command.

tips print the list of CliFM tips

u, undel, untrash [*, a, all] [FILE(s)]
If file names are passed as parameters, undelete these files, that is, restore them to their original location. Otherwise, this function prints a list of currently trashed files allowing you to choose one or more of these files to be undeleted. Use the *, *a* or *all* parameters to undelete all trashed files at once. TAB completion to list/select currently trashed files is available.

uc, unicode [on, off, status]
toggle unicode on/off.

unpin this command takes no argument. It just frees the current pin and, if it exists, deletes the *.pin* file generated by the *pin* command.

v, vv, paste sel [DESTINY]
the '*paste sel*' or '*v sel*' command copies the currently selected files, if any, into the current working directory. To copy these files into another directory, tell 'paste' where to copy these files. Ex: '*paste sel /path/to/directory*'. The *copy* command (*c*) could be used in the same way: '*c sel*' indeed copies selected files into the current directory. Use the '*vv*' command instead of just '*v*' to copy selected files into DESTINY and rename them at once: '*vv sel DIR*'.

ver, version
show CliFM version details.

ws [NUM, +, -]
CliFM offers up to eight workspaces, each with its own independent path.

With no argument, the *ws* command prints the list of workspaces and its corresponding paths, highlighting the current workspace. Use NUM to switch to workspace NUM, the plus sign (+) to switch to the next workspace, and the minus sign (-) to switch to the previous workspace. Four keyboard shortcuts are available to easily switch to any of the first four workspaces: **Alt-[1-4]**.

Every time an empty workspace is created, it starts in the path of the workspace from which it was invoked (in other words, in the current working directory).

x, X [DIR]
open DIR, or the current working directory if DIR is not specified, in a new instance of CliFM (as root if *X*, as the current unprivileged user if *x*) using the value of *TerminalCmd* (from the configuration file) as terminal emulator. If this value is not set, *xterm* will be used as fallback terminal emulator. This function is only available for graphical environments.

7. FILE FILTERS

CliFM offers three kinds of file filters:

a) A dot filter to permanently exclude hidden files from the files list. See the **-A** and **-a** options above.

b) Files filter via regular expressions (using the *Filter* option in the configuration file or the CLIFM_FILTER environment variable) to permanently exclude certain groups of file names, for example, backup files or files ending with a tilde (~).

c) Local filter via the *Quick search* function (supporting both regular expressions, wildcards, and invert matching) to temporarily filter the current list of files.

8. KEYBOARD SHORTCUTS

Right, **C-f:** Accept the entire current suggestion

Alt-Right, **M-f:** Accept only the first word of the current suggestion (up to first slash or space)

M–c: Clear the current command line buffer

M–q: Delete last word (up to last slash or space)

M–i, M–.: Toggle hidden files on/off

M–l: Toggle long view mode on/off

M–g: Toggle list–folders–first on/off

M–,: Toggle list only directories on/off

C-M-l: Toggle max file name length on/off

C-M-i: Toggle disk usage analyzer on/off

C–r: Refresh the screen (reprint files in current directory and update prompt)

M–t: Clear program messages

M–m: List mountpoints

M–b: Launch the Bookmarks Manager

M–h: Show directory history

M–n: Create new file or directory

M–s: Open the Selection Box

M–a: Select all files in the current working directory

M–d: Deselect all selected files

M–p: Change to pinned directory

M–1: Switch to workspace 1

M–2: Switch to workspace 2

M–3: Switch to workspace 3

M–4: Switch to workspace 4

M–r: Change to root directory

M–e, Home: Change to home directory

M–u, S-Up: Change to parent directory

M–j, S-Left: Change to previous visited directory

M–k, S-Right: Change to next visited directory

C–M–j: Change to first visited directory

C–M–k: Change to last visited directory

C–M–o: Switch to previous profile

C–M–p: Switch to next profile

C–M–a: Archive selected files

C−M−e: Export selected files

C−M−r: Rename selected files

C−M−d: Remove selected files

C−M−t: Trash selected files

C−M−u: Restore trashed files

C−M−b: Bookmark last selected file/directory

C−M−g: Open/change-to last selected file/directory

C−M−n: Move selected files into the current directory

C−M−v: Copy selected files into the current directory

M−y: Toggle light mode on/off

M−z: Switch to previous sorting method

M−x: Switch to next sorting method

C−x: Launch new instance of the program

F1: Go to the manpage

F2: List commands

F3: List keybindings

F6: Open the MIME list file

F7: Open the jump database file

F8: Open the current color scheme file

F9: Open the keybindings file

F10: Open the configuration file

F11: Open the bookmarks file

F12: Quit

NOTE: C stands for Control, S for Shift, and M for the Meta key (the Alt key in most keyboards).

NOTE 2: Some of these keybindings might not work on your console/terminal emulator, depending on your system. Some useful tips on this regard:

Haiku terminal: Most of these keybindings won't work on the Haiku terminal, since Alt plays here the role Ctrl usually plays in most other systems (see the Haiku documentation). To fix this just set your custom keybindings.

Kernel built-in console: Key sequences involving the Shift key (S-up, S-left, and S-right in our case) will just not work. Use the alternative key sequences instead: M-u, M-j, and M-k respectively

NetBSD (wsvt25) and OpenBSD (vt220) kernel consoles: Key sequences involving the Alt key won't work out of the box. Here's how to make it work:

On OpenBSD:
1) Copy /etc/examples/wsconsctl.conf to /etc (if it does not already exist)
2) Add the metaesc flag to your current keyboard enconding. For example
keyboard.encoding=us.metaesc
You might need to reboot the machine for changes to take effect.

On NetBSD:
Add the metaesc flag to your current encoding in /etc/wscons.conf. Example: encoding
us.metaesc
You might need to reboot the machine for changes to take effect.

Konsole: If Shift+left and Shift+right are not already bound to any function, you need to bind them manually. Go to Settings -> Edit current profile -> Keyboard -> Default (Xfree4), and add these values:

```
Left+Shift      \E[1;2D
Right+Shift     \E[1;2C
```

If they are already bound, by contrast, you only need to unbound them. Go to "Settings -> Configure keyboard shortcuts", click on the corresponding keybinding, and set it to "Custom (none)").

Terminology/Yakuake: Shift+left and Shift+right are already bound to other functions, so that you only need to unbind them or rebind the corresponding functions to different key sequences.

Of course, the above two procedures should be similar in case of keybinding issues in other terminal emulators.

In case some of these keybindings are already used by your Window Manager, you only need to unbind the key or rebind the corresponding function to another key. Since each Window Manager uses its own mechanisms to set/unset keybindings, you should consult the appropriate manual.

Customizing keybindings

The above are the default keyboard shortcuts. However, they can be freely modified using the 'kb edit' command (or pressing F9) or just editing the keybindings file (see the **FILES** section below) to your liking.

Since CliFM does not depend on the curses library, keybindings are set up via ANSI escape codes, for example, "\[17~" for the F6 key. The two main difficulties with ANSI escape codes are: 1) They are not intuitive at all, and 2) They vary depending on the terminal emulator used. This is why we provide a plugin (kbgen) to more easily configure your keybindings.

The plugin can be found in the plugins directory as a C source file. The first step, therefore, is to compile this source file to produce a binary file. Compile as follows:

```
gcc -o kbgen kbgen.c
```

Note: Depending on your system, you might need to link against the curses library adding either **-lcurses** or **-lncurses** to the above line.

Now, just run the plugin entering './kbgen'. Use either octal, hexadecimal codes or symbols. Example: For F12 'kbgen' will print the following lines:

```
Hex  | Oct  | Symbol
---- | ---- | ------
\x1b | \033 | ESC (\e)
\x5b | \133 | [
\x32 | \062 | 2
\x34 | \064 | 4
\x7e | \176 | ~
```

In this case, supposing you want to use F12 to open the configuration file, the keybinding would be any of the following:

```
open-config:\x1b\x5b\x32\x34\x7e (Hex)
open-config:\033\133\062\064\176 (Oct)
open-config:\e[24~ (Symbol)
```

GNU emacs escape sequences are also allowed (ex: "\M-a", Alt-a in most keyboards, or "\C-r" for Ctrl-r). Some codes, especially those involving keys like Ctrl or the arrow keys, vary depending on the terminal emulator and the system settings. These keybindings should be set up thus on a per terminal basis. You can also consult the terminfo database via the *infocmp* command. See **terminfo**(5) and **infocmp**(1).

Readline keybindings

System readline keybindings for command line editing, such as *Ctrl-a*, to move the cursor to the beginning of the line, or *Ctrl-e*, to move it to the end, should work out of the box. Of course, you can modify readline keybindings using the *$HOME/.inputrc* file, either globally or for some specific terminal or application. In this latter case, it is possible to set keybindings specifically for CliFM using the *application* construct, that is, telling readline that the following keybindings apply only to CliFM. For example, to bind the function "kill-whole-line" to *Ctrl-b*, add the following lines to your *.inputrc* file:

```
$if clifm
"\C−b": kill−whole−line
$endif
```

Keybindings for plugins

CliFM provides four customizable keybindings for custom plugins. The procedure for setting a keybinding for a plugin is the following:

1) Copy your plugin to the plugins directory (or use any of the plugins already in there)
2) Link pluginx (where 'x' is the plugin number [1-4]) to your plugin using the 'actions edit' command. Ex: "plugin1=myplugin.sh"
3) Set a keybinding for pluginx using the 'kb edit' command. Ex: "plugin1:\M−7"

9. THEMING

All customization settings (theming) are made from a single configuration file (the color scheme file). Multiple color scheme files are allowed, each specifying a full theme for CliFM. This file includes:

FiletypeColors = Colors for different file types, such as directory, regular files, and so on. See the **COLORS** section below.

InterfaceColors = Colors for CliFM's interface, such as ELN's, file properties bits, suggestions, syntax highlighting, etc. See the **COLORS** section below.

ExtColors = Colors for files based of file name's extension. See the **COLORS** section below.

DirIconColor = Color for the directory icon (when icons are enabled). See the **COLORS** section below.

Prompt = Main prompt string. See the **THE PROMPT** section below.

WarningPromptStr = Warning prompt string. See the **THE PROMPT** section below.

DividingLine = The line dividing the current list of files and the prompt. See the **THE DIVIDING LINE** below.

FzfTabOptions = Options to be passed to fzf when using the fzf mode for TAB completion, including colors. See the **BUILT-IN EXPANSIONS** section below.

The color scheme (or just theme) can be set either via the command line (*--color-scheme=NAME*), via the *ColorScheme* option in the main configuration file, or using the *cs* command, for instance, 'cs mytheme'.

By default, CliFM includes two themes: *default* (dark) and *light*.

1. COLORS

All color codes are specified in the corresponding color scheme file (by default *~/.config/clifm/colors/default.cfm*). You can edit this file pressing **F8** or entering *cs edit*.

Color codes

Colors are specified using the same format used by **dircolors(1)** and the **LS_COLORS** environment variable, namely, a colon separated list of codes with this general format: *filetype=color*. This is the list of **file type codes** (you'll find them in the *FiletypeColors* section of the current color scheme file):

```
di = directory
ed = empty directory
```

nd = directory with no read permission
ne = empty directory with no read permission
fi = regular file
ef = empty regular file
nf = file with no access permission
ln = symlink
mh = multi-hardlink file
or = orphaned or broken symlink
bd = block device
cd = character device
pi = FIFO, pipe
so = socket
su = SUID file
sg = SGID file
tw = sticky and other writable directory
st = sticky and not other writable directory
ow = other writable directory
ex = executable file
ee = empty executable file
ca = file with capabilities
no = unknown file type
uf = unaccessible files (**fstatat**(3) error)

The following codes are used for different interface elements (in the *InterfaceColors* section of the current color scheme file):

Suggestions
sb = shell built-ins
sc = aliases and shell command names
sf = ELN's plus bookmarks, file, tag, and directory names
sh = commands history entries
sx = suggestions for CliFM's internal commands and parameters
sp = suggestions pointer (ex: 56 > filename, where '>' is the suggestion pointer)

Syntax highlighting
hb = brackets '()[]{}'
hc = comments (lines starting with '#')
hd = slashes
he = expansion chars '~*'
hn = numbers
hp = option parameters (starting with '-')
hq = quoted strings (both single and double quotes)
hr = process redirection (>)
hs = process separators (; & |)
hv = variable names (starting with '$')

Prompt elements
li = selected files
ti = trash indicator
em = error message indicator
wm = warning message indicator
nm = notice message indicator
si = stealth mode indicator
tx = command line text (regular prompt)

wp = command line text (warning prompt)

File properties
dn = Dash (unset property)
dr = Read permission bit
dw = Write permission bit
dx = Executable permission bit
dp = SUID, SGID, sticky bit
dg = File ID (UID, GID) whenever the current user owns the file or is in the file's group
dd = Last modification time
dz = Directories size
do = Octal value for file properties

Miscellaneous interface elements
bm = bookmarked directory in the bookmarks screen
fc = files counter
df = default color
dl = dividing line
el = ELN color
mi = misc indicators (disk usage, sort method, bulk rename, jump database list)
ts = matching suffix for possible TAB completed entries
tt = tilde for trimmed file names
wc = welcome message
wsN = color for workspace N (1-8)
xs = exit code: success
xf = exit code: failure

Color codes are just traditional ANSI color codes less the escape character and the final 'm'. Thus, for instance, if you want non−empty directories to be bold blue, add this to the *FiletypeColors* line in the corresponding color scheme file: **di=01;34**. If you want ELN's to be red, add this code to the *InterfaceColors* line: **el=00;31**

Color codes can be used for file extensions as well using this format: *.ext=color. For example, to print C source files in bold green, add this to the *ExtColors* line in the corresponding color scheme file: ***.c=01;32**

Color variables

Up to 64 custom color variables can be used via the *define* keyword to make it easier to build and read theme files. Example:

```
define RED=00;31
define MY_SPECIAL_COLOR=04;38;2;255;255;0;48;2;0;14;191

FiletpeColors="di=RED:"
InterfaceColors="el=MY_SPECIAL_COLOR:"
```

These variables can only be used for *FiletypeColors*, *InterfaceColors*, *ExtColors*, and *DirIconColor*. The *Prompt* and *WarningPromptStr* lines use full ANSI escape sequences instead.

Though by default CliFM uses only 8 colors (16 with the high intensity variant), you can use 256 and RGB colors as well. Example:

fi=04;38;2;245;76;00;48;2;00;00;255

will print regular files underlined and using a bold orange RGB color on a blue background. In this case, just make sure to use a terminal emulator supporting RGB colors. To test your terminal color capabilities use the *colors.sh* script (in the plugins directory).

NOTE: It might happen that, for some reason, you need to force CliFM to use colors despite the value of the **TERM** variable. The OpenBSD console, for example, sets **TERM** to *vt220* by default, which, according to the *terminfo* database, does not support color. However, the OpenBSD console does actually support color. In this case, you can set the **CLIFM_FORCE_COLOR** to either *true* or *1* to use color even if the value of **TERM** says otherwise.

To see a colored list of the currently used file color codes run *cc* or *colors* in CliFM.

To run colorless use the *--no-color* command line option or set either **CLIFM_NO_COLOR** or **NO_COLOR** environment variables to any value. For more information about the no-color initiative see *https://no-color.org/*

For a full no-color experience recall to edit your prompt removing all color codes.

2. THE PROMPT

CliFM's prompt (regular and warning ones) is built from the *Prompt* line in the color scheme file following almost the same escape codes and rules used by the Bash prompt, except that it does not accept shell functions (like conditionals and loops). Command substitution (in the form $(cmd)), string literals, and escape codes can be used to build the prompt line and its colors. This is a list of supported escape codes:

\e: Escape character

\s: The name of the shell (everything after the last slash) currently used by CliFM

\S: Current workspace number

\l: Print an 'L' if in light mode

\P: The current profile name

\u: The username

\H: The full hostname

\h: The hostname, up to the first '.'

\n: A newline character

\r: A carriage return

\a: A bell character

\d: The date, in abbreviated form (ex: "Tue May 26")

\t: The time, in 24−hour HH:MM:SS format

\T: The time, in 12−hour HH:MM:SS format

\@: The time, in 12−hour am/pm format

\A: The time, in 24−hour HH:MM format

\w: The full current working directory, with $HOME abbreviated with a tilde

\W: The basename of $PWD, with $HOME abbreviated with a tilde

\p: A mix of the two above, it abbreviates the current working directory only if longer than PathMax (a value defined in the configuration file).

\z: Exit code of the last executed command. Green in case of success and bold red in case of error

\$ '#', if the effective user ID is 0, and '$' otherwise

\nnn: The character whose ASCII code is the octal value nnn

****: A literal backslash

\[: Begin a sequence of non−printing characters. This is mostly used to add color to the prompt line

\]: End a sequence of non−printing characters

The following files statistics escape codes are also recognized (not available in light mode)

\D: Amount of sub-directories in the current directory

\R: Amount of regular files in the current directory

\X: Amount of executable files in the current directory

\.: Amount of hidden files in the current directory

\U: Amount of SUID files in the current directory

\G: Amount of SGID files in the current directory

\F: Amount of FIFO/pipe files in the current directory

\K: Amount of socket files in the current directory

\B: Amount of block device files in the current directory

\C: Amount of character device files in the current directory

\x: Amount of files with capabilities in the current directory

\L: Amount of symbolic links in the current directory

\o: Amount of broken symbolic links in the current directory

\M: Amount of multi-link files in the current directory

\E: Amount of files with extended attributes in the current directory.TP

\O: Amount of other-writable files in the current directory

*****: Amount of files with the sticky bit set in the current directory

\?: Amount of files of unknown file type in the current directory

\!: Amount of unstatable files in the current directory

By default, for example, CliFM's prompt line is this:

```
"\[\e[0m\][\S\[\e[0m\]]\l \A \u:\H \[\e[00;36m\]\w\n\[\e[0m\]<\z\[\e[0m\]>\[\e[0;34m\]
\$\[\e[0m\] "
```

which once decoded should look something like this:

```
[1] 13:45 user:hostname /my/path
<0> $
```

with the workspace number and the path printed in cyan, the last exit status in green, and the dollar sign in blue.

A more "classic" prompt could be construed as follows:

```
"\u@\U \w> "
```

or, using now command substitution:

```
"$(whoami)@$(hostname) $(pwd)> "
```

Advanced prompt customization

Besides commands substitution, which allows you to include in the prompt any information you like via shell scripts or simple shell commands, the use of Unicode characters allows you to build colorful and modern prompts.

Inserting Unicode characters in the prompt can be made in two ways:

a) Pasting the character itself using a text editor

b) Entering the octal code corresponding to the character. Use **hexdump**(1) as follows to get the appropriate hex code:

```
echo -ne "[paste the char here]" | hexdump -c
```

The first line of the output will be something along these lines:

```
00000000  256 234 356             |...|
```

In this case, the octal code is: "256 234 356". So, to insert this Unicode character in the prompt, add it as follows:

```
Prompt="... \256\234\356 ..."
```

Note: Make sure you have installed a font able to display Unicode characters.

A few advanced prompt examples can be found in the prompts file in our Github site. Take a look at <https://github.com/leo-arch/clifm/wiki/Customization#the-prompt>

A simple use case for the files statistics escape codes

We all want to keep our systems safe. One of the many ways to get a bit of safety is by checking that there is not file in our file system that could somehow endanger our machines. SUID, SGID, executable, and other-writable files are to be count among these dangers. This is why it could be useful to build a little files scanner for our prompt using the above mentioned files statistics escape codes. This is the code for our scanner:

```
"\[\e[0m\]\[\e[1;31m\]\U\[\e[0m\]:\[\e[1;33m\]\G\[\e[0m\]:
\[\e[1;32m\]\O\[\e[0m\]:\[\e[1;34m\]\X\[\e[0m\]"
```

By adding this code to our prompt line, we get something like this:

```
24:2:-:2389
```

This tells us that in the current directory we have 24 SUID files (printed in bold red), 2 SGID files (bold yellow), no other-writable file, and 2389 executable files.

NOTE: A predefined prompt with this files scanner integrated can be found in the *prompts.cfm* file.

NOTE 2: Most of the information these escape codes rely on depends on **stat**(3). Now, since **stat**(3) is not used when running in light mode (for performance reasons), this information won't be available in light mode either.

Prompt notifications

A bold red 'R' at the left of the prompt reminds the user that the program is running as root. A bold green asterisk indicates that there are elements in the Selection Box. In the same way, a yellow 'T' means that there are currently files in the trash can, just as a bold blue 'S' means that the program is running in stealth mode. Finally, CliFM makes use of three kind of messages: errors (a red 'E' at the left of the prompt), warnings (a yellow 'W'), and simple notices (a green 'N').

If *PromptStyle* is set to "custom" in the configuration file, the above notifications won't be printed by the prompt, and the information necessary to construct them will be exported to the envitonment to let your custom prompt handle them itself. See the **ENVIRONMENT** section below.

The Warning Prompt

The suggestions system includes a secondary, warning prompt, used to highlight wrong/invalid/non-existent command names. Once an invalid command is entered, the regular prompt will be switched to the warning prompt and the whole input line will turn dimmed red (though it can be customized to fit your needs).

The wrong command name check is omitted if the input string:

Is quoted (ex: "string" or 'string')
Is bracketed (ex: (string), [string], or {string})
It starts with a stream redirection character (ex: <string or >string)
Is a comment (ex: #string)
It starts with one or more spaces
Is an assignment (ex: foo=var)
It is escaped (ex: \string)

The warning prompt could be customized by means of the same rules used by the regular prompt. To use a custom warning prompt just modify the *WarningPromptStr* line in the color scheme file. It defaults to

"\[\e[0m\]\[\e[00;02;31m\](!) > "

the last line of the regular prompt will become "**(!) >** ", printed in a dimmed red color, including the input string.

To change the color of the input text while in the warning prompt use the **wp** color code (see the **COLOR CODES** section above). It defaults to dimmed red, just as the warning prompt itself.

To disable this feature use the *--no-warning-prompt* command line option or set the *WarningPrompt* option to **false** in the configuration file.

NOTE: Bear in mind that the warning prompt depends on the suggestions system, so that it won't be available if this system is disabled.

3. THE DIVIDING LINE

The line dividing the current list of files and the prompt. It could be customized via the *DividingLine* option in the color scheme file to fit your prompt design and/or color scheme.

DividingLine accepts one or more ASCII or Unicode characters (in both cases you only need to type/paste here the chosen character(s)). If only one character is specified (by default, "-"), it will be repeatedly printed to fulfill the current line up to the right edge of the screen or terminal window. If you don't want to cover the whole line, just specify three or more characters, in which case only these characters (and no more) will be used as dividing line. For example: "------->". To use an empty line, set *DividingLineChar* to "0" (that is, as a character, not as a number). Finally, is this value is not set, a special line drawn with box-drawing characters will be used (box-drawing characters are not supported by all terminal-emulators).

The color of this line is set via the *dl* color code in the color scheme file. Consult the **COLOR CODE** section above for more information.

4. FZF WINDOW

Refer to the **TAB completion** section below.

10. BUILT–IN EXPANSIONS

The SEL keyword

CliFM will automatically expand the **'sel' keyword**: 'sel' indeed amounts to 'file1 file2 file3 ...' In this way, you can use the 'sel' keyword with any command.

If you want to set the executable bit on several files, for instance, simply select the files you want and then run this command: 'chmod +x sel'. Or, if you want to copy or move several files into some directory: 'cp sel 12', or 'mv sel 12' (provided the ELN 12 corresponds to a directory), respectively.

If the destiny directory is omitted, selected files are copied into the current working directory, that is to say, 'mv sel' amounts to 'mv sel .'.

To trash or remove selected files, simply run 'tr sel' or 'rm sel' respectively. The same goes for wildcards and braces: 'chmod +x *', for example, will set the executable bit on all files (excluding hidden files) in the current working directory, while 'chmod +x file{1,2,3}' will do it for file1, file2, and file3 respectively.

If using the FZF mode for TAB completion (see below), you can operate only on *some* selected files as follows: type 'CMD sel' and, without appending any space char, press TAB: the list of selected files will be displayed. Choose one or more of them (use TAB to mark entries) to operate only on those specific files. For example, to print the file properties of some specific selected files: p sel->TAB, select the files you want via TAB, press Enter or Right (marked files will be inserted in the command line), and the press Enter, as usual.

TAB completion

There are two modes for TAB completion: *standard* (interface provided by readline), and *fzf*, which depends on **FZF** (*https://github.com/junegunn/fzf*) (version 0.18.0 or later). To enable the *fzf* mode (the standard mode is used by default), you can either use the *--fzftab* command line option, set *TabCompletionMode* to *fzf* in the configuration file, or set **CLIFM_USE_FZF** environment variable to either *1* or *true*. Example: 'export CLIFM_USE_FZF=true'.

If using the *fzf* mode, the completions interface could be customized using the *FzfTabOptions* option in the color scheme file. --height, --margin, +i/-i, --read0, --query, and --ansi will be appended to set up some details of the completions interface. Set this value to *none* to pass no option, to the empty string to load the default values, or to any other custom value. Unless set to *none*, any option specified here will override **FZF_DEFAULT_OPTS**.

Default values for this option are:

--color=16,prompt:6,fg+:-1,pointer:4,hl:5,hl+:5,gutter:-1,marker:2 --bind tab:accept,right:accept,left:abort --inline-info --layout=reverse-list

Consult **fzf**(1) for more information.

If set neither in *FzfTabOptions* nor in **FZF_DEAFULT_OPTS** (in this order), the height of the FZF window is set to the default value: 40% of the current terminal amount of line/rows.

To use FZF global values (defined in **FZF_DEFAULT_OPTS**), just set *FzfTabOptions* to *none*.

Besides the default *TAB completion* for command **names and paths**, you can also expand **ELN's** using the TAB key. Example: type 'o 12', press TAB, and it becomes 'o filename ', or, if 12 refers to a directory, 'o dir/'. CliFM uses a Bash−style quoting system, so that this file name: "this is a test@version{1}" is expanded as follows: this\ is\ a\ test\@version\{1\}

ELN's and **ELN ranges** will be also automatically expanded, provided the corresponding ELN's actually exist, that is to say, provided some file name is listed on the screen under those numbers. For example: 'diff 1 118' will only expand '1', but not '118', if there is no ELN 118. In the same way, the range 1−118 will only be expanded provided there are 118 or more elements listed on the screen.

Since ranges could be a bit tricky, TAB completion is available to make sure this range actually includes the desired file names.

If this feature somehow conflicts with the command you want to run, say, 'chmod 644 ...', because the current amount of files is equal or larger than 644 (in which case CliFM will expand that number), then you can simply run the command as external: ';chmod 644...'

TAB completion for commands, paths, environment variables, bookmarks, profiles, color schemes, file tags, command history, directory history (via the *jump* command), remote resources, sort methods, ranges*****, the 'sel' keyword*****, trashed files*****, plus the deselect***** and the open-with commands (*ow*) is also available. To make use of the bookmarks completion, make sure to specify some name for your bookmarks, since these names are used by the completion function.

***** When using FZF mode for TAB completion, multi-selection is available: Press TAB to expand possible selections, then press TAB again to mark desired entries. Once desired entries are marked, press Enter or the Right arrow key: marked entries will be inserted into the command line. Multi-selection is also available for the following commands, provided there is no slash in the query string: *ac*, *ad*, *bb*, *br*, *d/dup*, *p/pr/prop*,

r, *s*, *t/tr/trash*, and *te*.

Of course, combinations of all these features is also possible. Example: 'cp sel file* 2 23−31 .' will copy all selected files, plus all files whose name starts with "file", plus those files corresponding to the ELN's 2, and 23 to 31, into the current working directory.

In addition to completions and expansions, an *auto-suggestions system* is also available. See the **AUTO-SUGGESTIONS** section below.

11. RESOURCE OPENER

As CliFM's built-in resource opener, *Lira* takes care of opening files when no opening application has been specified in the command line. It does this by automatically parsing a MIME list file (see the **FILES** section below): it looks first for a matching pattern (either a MIME type or a file name), then checks the existence of the command associated to this pattern, and finally executes it.

Lira is controlled via the *mime* command. File associations are stored in the MIME list file.

When running for the first time, or whenever the MIME list file cannot be found, CliFM will copy the MIME definitions file from the **DATADIR** directory (usually */usr/share/clifm/mimelist.cfm*) to the local configuration directory.

Lira will check the file line by line, and if a matching line is found, and if at least one of the specified applications exists, this application will be used to open the corresponding associated file. Else, the next line will be checked. In other words, the precedence order is top to bottom (for lines) and left to right (for applications).

NOTE: In case of directories (whose MIME type is *inode/directory*), the entry will be used **only** for the open-with command (*ow*).

This MIME list file follows a few simple syntax rules:

Each line in the MIME list file consists of:

a) 'X' or '!X' to specify GUI and non-GUI environments respectivelly;

b) 'N' to instruct *Lira* to match a file name instead of a MIME type;

c) A left value, containing either a file name or a MIME type to be matched. Regular expressions are supported;

d) A right value, a list of semicolon separated commands (and optionally the commands parameters) to be associated to the corresponding left value;

Note that the syntax departs here from the Freedesktop specification in that we do not rely on desktop files (mostly used by desktop environments), but rather on commands and parameters. In general thus, the syntax is this:

```
[!]X[:N]:REGEX=CMD [ARGS] [%f];CMD [ARGS] [%f]; ...
```

Use the **%f** placeholder to specify the position of the file name to be opened in the command. For example, 'mpv %f --terminal=no' will be translated into 'mpv FILE --terminal=no'. If the placeholder is not specified, the file name will be appended to command string. Thus, this: 'mpv --terminal=no' amounts to this: 'mpv --terminal=no FILE'.

Running the opening application in the background:

For GUI applications:

```
APP %f &>/dev/null &
```

For terminal applications:

```
TERM -e APP %f &>/dev/null &
```

Replace 'TERM' and 'APP' by the corresponding values. The *-e* option might vary depending on the terminal emulator used (TERM).

NOTE: In case of archives, the built-in *ad* command could be used as opening application.

NOTE 2: Environment variables (e.g. **$EDITOR**, **$VISUAL**, **$BROWSER**, and even **$PAGER**) are also recognized by *Lira*. You can even set custom environment variables to be used exclusively by CliFM. For example, you can set **CLIFM_TERM**, **CLIFM_EDITOR**, and **CLIFM_PDF**, and then use them to define some associations:

```
X:text/plain=$CLIFM_TERM -e $CLIFM_EDITOR %f &
X:N:.*\.pdf$=$CLIFM_PDF %f &
```

Examples:

Match a full file name:

```
X:N:some_filename:leafpad;mousepad;kate;gedit
```

Note: the 'N' character indicates that this rule is intended to match a **file name** instead of a MIME type, just as 'X' means that this rule is aimed to **graphical** environments and '!X' that it is aimed rather to **non-graphical** environments.

Note 2: If the file name contains a dot, quote it like this: some_filename.ext (to prevent the REGEX parser from interpreting the dot)

Match multiple file names (starting with 'str'):

```
X:N:^str.*:leafpad;mousepad;kate;gedit
```

Match a single extension:

```
X:N:.*\.txt$:leafpad;mousepad;kate;gedit
!X:N:.*\.^txt$:nano;vim;vi;emacs
```

Match multiple extensions:

```
X:N:.*\.(sh|c|py|pl)$:geany;leafpad;nano
```

Match single mimetype:

```
X:^audio/mp3$=mpv %f --terminal=no;ffplay -nodisp -autoexit;mpv;mplayer
```

Match mutiple mimetypes:

```
X:^audio/.*=mplayer;mplayer2;vlc;gmplayer;smplayer;totem
```

In case of MIME types, you can also write the entire expression without relying on any regular expression. For example:

```
!X:text/plain=$TERM -e $EDITOR %f &>/dev/null &
```

For more information take a look at the mimelist file itself (*F6* or *mm edit*).

Using CliFM as a standalone resource opener

Thought CliFM is a file manager, it can be used as a simple resource opener via the --open command line option. For example:

```
clifm --open /path/to/my_file.jpg
clifm --open /path/to/my_dir
clifm --open https://some_domain
```

NOTE: When opening web resources CliFM will query the mimelist file using text/html as MIME type. Whatever association it finds for this specific MIME type will be used to open the web resource.

Positional parameters could be used as well, provided the parameter does not point to a directory name, in which case it will be used as CliFM starting path. For instance:

```
clifm /path/to/my_file.jpg
clifm https://some_domain
```

12. AUTO-SUGGESTIONS

Gemini is a built-in suggestions system (similar to that provided by the Fish shell). As you type, *Gemini* will suggest possible completions right after the current cursor position.

The following checks are availabe (the order can be customized, see below):

a. ELN's

b. CliFM commands and parameters (including the *sel* keyword)

c. Entries in the command history list (already used commands)

d. File names in the current working directory

e. Entries in the jump database

f. Aliases names

g. Bookmarks names

h. Program names in **PATH**

i. Shell builtins

NOTE: The shell name is taken from */bin/sh*. The following shell are supported: bash, dash, fish, ksh, tcsh, and zsh. Command names are checked in the following order: CliFM internal commands, commands in **PATH**, and shell builtins.

To accept the entire suggestion, just press **Right** or **Ctrl-f**: the cursor will move to the end of the suggested command and the suggestion color will change to that of the typed text; next, you can press **Enter** to execute the command as usual. Otherwise, if the suggestion is not accepted, it will be simply ignored and you can continue editing the current command line however you want.

To aceppt the first suggested word only (up to first slash or space), press rather **Alt-Right** or **Alt-f**. Not available for ELN's, aliases and bookmarks names.

Bear in mind that suggestions for ELN's, aliases and bookmarks names, and the jump function (invoked by the *j* command) do not work as the remaining suggestions: they do not suggest possible completions for the current input, but rather the value pointed to by it. For example, if you type "12" and the current list of files includes a file name whose ELN is '12', the file name corresponding to this ELN will be printed next to "12" as follows: **12_ > filename** (where the underscore is the current cursor position). Press 'Right' or 'Ctrl-f' to accept the suggestion, in which case the text typed so far will be replaced by the suggestion.

The order of the suggestion checks could be customized via the *SuggestionStrategy* option in the configuration file. Each check is assigned a lowercase letter:

a = Aliases names
b = Bookmark names
c = Possible completions
e = ELN's
f = Files in the current directory
h = Entries in the commands history
j = Entries in the jump database

The value taken by *SuggestionStrategy* is a string of seven(7) characters containing the above letters. The letters order in this string specifies the order in which the suggestion checks will be performed. For example, to perform all checks in the same order above, the value of the string should be **abcefhj** (without quotes). Or, if you prefer to run the history check first: **habcefj**. Finally, you can ignore one or more checks using a dash (-). So, to ignore the bookmarks and aliases checks, set *SuggestionStrategy* to **h--cefj**. The default value for this option is **ehfjbac**.

Note: The check for program names in **PATH** is always executed at last, except when the *ExternalCommands* option is disabled, in which case suggestions for them are simply not displayed.

Suggestions will be printed using one of the following color codes (see the **COLOR CODES** section above):

sf: Used for file and directory names. This includes suggestions for ELN's, bookmarks names, files in the current directory, and possible completions. Default value: 02;04;36 (dimmed underlined cyan)

sh: Used for entries in the commands history. Default value: 02;35 (dimmed magenta)

sc: Used for aliases and program names in **PATH**. Default value: 02;31 (dimmed red)

sx: Used for CliFM internal commands and parameters. Default value: 02;32 (dimmed green)

sp: Greater-than sign (>) used when suggesting ELN's, bookmarks, and aliases names. Default value: 02;31 (dimmed red)

You can set **SuggestFiletypeColor** to "true" in the configuration file to use the color of the file type of the current file name (as set in the color scheme file) instead of the value of *sf*. For example, if a suggestion is printed for a file that is a symbolic link, *ln* or *or* (if a broken link) will be used instead of *sf*.

13. SHELL FUNCTIONS

CliFM includes a few shell functions to perform specific actions (cd-on-quit, file-picker, and subshell-notice). Take a look at the corresponding files, in */usr/share/clifm/functions*, and follow the instructions. Needles to say, you can write your own functions.

14. PLUGINS

Plugins are just scripts or programs (written in any language) able to add, extend or improve CliFM functionalities. Though several plugins are provided at installation time (in the *plugins* directory), you can write your owns as you like, with any language you like, and for whatever goal you want. Writing plugins is generally quite easy; but your mileage may vary depending on what you are trying to achieve. A good place to start is examining the provide plugins and reading the *actions* command description, and the **ENVIRONMENT** and **FILES** sections below.

A convenient helper script is provided to get a consistent look across all plugins, specially those running FZF. This helper script is located in *DATADIR/clifm/plugins/plugins-helper*, but it will be overridden by *XDG_CONFIG_HOME/clifm/plugins/plugins-helper* if found. The location of this file is set by CliFM itself in the **CLIFM_PLUGINS_HELPER** environment variable to be used by plugins. Source the file and

use any of the functions and variables provided by it to write a new FZF plugin:

```
# Source our plugins helper
if [ -z "$CLIFM_PLUGINS_HELPER" ] || ! [ -f "$CLIFM_PLUGINS_HELPER" ]; then
    printf "CliFM: Unable to find plugins-helper file\n" >&2
    exit 1
fi
# shellcheck source=/dev/null
. "$CLIFM_PLUGINS_HELPER"
```

Plugin	Dependencies	Action name
batch_create		bn
batch_copy		bcp
bm_import		bi
clip	xclip	clip
colors		
cprm	scp, fzf	cr
disk_analyzer	du, fzf	da
dragondrop	dragon or dragon-drag-and-drop	dr
fdups	find, md5sum, sort, uniq, xargs, sed, stat	fdups
finder	fzf or rofi	+
fzcd	fzf	_
fzfdirhist	fzf	dh
fzfhist	fzf	h
fzfnav	fzf	-
-archives	atool, bsdtar or tar	
-images	kitty terminal, imagemagick, and ueberzug or viu or catimg or img2txt or pixterm	
-fonts	fontpreview or fontforge	
-docs	libreoffice, catdoc, odt2txt, pandoc	
-PDF	pdftoppm, pdftotext or mutool	
-epub	epub-thumbnailer	
-DjVu	djvulibre or djvutxt	
-Postscript	ghostscript	
-videos	ffmpegthumbnailer	
-audio	ffmpeg, mplayer or mpv	

-web	w3m, linx, elinks or pandoc	
-markdown	glow	
-highlight	bat, highlight or pygmentize	
-torrent	transmission-cli	
-json	python or pq	
-file info	exiftool, mediainfo or file	
fzfsel	fzf	*
fzfdesel	fzf	**
git_status	git	
ihelp	fzf or rofi	ih
img_viewer	sxiv, feh or lsix	i
jumper	fzf or rofi	++
kbgen	(C source file)	kben
mime_list	file, fzf	ml
music_player	mplayer	music
pdf_viewer	pdftotext	ptot
recur_rm	find, fzf	rrm
rgfind	fzf and ripgrep	//
update		
vid_viewer	ffmpegthumbnailer	vid
wallpapper_setter	feh, xloadimage or hsetroot	wall

NOTE: The *fzfnav* plugin uses **fzf**(1) to navigate the file system and *BFG* (a script located in the plugins directory) to show previews (to show files preview *BFG* requires **ueberzug** to be installed or the Kitty protocol via the Kitty terminal). A configuration file (BFG.cfg, in the plugins directory itlsef) is provided to customize the previewer's behavior.

In addition to the built-in BFG previewer, *fzfnav* supports the use of both Ranger's *scope.sh* script and **pistol**. To use **scope**, just edit the *BFG* configuration file and set USE_SCOPE to 1 and SCOPE_FILE to the correct path of the *scope.sh* file (normally *$HOME/.config/ranger/scope.sh*). To use **pistol** instead, set USE_PISTOL to 1.

NOTE 2: The *git_status* plugin is not intended to be used as a normal plugin, that is, executed via an action name, but rather to be executed as a prompt command:

```
promptcmd /usr/share/clifm/plugins/git_status.sh
```

Whereas this plugin provides basic Git integration, it could be easily modified (it is just a few lines long) to include whatever git function you might need.

Take a look at the Wiki for more information: *https://github.com/clifm/wiki/Advanced#plugins*

15. AUTOCOMMANDS

Heavily inspired by **Vifm**, the *autocommands* function allows the user a fine-grained control over CliFM settings. It is mostly devised as a way to improve performance for remote file systems (usually slower than local ones) by allowing you to turn off some features (like the files counter) that might greatly affect performance under some circumstances (like remote connections). However, this function is not restricted to this use: use it for whatever purpose you find useful.

Add a line preceded by the *autocmd* keyword to the config file. The general syntax is: **autocmd PATTERN cmd,cmd,cmd**

PATTERN is a glob expression to match directory names. If no glob metacharacter is provided, the string will be compared as is to the current working directory. To invert the meaning of a pattern, prepend an exclamation mark: **!PATTERN**

PATTERN is followed by a comma separated list of commands:

!CMD: The exclamation mark allows you to run shell commands, custom binaries or scripts

The following codes are used to control CliFM's files list:

Code	Description	Example
cs	Color scheme	cs=amber-256
fc	Files counter	fc=0
hf	Hidden files	hf=0
lm	Light mode	lm=1
lv	Long/detail view	lv=0
mf	Max files	mf=100
mn	Max file name length	mn=20
pg	Pager	pg=0
st	Sort method	st=5
sr	Reverse sort	sr=1

A few example lines:

1. Run in light mode and disable the files counter for the remotes folder:
autocmd /media/remotes/** lm=1,fc=0

2. Just a friendly reminder:
autcomd ~/important !printf "Important: keep your fingers outta here!\n" && read −n1

3. This directory has thousands of files. Show only the first hundred and enable the pager:
autocmd /usr/bin mf=100,pg=1

4. Lots of media files (with large file names). Trim file names to 20 chars max and run the files previewer:
autocmd ~/Downloads mn=20,!~/.config/clifm/plugins/fzfnav.sh

5. Mmm, just because I can. Be creative!
autocmd /home/user hf=0,cs=amber-256,lv=1
autocmd / lv=1,fc=0,cs=solarized,st=5

The first example is the recommended configuration for directories containing remote file systems

As seen in the fifth example, plugins could be used here as well: in this case, we want to run *fzfnav* (to make use of the files preview capability) whenever we enter into the *Downloads* directory, usually containing videos, music, and images.

NOTE: If you decide to use a plugin, bear in mind that it won't be able to communicate with CliFM, because autocommands always executes commands as external applications using the system shell.

Autocommand files: *.cfm.in* and *.cfm.out*

Two files are specifically checked by the autocommands function: *.cfm.in* and *.cfm.out*.

The content of these files is a single instruction, either a shell command or, if you need more elaborated stuff, a script (or custom binary). Note that codes to modify CliFM's settings (as described above) are not available here.

If a directory contains a file named *.cfm.in*, CliFM will execute (via the system shell) its content when **entering** this directory (before listing files). If the file is named rather *.cfm.out*, its content will be executed immediately after **leaving** this directory (and before listing the new directory's content).

For example, if you want a simple notification whenever you enter or leave your home directory, just create both *.cfm.in* and *.cfm.out* files in the home directory with the following content:

For *.cfm.in*:
```
printf "Entering %s ...\n" "$PWD"
```

For *.cfm.out*:
```
printf "Leaving %s ...\n" "$OLDPWD"
```

16. FILE TAGS

Etiqueta is CliFM's built-in files tagging system

1. How Etiqueta works?

File tags are created via symlinks using an specific directory under the user's profile:
${XDG_CONFIG_DIR:-/home/USER/.config}/clifm/profiles/USER/tags

Every time a new tag is created, a new directory named as the tag itself is created in the tags directory. Tagged files are just symbolic links to the actual files created in the appropriate directory. For example, if you tag *~/myfile.txt* as *work*, a symbolic link to *~/myfile.txt*, named *myfile.txt* will be created in *tags/work*.

2. Handling file tags

tag is the main **Etiqueta** command and is used to handle file tags. Its syntax is as follows:

tag [ls, list] [new] [rm, remove] [mv, rename] [untag] [merge] [FILE(s)] [[:]TAG]

NOTE: the *:TAG* notation is used for commands taking both file and tag names: 'tag FILES(s) :TAG ...', to tag files, and 'tag untag :TAG file1 file2', to untag files. Otherwise, *TAG* is used (without the leading colon). For example: '*tag new docs*', to create a new tag named *docs*, or 'tag remove png', to delete the tag named *png*.

The following command shortcuts are available:
ta Tag files
td Delete tag(s)

tl List tags or tagged files
tm Rename (mv) tag
tn Create new tag(s)
tu Untag file(s)
ty Merge two tags

3. Usage examples

- List available tags:

tl

- Tag all .PNG files in the current directory as both *images* and *png*:

ta *.png :images :png

NOTE: Tags are created if they do not exist

- Tag all selected files as *special*:

ta sel :special

- List all files tagged as work and all files tagged as *documents*:

tl work documents

- Rename the tag *documents* as *docs*:

tm documents docs

- Merge the tag *png* into *images*:

ty png images

NOTE: All files tagged as *png* will be now tagged as *image*s, and the *png* tag will be removed.

- Remove the tag *images* (untag all files tagged as *images*):

td images

- Untag a few files from the *work*:

tu :work file1 image.png dir2

NOTE: TAB completion is available to complete tagged files. If using the FZF mode, multi-selection is also available via the TAB key.

4. Operating on tagged files

The *t:TAG* construct (or tag expression) is used to operate on tagged files via any command, be it internal or external. A few examples:

- Print the file properties of all files tagged as *docs*:

p t:docs

NOTE: TAB completion is available to expand tag expressions into one or more of the corresponding tagged files. If using the FZF mode, multi-selection is also available via the TAB key.

- Remove all files tagged as *images*:

r t:images

- Run **stat**(1) over all files tagged as *work* and all files tagged as *docs*:

stat t:work t:docs

4.1 Operating on *specific* tagged files

NOTE: This feature, as always when multi-selection is involved, is only available when TAB completion mode is set to FZF. See the **TAB completion** subsection of the **BUILT-IN-EXPANSIONS** section above.

You might not want to operate on **all** files tagged as some specific tag, say *work*, but rather on **some** files tagged as *work*. TAB completion is used to achieve this aim.

Let's suppose you have a tag named *work* which contains ten tagged files, but you need to operate (say, print the file properties) only on two of them, say, *work1.odt* and *work2.odt*:

p t:work<TAB>

The list of files tagged as *work* will be displayed via FZF. Now just mark the two files you need using **TAB**, press **Enter** or **Right**, and the full path of both files will be inserted into the command line. So, '**p t:work**' will be replaced by '**p /path/to/work1.odt /path/to/work2.odt**'.

17. STANDARD INPUT (STDIN)

CliFM is able to read and list paths from the input stream. Each path in the list should be an absolute path, terminated with a new line character (\n) and stripped from extra characters not belonging to the path itself. The size of the input stream buffer is 262MiB (65536 paths, provided each path takes PATH_MAX bytes (4096 by default)).

Each file passed via standard input will be stored as a symbolic link pointing to the original file in a temporary directory (which will be deleted at program exit) and listed on startup. Bear in mind that the restore last path function is disabled when listing in this way. Examples:

```
ls -Ad /var/* | clifm
```

This command will pass all files in /var to CliFM

If you need to perform more specific queries, you can use *find* as follows:

```
find −maxdepth 1 −size +500k −print0 | tr '\0' '\n' | sed 's/\.\///g' | clifm
```

The above command will pass all files in the current directory bigger than 500KiB to CliFM.

You can also use stream redirection:

```
ls -Ad $PWD/* > list.txt
clifm < list.txt
```

All operations performed on these symbolic links (provided the current working directory is the temporary directory where all these files are stored) will be performed on the target files and NOT on the symbolic links themselves.

18. NOTE ON SPEED

CliFM is by itself quite fast by default, but if speed is still an issue, it is possible to get some extra performance.

The two most time consuming features are:

1) The files counter, used to print the amount of files contained by listed directories. Disabling this option produces a nice performance boost.

2) In normal mode, **fstatat**(3) is used to gather information about listed files. Since this function, especially when executed hundreds (and even thousands) of times, is quite time consuming, the *light mode* was implemented as an alternative listing process omitting all calls to it. When running in light mode, however, only basic file classification is performed, namely, that provided by the *d_type* field of a dirent struct (see **readdir**(3)). Bear in mind, nonetheless, that whenever _DIRENT_HAVE_D_TYPE was not set at compile time, or in case of a DT_UNKNOWN value for a given entry (we might be facing a file system not returning the *d_type* value, for example, loop devices), CliFM will fall back to **stat**(3) to get basic files classification.

Besides these two features, a few more things can be disabled to get some extra speed (though perhaps unnoticeable): icons (if enabled), columns, colors, and, if already running without colors, file type indicators. Because listing lots of files could be expensive and time consuming, you can also try to limit the amount of files printed for each visited directory (see the *mf* command above).

Despite the above, however, it is important to bear in mind that listing speed does not only depend on the program's code and enabled features, but also on the terminal emulator used. Old, basic terminal emulators like Xterm, Aterm, and the kernel built-in console are really slow compared to more modern ones like Urxvt, Lxterminal, ST, and Terminator, to name just a few.

If using Xterm, a nice speed boost is provided by the fast scroll option: set *fastScroll* to true in your *~/.Xresources* file. See **xterm**(1).

19. KANGAROO FRECENCY ALGORITHM

The directory jumper function is designed to learn the navigation habits of the user. The information is stored in a database (see the **FILES** section below) used to get the best match for a given string provided by the user. In this sense, Kangaroo is like a quick, smart, and evolved cd function.

The information stored in the database, always per directory, is:
a) Number of visits
b) Date of first visit (seconds since the Unix epoch)
c) Date of the last visit
d) The full path of each visited directory

With this information it is possible to construct a ranking of directories to offer the user the most accurate matches for each query string. The matching algorithm takes into account mainly two factors: frequency and recency (which is why this kind of algorithm is often called a **frecency** algorithm).

After getting an initial list of matches based on the query string(s) entered by the user, the frequency algorithm is applied on each entry in the list. The algorithm is quite simple: **(visits * 100) / days-since-first-visit**. As a result, we get the average of visits per day since the day of the first visit (what we call the directory rank).

There are however some further steps in the ranking process: **Bonus points**.

Extra credits or penalties are assigned based on the directories **last access time** according to the following simple algorithms:
Within last hour: rank * 4
Within last day: rank * 2
Within last week: rank / 2
More than a week: rank / 4

If the last query string matches the **basename** of a directory, the entry for this directory has 300 extra credits. This is done simply because we normally use directory basenames as query strings: they are easier to remember.

In the same way, **pinned** directories get 1000 extra credits, **bookmarked directories** 500 credits, and directories currently in a **workspace** 300 credits.

For example: if the query string is "test", /media/data/test will be matched. Now, if this directory was accessed within the last hour, and its rank was 200, it becomes 800. But, because the search string matches its basename, it gets 300 extra credits, and, if this directory is in addition bookmarked and pinned, it gets 1500 extra credits. In this way the total rank of this directory in the matching process is 2600. In doing this, we have more chances of matching what the user actually wanted to match.

Once all entries in the initial list of matches have been filtered via the above procedure and ranked, we can return the best ranked entry. The higher rank a directory has, the more priority it has over the remaining entries in the initial list of matches.

Automatic maintenance is done on the database applying a few simple procedures:

a) Each entry in the database is checked at startup to remove non-existent directories.

b) Once the rank of a directory falls below MinJumpRank (by default 10), it is forgotten and deleted from the database. The MinJumpRank value can be customized in the configuration file. To make non-frequently visited directories disappear quicker from the database, increase this value.

c) Once the sum total of ranks reaches MaxJumpTotalRank (by default 100000), each individual rank is divided by a dynamic factor so that the total rank becomes less than or equal to MaxJumpTotalRank. If some rank falls in the process below MinJumpRank, it is removed from the database. MaxJumpTotalRank can be modified in the configuration file. The higher the value of MaxJumpTotalRank, the more time directories will be kept in the database.

NOTE: Directories visited in the last 24 hours will not be removed from the database, no matter what their rank is.

The idea of 'frecency' was, as far as I know, first devised and designed by Mozilla. See *https://wiki.mozilla.org/User:Mconnor/Past/PlacesFrecency*. However, it is also implemented, though using different algorithms, by different projects like **autojump**, **z.lua**, and **zoxide**.

20. ENVIRONMENT

The following variables are read at initialization time:

NO_COLOR
If set to any value, CliFM will run colorless

CLIFM_NO_COLOR
Same as **NO_COLOR**, but specific to CliFM

CLIFM_FILE_COLORS
A colon separated list of file type color codes in the same form specified above in the **COLOR CODES** section

CLIFM_EXT_COLORS
Same as above, but for file extensions

CLIFM_IFACE_COLORS
Same as above, but for different element of ClIFM's interface

CLIFM_FILTER
Define a file filter. If set, this variable overrides the *Filter* option in the configuration file

Except when running in stealth mode, CliFM sets the following environment variables:

CLIFM
This variable is set to the path of the configuration directory. By inspecting this variable other programs can find out if they were spawned by CliFM. It can also be used to quickly jump into the configuration directory: 'cd $CLIFM' or just '$CLIFM'

CLIFM_PLUGINS_HELPER
Set to the full path of the plugins-helper script used by many plugins.

CLIFM_PROFILE
This variable is set to the current profile of CliFM (if using two or more instances of CliFM under different profiles, the last onw wil be used). Specially useful to develop CliFM plugins on a per profile basis.

CLIFM_SELFILE
The path to the current selection file.

CLIFM_COLORLESS
Set to 1 if running colorless (via the **NO_COLOR** or **CLIFM_NO_COLOR** environment variables, or the *--no-color* command line option).

CLIFM_BUS
This variable contains the path of a pipe by means of which plugins can talk to CliFM. Just write to the pipe and CliFM will hear and handle the message immediately after the plugin's execution. If the message is a path, CliFM will run the open function, changing the current directory to the new path, if a directory, or opening it with the resource opener, if a file. Otherwise, if the message is not a path, it will be taken and executed as a command. Examples:

´echo "/tmp" > "$CLIFM_BUS"´ tells CliFM to change the current directory to /tmp

´echo "s *.png" > "$CLIFM_BUS"´ makes CliFM select all files in the current directory ending with ".png"

The pipe (CLIFM_BUS) is deleted immediately after the execution of its content and recreated before running any other plugin.

If *PromptStyle* is set to "custom" in the configuration file, the following variables are exported to the environment to be used, if needed, by your custom prompt:

CLIFM_STAT_SEL
Current amount of selected files

CLIFM_STAT_TRASH
Current amount of trashed files

CLIFM_STAT_MSG
Current amount of system messages

CLIFM_STAT_WS
Current workspace number

CLIFM_STAT_EXIT
Exit code of the last executed command

CLIFM_STAT_ROOT
1 if user is root (UID = 0), 0 otherwise

CLIFM_STAT_STEALTH
1 if running in stealth mode, 0 otherwise

21. SECURITY

Since CliFM executes OS commands, it needs to provide a way to securely run these commands, specially when it comes to untrusted environments. Two features are provided to achieve this aim: **secure environment** and **secure commands**.

Both features are aimed to protect the program and the system as such from malicious input, either coming from environment variables or from indirect input, that is to say, input coming not from the command line (in which is assumed that it is the user herself who is executing the given command), but from files: this is the case of default associated applications (the *mime* command), autocommands, (un)mount commands (via the *net* command), just as profile and prompt commands.

In an untrusted environment, an attacker could cause unexpected and insecure behavior (even command injection) using environment variables, or inject malicious commands via indirect input, commands which will be later executed by CliFM without the user's consent (i.e. automatically). This is why we provide a mechanism to minimize this danger: if running in an untrusted environment, the secure environment and secure commands features are there to prevent (at least as far as possible) this kind of attacks.

A) Secure environment

Programs inherit the environment from the parent process. However, if this inherited environment is not trusted, not secure, it is always a good idea to sanitize it using only sane values, preventing thus undesired and uncontrolled input that might endanger the program and the system itself.

The *secure-environment* function forces CliFM to run on a such a sanitized environment.

There are two secure-environment modes, the *regular*, and the *full* one. To enable the regular mode, run CliFM with the *--secure-env* command line option. Otherwise, enable the full mode using *--secure-env-full*.

a) *Regular*: in this mode, the inherited environment is cleared, though a few variables are preserved to keep CliFM running as stable as possible. These preserved variables are: **TERM**, **DISPLAY**, **LANG**, **TZ**, and, if FZF TAB completion mode is enabled, **FZF_DEFAULT_OPTS**.

The following variables are set in an environment agnostic way (that is, securely):
- **HOME**, **SHELL**, and **USER** are retrieved using **getpwuid(3)**
- **PATH** is set consulting **_PATH_STDPATH** (or **_CS_PATH** if the former is not available)
- **IFS** is set to a sane, hard-coded value: " \n\t"

b) *Full*: this mode is just like the regular mode, except that nothing is imported from the environment at all and only **PATH** and **IFS** are set (as described above). Everything else remains unset, and is the user's responsibility to set environment variables (via the export function), as needed. In this case, you might want to set, at least, **TERM**, and, if running in a graphical environment, **DISPLAY**.

Be aware that enabling secure-environment might break some functions, depending on the system configuration.

B) Secure commands

Some commands are automatically executed by CliFM: (un)mount commands (via the *net* command), opening applications (via *Lira*), just as prompt, profile, and autocommands. These commands are read from a configuration file and then executed. Now, if an attacker has access to any of these files, she might force CliFM to run any arbitrary command, and thereby possibly exposing the whole system.

Every time a command is thus automatically executed via the system shell (i.e. without the user's direct consent), the secure commands function performs three different, though intrinsically related tasks aimed to mitigate command injection and/or unexpected behavior:

a) Only command base names are allowed: *nano*, for instance, is allowed, while */usr/bin/nano* is not. In this way we can guarantee that only commands found in a sanitized **PATH** (see the point **c** below) will be executed. This is done in order to prevent the execution of custom binaries/scripts, for example: */tmp/exec_file*.

b) Commands are validated using a **whitelist** of safe characters (mostly to prevent stream redirection, conditional execution, and so on, for example, 'your_command;some_injected_command'). This set of safe characters slightly vary depending on the command being executed (because they use different syntaxes):

Net command: a−zA−Z −_.,/=
Prompt, profile, autocommands: a−zA−Z −_.,/"'
Mime command: a−zA−Z −_.,%&

Commands containing *at least one* unsafe character will be rejected. Of course, we cannot (and should not) prevent what looks like legitimate, benign commands from being executed. But we can stop commands that, in an untrusted environment, look suspicious. This is specially the case of stream redirection (>), pipes (|), sequential (;) and conditional execution (&&, ||), command substitution ($(cmd)), and environment variables ($VAR).

c) Unless already running in a secure environment (via the *−−secure−env* or *−−secure−env−full* options), a sanitized environment will be created for the command to be executed (returning afterwards to the original environment). The values for this secure environment are as follows:

PATH Taken from **_PATH_STDPATH** (or **_CS_PATH**)
IFS " \t\n"
USER, **HOME**, **SHELL** Retrieved from the password database via **getpwuid(3)**
LOGNAME Same as **USER**
DISPLAY, **TZ**, **LANG**, **TERM** Imported from the environment and sanitized
LC_ALL Same as **LANG**

22. MISCELLANEOUS NOTES

Sequential and conditional execution of commands:

For each of the internal commands (see the **COMMANDS** section above) you can use the semicolon to execute them sequentially and/or the double ampersand to execute them conditionally. Example: 'cmd1; cmd2 && cmd3'.

Though you can use here external commands as well, bear in mind that, whenever at least one internal command is involved in a chained list of commands, CliFM will take care of executing this list (simply because the system shell isn't able to understand any of these commands), so that no shell inter−process function (like pipes), nor any stream redirection or shell expression (like IF blocks or FOR loops) will be available. However, the shell is still used to run single external commands found in the chained list, but in isolation from the remaining commands in this list.

As a rule of thumb, when using chained commands make sure to always expand ELN's to avoid undesired consequences. If, for instance, you issue this command: 'touch aaa && r 3', you will end up deleting a file you were not intended to delete, simple because after the successful execution of the first command, the

ELN 3 corresponds now to a different file.

External commands:

CliFM is not limited to its own set of internal commands, like open, sel, trash, etc., but it can run any external command as well, provided external commands are allowed (see the −x option, the 'ext' command, or the configuration file). By beginning the external command by a colon or a semicolon (':', ';') you tell CliFM not to parse the input string, but instead letting this task to the system shell. However, bear in mind that CliFM is not intended to be used as a shell, but as the file manager it is.

Terminal emulators and non-ASCII characters:

It depends on the terminal emulator you use to correctly display non−ASCII characters and characters from the extended ASCII charset. If, for example, you create a file named "ñandú" (the Spanish word for 'rhea'), it will be correctly displayed by the Linux console, Lxterminal, and Urxvt, but not thus by Xterm or Aterm.

.Xresources:

CliFM will create *$HOME/.Xresources*, if it doesn't already exist, for keybindings to work correctly. However, some (and even all) of these keybindings might not work in some terminals, though they do work fine on the console (TTY), xvt−like terminal emulators like Urxvt and Aterm, and xterm−like ones. However, keybinding can be edited freely to make them work on any terminal emulator.

Spaces and escape codes:

When dealing with file names containing spaces, you can use both single and double quotes (ex: "this file" or 'this file') plus escape sequences (ex: this\ file).

Starting path:

By default, CliFM starts in the current working directory. However, you can always specify a different path by passing it as positional parameter. Ex: clifm /home/user/misc. You can also permanently set up the starting path in the CliFM configuration file. If the RestoreLastPath option is set to true, CliFM will start instead in the last visited directory (and in the last used workspace), unless the starting path (and optionally the workspace number) is specified via command line.

Default profile:

CliFM's default profile is "default". To create alternative profiles use the −P command line option or the 'pf add' command (see above).

23. FILES

CONFIGURATION FILE

The configuration file is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/clifmrc*. It will be copied from *DATADIR/clifm* (usually */usr/share/clifm*), and if not found, it will be created anew with default values. Here you can permanently set up CliFM options, add aliases and some prompt commands (which will be executed immediately before each new prompt line). Just recall that in order to use prompt commands you must allow the use of external commands. See the −x

option and the 'ext' command above.

A description for each option in the configuration file can be found in the configuration file itself.

PROFILE FILE

The profile file is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/profile.cfm*. In this file you can add those commands you want to be executed at startup. You can also permanently set here some custom variables, ex: 'dir="/path/to/folder"'. This variable may be used as a shortcut to that folder, for instance: 'cd $dir'. Custom variables could also be temporarily defined via the command prompt: Ex: user@hostname ~ $ var="This is a test". Temporary variables will be removed at program exit.

KEYBINDINGS FILE

The keybindings file is *$XDG_CONFIG_HOME/clifm/keybindings,cfm*. It will be copied from *DATADIR/clifm* (usually */usr/share/clifm*), and if not found, it will be created anew with default values. This file is used to specify the keyboard shortcuts used for some ClifM's functions. The format for each keybinding is always "keyseq:function", where 'keyseq' is an escape sequence in GNU emacs style. A more detailed explanation can be found in the keybindings file itself.

PLUGINS DIRECTORY

The directory used to store programs or scripts pointed to by actions (in other words, plugins) is *DATADIR/clifm/plugins* (usually */usr/share/clifm/plugins*). To edit these plugins copy them to *$XDG_CONFIG_HOME/clifm/plugins* and edit them to your liking. Plugins in this local directory take precedence over those in the system one.

COLORS DIRECTORY

This directory, *$DATADIR/clifm/colors*, contains available color schemes as files with a *.cfm* extension. You can create as many color schemes as you want by just dropping them in this directory. The default color scheme file (default.cfm) could be used as a guide. You can copy these color schemes to the local colors directory (*$XDG_CONFIG_HOME/clifm/colors*) and edit them to your liking. Color schemes in the local colors directory take precedence over those in the system directory.

ACTIONS FILE

The file used to define custom actions is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/actions.cfm*. It will be copied from *DATADIR/clifm* (usually */usr/share/clifm*), and if not found, it will be created anew with default values.

MIMELIST FILE:

The mime list file is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/mimelist.cfm*. It is a list of file types and file extensions and their associated applications used by *lira*. It will be copied from *DATADIR/clifm* (usually */usr/share/clifm*).

BOOKMARKS FILE

The bookmarks file is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/bookmarks.cfm* Just the list of the user's bookmarks used by the bookmarks function.

HISTORY FILE

The history file is *~/.config/clifm/profiles/PROFILE/history.cfm*. A list of commands entered by the user and used by the history function.

COMMANDS LOG FILE

The commands log file is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/log.cfm*. The file contains a series of fields separated by a colon in the following way: 'date:user:current_dir:command. All commands executed as external will be logged.

MESSAGES LOG FILE

The messages log file is *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/messages.cfm*. A file containing a list of system messages, either errors, warnings, or simple notices. The messages log

format is: "[date] message".

KANGAROO DATABASE

The directory jumper database is stored in *$XDG_CONFIG_HOME/clifm/profiles/PROFILE/jump.cfm*.

NOTE: If *$XDG_CONFIG_HOME* is not set, *$HOME/.config/* is used instead.

24. EXAMPLES

NOTE: Always try TAB. TAB completion is available for many things

NOTE2: Suggestions for possible completion will be printed next to the text typed so far. To accept the given suggestion, press Left (or Alt-f to accept onlt the first suggested word). Otherwise, the suggestion is just ignored

Get help: **F1**: manpage **F2**: keybindings **F3**: commands

- Change directory to */etc*

  /etc

- Change-to/open the directory/file whose ELN is 5 in the current directory:

  5

TIP: Press TAB to make sure 5 is the file you want. If the suggestions system is enabled, just pay attention to the suggestion. Press Left to accept the given suggestion

- Open *myfile.txt* (with the default associated application):

  myfile.txt

- Open *myfile.txt* using *vi*:

  myfile.txt vi (or **vi myfile.txt**)

- Open the file whose ELN is 24 in the background:

  24&

- Jump to *~/media/data/docs/work/mike/xproject*:

  j xproj

NOTE: This depends however on the database ranking. For more accuracy: 'j mike xproj'

- Go back to the directory you came from:

  b (or **Shift+left** or **Alt+j**)

NOTE: Enter **f**, or press **Shift+right** or **Alt+k** to go back to the first directory

- Create a new file named *myfile* and a new directory named *mydir*:

  n myfile mydir/

NOTE: Since CliFM is integrated to the system shell, you can also use any of the shell commands you usually use to create new files. Ex: 'touch myfile' or 'nano myfile'

- Change to detail/long view mode:

  Alt+l

- Print the properties of the file whose ELN is 4:

  p4

- Reprint the list of files in the current directory:

  rf

- Select all c files in the current directory:

  s *.c

- Select multiple files in the current directory by ELN:

  s 1-4 8 19-26

- List selected files:

  sb

- Deselect a few files:

  ds

- Tag all PDF files in the current directory as *mypdfs*:

  ta *.pdf :mypdfs

- Print the file properties of all files tagged as *mypdfs*:

  p t:mypdfs

- Search for all PDF files in the current dirctory:

  /*.pdf

- Create a directory named *mydir* and cd into it:

  n mydir/ && mydir

- Copy selected files into the current directory:

  c sel

- Remove all selected files:

  r sel

NOTE: To remove files in bulk via a text editor use the *rr* command.

- Rename the file whose ELN is 12:

  m12

- Bookmark *mydir*:

  bm add mydir

- Open the bookmarks screen. Once there, enter the bookmark ELN (1 ... n) or its hotkey ([xx]) to open it:

  bm (or **Ctrl+b**)

- Switch to workspace 2:

  ws2 (or **Alt+2**)

- View and/or edit the configuration file:

  edit (or **F10**)

- Change to profile *test*:

  pf set test

- Show hidden files:

  hf on (or Alt+.)

List available actions/plugins:

  actions

- Want file previews?

  - (yes, just a dash)

NOTE: This runs the plugin *fzfnav.sh*. Take a look at the manpage for needed dependencies

- Want icons?

  icons on

NOTE: Recall to install **icons-in-terminal** before

- I'm tired, quit:

  q

There is a lot more you can do, but this should be enough to get you started.

EXIT STATUS

CliFM returns the exit status of the last executed command:

0 Successful execution

1 Error

CONFORMING TO

CliFM is C99 compliant, and, if compiled with the _BE_POSIX flag, it is POSIX.1-2008 compliant as well. If not, just a single non-POSIX function is used: **statx**(2) (Linux specific), to get files birth time.

BUGS AND FEATURE REQUESTS

Report at <https://github.com/leo−arch/clifm/issues>

AUTHOR

L. M. Abramovich